1 Introduction

1.1 Purpose
The purpose of this document is to define a specific implementation of DragonToolkit based on requirements given in the requirements document. Any changes made to the design in the future shall be reflected as a revision to this document.

1.2 Scope
This document is intended to be used by developers to implement DragonToolkit.

1.3 Definitions
- **Website**: collection of pages allowing the user to access the tool wizards and other user options
- **Wizard**: guided tool to help user to generate customized component code
- **Browser**: a software application for retrieving, presenting, and traversing information resources on the World Wide Web (WWW)
- **Code**: the code is the set of instructions for a computer governed by a particular language
- **Cross platform code**: the set of instructions for computer that is portable and can be ran on most if not all operating systems; this means that a particular file or application (or even certain hardware) can be used on more than one computer platform
- **CSS**: cascading style sheets are declarations that describe how a document should be presented on the Web; CSS can be written for different media, can be part of an HTML document, or can be a separate file that is applied to multiple HTML documents
- **Frame**: a frame is a method of displaying multiple HTML documents on one page
- **Mark up Language**: a markup language is a system for annotating a text in a way which is syntactically distinguishable from that text
- **HTML**: HyperText Markup Language: predominant markup language for web pages that provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, etc as well as for links, quotes, and other items
- **WYSIWYG**: What You See Is What You Get: term used in computing to describe a system in which content displayed during editing appears very similar to the final output
- **Web Application**: a computer software application that is hosted in a browser-controlled environment or coded in a browser-supported language and reliant on a common web browser to render the application executable.
- **URL**: Uniform Resource Locator: global address of documents and other resources on the World Wide Web.
2 Survey of Technologies

2.1 Web Application
The main web application must be accessible through a web browser. It will provide access to the tool wizards.

2.1.1 Client
- Javascript

2.1.2 Server
- XHTML/CSS
- Java
- JSP
- Tomcat
- MySQL

2.2 Wizards
- Javascript
- JQuery

2.3 Output Code
The code generated by the wizards will require these technologies.

- Javascript

It can use other libraries as needed.

- JQuery
- MySQL
3  Code Architecture

3.1  Web Application
The main application program will be written using XHTML 1.0 and CSS. This will provide the overall structure and appearance of the application.

Functional aspects of the application will be implemented with Java and JSP.

3.2  Wizards
The wizards will follow an object-oriented design. Each wizard instance will be a subclass of an abstract wizard class.

![Diagram](image)

**Figure 3-1**

3.2.1  AbstractWizard
The AbstractWizard class represents a tool wizard. Specific wizards will inherit and implement this class.

In order to draw the wizard, the bridge pattern will be used to attach a renderer to the wizard.
3.2.1.1 Data
- ArrayList&lt;WizardPage&gt; - list of pages in wizard
- WizardRenderer – an object capable of rendering the wizard
- Title – string containing title of the tool

3.2.1.2 Methods
- next() - navigate to next page
- previous() - navigate to previous page
- save() - Save current state
- generate Code() - Generates code for that specific wizard
- render() - Create HTML form for the specific wizard

3.2.2 WizardPage
The WizardPage class specifies a specific page of the wizard. This includes data fields.

3.2.3 Field
The abstract Field class represents a component of a WizardPage. These input fields are rendered on the page by AbstractWizardRenderer.

3.2.3.1 TextField
TextField is a subclass of Field. It holds a string value.

3.2.3.2 FieldList
FieldList is a subclass of Field. It represents a collection of specialized input fields.

3.2.4 AbstractWizardRenderer
The AbstractWizardRenderer class provides an abstract interface for displaying the wizard.

3.2.5 HTMLWizardRenderer
The HTMLWizardRenderer class implements AbstractWizardRenderer. It builds an HTML form to display the wizard pages.
3.2.6 DBanner
The DBanner wizard is an application that allows for rotating a banner that displays multiple (linked) html pages. Page timing can be user defined. Playback buttons will also allow users to play/pause or seek to specific html pages.

3.2.6.1 Data
width- Integer for the size in the x direction of the banner
height- Integer for the size in the y direction of the banner
display_time- Integer for the amount of time to display the banner
pages- List of different banners to display

3.2.6.2 Methods
generatePage()- Generate the page of the banner
generateCode()- Generates code for the DBanner page

\[
\begin{array}{|c|}
\hline
\text{AbstractWizard} \\
pages: \text{List<WizardPage>} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
\text{DBanner} \\
width: \text{int} \\
height: \text{int} \\
display\_time: \text{int} \\
pages: \text{ArrayList<String>} \\
generatePage(url: void): \text{String} \\
\hline
\end{array}
\]

Figure 3-2

3.2.6.3 Output

\[
\begin{array}{|c|}
\hline
\text{Tool Body} \\
toolBody: \text{String} \\
loadPage(): \text{void} \\
previous(): \text{void} \\
next(): \text{void} \\
in1(): \text{void} \\
timer(): \text{void} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
\text{External Page} \\
pageBody: \text{String} \\
\hline
\end{array}
\]

Figure 3-3
3.2.6.3.1 External Pages
Individual pages that will be loaded by the DBanner player component.

3.2.6.3.2 Tool Body
Core Javascript code that runs DBanner; HTML structure for laying out the banner; Playback buttons; Playback buttons invoke the methods listed below;

loadPage( var page ) – Loads the specified banner page
previous() – Loads the previous page
next() – Loads the next page
init() – Loads the external HTML pages specified in the wizard; Initialize slideshow
timer() – Calls the next page on user specified interval
3.2.7 DBannerZoom

DBannerZoom provides a magnified viewing window to display images. The user moves the window over the image using the mouse cursor, which magnifies the image at that point and displays the zoomed window at the cursor location.

3.2.7.1 Data

- strLocationURL: A string for the location of the picture to use
- percentSize: A double to increase the size of the zoomed picture
- windowHeight: An integer for the size of the zoomed window, in the Y direction
- windowWidth: An integer for the size of the zoomed window, in the X direction
- strLocationURLLarge: A string representing the location of the new enlarged image

3.2.7.2 Methods

generateCode(): generates the code necessary to create the DBannerZoom frame.

![Diagram](image)

3.2.7.3 Output

![Diagram](image)
3.2.7.3.1 Tool Body
Core Javascript code that runs DBannerZoom; HTML structure for laying out the images

3.2.7.3.2 Zoom Plugin
Modified JQuery library
3.2.8 DImageNav

Creates graphical buttons from an image. The image is split into pieces, each of which is used as the background for a button. This allows the user to easily create a menu bar with a consistent theme.

3.2.8.1 Data

strURL- A string for the url of the image
numButtons- The number of buttons to divide the image into

3.2.8.2 Methods

divideImage() - divides an image into several smaller images.
generateImageLink() - creates a link to each of the smaller images created from the original
generateCode() - generates the code necessary to create the DImageNav bar

3.2.8.3 Image

3.2.8.3.1 Methods

Img- The data associated with an image

---

**Figure 3-6**

---

3.2.8.4 Output

---

**Figure 3-7**
3.2.8.4.1  Tool Body
Javascript code that runs DImageNav. HTML for buttons.
Splits up the picture into a finite set of images to be placed on buttons.

Split() - Splits picture into equal pieces
Init() - Load split image onto each button
loadPic() - Get picture

3.2.8.4.2  Image Slice
HTML displaying each part of the main image.
3.2.9 DLearn
Create customized flash cards that are presented on a web page. Flashcards may contain text and/or pictures. Various viewing layouts may be set by the user.

3.2.9.1 Data
strTitle- A string for the title of a card
strInstructions- A string for displaying instructions
listOfCards- A list of cards used for the program
Front- Front of card corresponding to a string.
Back- Back of card representing a string

3.2.9.2 Methods
generateCard() - Creates a card and places it into the list of cards
generateCode() - generates the code necessary to create the dLearn page.

3.2.9.3 Card
The Card class implements DLearn by providing the cards for the wizard. Each card contains a front and back.

![Diagram of Card class](image)

3.2.9.4 Output

![Diagram of Tool Body and Flash Cards](image)
3.2.9.4.1 Tool Body
Javascript code that creates flash cards. HTML structure for the layout of each card, with buttons. Each card contains either an image or text.

imageOrText()- Determines if the input for the card is either an image or text.
addFront()- Add to front of card
addBack()- Add to back of card
hide()- Makes components on card invisible.
show()- Makes components on card visible.
next() - Load next flash card.

3.2.9.4.2 Flash Card
HTML displaying each flash card
3.2.10 DLinkDetective
Automatically check for dead/invalid links by traversing to each individual link and returning a detailed report of any issues to the developer.

3.2.10.1 Data
strLocationURL- A string that represents the domain or website name to crawl
LocationImage- Boolean that checks if the user has enabled the option to scan for images on the website
hyperlinks- Boolean that checks for hyperlinks, if selected

3.2.10.2 Methods
generateCode() - generates the code necessary to create the DLinkDetective page.

3.2.10.3 Outputs

3.2.10.3.1 Tool Body
Javascript code to validate links found on a page.

generateReport()- Report created after all links have been checked.
checkURL()- Checks URL for HTTP status message.
3.2.11 DBreadCrumbs
A trail of hyperlinks is automatically generated that allows a user to track their current location inside a website. Also, the user may jump to any page previously visited within that site.

3.2.11.1 Data
- strURL: A string representing the URL of the site
- style: An integer value representing the breadcrumb style
- hexColorCode: A string value for the color in hex

3.2.11.2 Methods
- generateStyleDefn(): A method that returns a string of the style of breadcrumb
- generateCode(): generates the code necessary to create the DBreadCrumbs page.

3.2.11.3 Output
- ToolBody
  - ToolBody: String
  - setStyle(): void

3.2.11.3.1 Tool Body
Generates hyperlinks of pages visited by the user
- setStyle(): Bread crumb trail style is set to user's preferences.
3.2.12 DFormBuilder
Add forms with customized input fields to a page. Validation options will be included to filter content for malicious inputs and to verify the integrity of the data.

3.2.12.1 Data
Name- A string for the name of the field property
Submit_action- A URL that points to a page for when the action to occur once the form is complete
Field_list- The list of possible fields found on the form
Field_name- The name of each filed that can be placed on the form
Field_type- An integer representing the type of field going to be used on the form

3.2.12.2 Methods
generateField()- Create the field(s) necessary
generateAction()- Create the action to go along with the form
generateCode()- generates the code necessary to create DFormBuilder

3.2.12.3 Field
Input fields that are rendered on the page.

3.2.12.3.1 RadioButton
RadioButton is a sublcas of Field. It will contain a value to represent which option was chosen from the list.

3.2.12.3.2 TextField
TextField is a subclass of Field. It holds a string value.

3.2.12.3.3 CheckBox
CheckBox is a subclass from Field. A checkbox will hold a Boolean value to determine if the option has been selected.

3.2.12.4 Button
Button is a subclass from Field. A button will be used for an action to move from one window to another.
3.2.12.5 Output

![Figure 3-15](image)

<table>
<thead>
<tr>
<th>Tool Body</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toolBody : String</code></td>
</tr>
</tbody>
</table>

| submitAction() : String |
| answers() : String |
| sendReport() : String |

3.2.12.5.1 Tool Body

Javascript code to create a new form. HTML structure for the layout of the form.

submitAction()- The action after the user submits the form
answers() - The input list from the users input on the form
sendReport()- Outputs report
3.2.13 DHangman
Generates a hang man game using a given word list from the webmaster. The user will then be prompted to enter letters; every wrong guess causes a new part of the 'victim' to appear on the scaffold. A successful guess will cause the letter to appear in the appropriate position on-screen. The game ends when the word has been filled in or the 'victim' dangling on the noose has been completed. The user is able to restart the game with a new word. The word list is automatically generated from the website.

3.2.13.1 Data
enterPhrase- A Boolean that is true if the checkbox for manually entering phrases is selected
genPhrase- A Boolean that is true if the checkbox for automitcally finding phrases on a website is selected
hexCodeColor- A string that represents the color, in hex
strImagePath- A string that is used for finding the path of the image for hangman
wordlist- A list of words that are either manually entered in or automatically generated from the website
hintList- Corresponding to the words, a list of hints that can be entered in to assist the user in solving the puzzle

3.2.13.2 Methods
generateCode() - generates the code necessary to create the DHangMan page

3.2.13.3 UtilityLibrary

3.2.13.3.1 Methods
generateWordList()- The list of generated words that the program finds automatically, assuming that option was checked previously
UtilityLibrary()- The creation of the utility library

Figure 3-16
3.2.13.4 Output

<table>
<thead>
<tr>
<th>Tool Body</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toolBody : String</code></td>
</tr>
<tr>
<td><code>currentWord : String</code></td>
</tr>
</tbody>
</table>

- `init() : void` – Initialize the game
- `drawBodyPart() : void` – Draw the game graphics
- `checkGuess() : boolean` – Validate the player’s guessed word
- `drawGuess() : void` – Draw the guessed letters
- `isComplete() : boolean` – Current status of the game
- `generateWordList() : String[]` – Return a list of words from the website
- `isDead() : boolean` – Indicates a game over

Figure 3-17

3.2.13.4.1 Tool Body

Javascript code that creates a hang man game

- `init()` – Initialize the game
- `drawBodyPart()` – Draw the game graphics
- `checkGuess()` – Validate the player’s guessed word
- `drawGuess()` – Draw the guessed letters
- `isComplete()` – Current status of the game
- `generateWordList()` – Return a list of words from the website
- `isDead()` – Indicates a game over
3.2.14 DSurvey
Web-based component that enables the creation of surveys with customized questions. The wizard will guide the user during the creation process allowing them to select question types. The application will provide templates for creating effective surveys.

3.2.14.1 Data
Title- A string for the title of the survey
Description_instructions- A string for instructions on the survey
Question_List- A series of questions in a list that can be used in the survey
Question- The specific question that is selected and displayed
Question_type- An integer for the index of which type of question

3.2.14.2 Methods
generateField()- Create the field(s) necessary
generateAction()- Create the action to go along with the form
generateCode()-generates the code necessary to create DSurvey

3.2.14.3 Output

3.2.14.3.1 Tool Body
Javascript code that allows users to create customized surveys.

init() – Initialize survey components
submitResponse() – Handles submission of the survey
3.2.14.4 Data Model of Survey

3.2.14.4.1 Question
An abstract class that covers all the things that are shared by all questions.

setPrompt() - sets the current prompt of the question
getPrompt() - returns the prompt of this question.
answer() - gets the user's answer for the current question
menu() - presents the choices for the current question

3.2.14.4.2 Essay
A class for essay questions.

setResponse() - takes in a string for the user's response.
getResponse() - returns the current response.
answer() - gets the user's answer for the current question
menu() - presents the choices for the current question

3.2.14.4.3 MultiChoice
A class for multiple-choice questions

setResponse() - takes in a string for the user's response.
getResponse() - returns the current response.
setLeft() - sets the list of items on the left side of the question.
getLeft() - gets the list on the left side.
setRight() - sets the list of items on the right side of the question (to be matched to the list on the left)
getRight() - gets the list of questions on the right side of the question.
answer() - gets the user's answer for the current question
menu() - presents the choices for the current question

3.2.14.4.3.1 Matching
A class for 'matching' questions. These are questions where the user is presented with a list of options and has to match them to the corresponding entry in a second list of answers.

setResponse() - takes in a string for the user's response.
getResponse() - returns the current response.
setCorrect() - sets the correct answer for the current question
getCorrect() - gets the correct answer for the current question
setChoices() - sets the choices for the current question
getChoices() - gets the list of choices for the current question
answer() - gets the user's answer for the current question
menu() - presents the choices for the current question
Figure 3-20
3.2.14.5 DWordSearch
Generates a word search from a list of words provided by the webmaster. The program will automatically place the words on the playing field, introduce random letters to camouflage them, and then allow the user to select words by highlighting them with the mouse. The game ends when the user selects all the words in the list.

The user is able to restart the game with a new word set.

The word list is automatically generated from the website, where the webmaster may choose words from the generated list and create their own list to use in the word search.

3.2.14.6 Data
Title- String of the title of the word search
Time_limit- An integer for the time limit for gameplay. Once expired, the game has ended and words cannot be selected
Difficulty- Integer that can be set to choose the level of difficulty for the game
wordlist- A list of strings containing the words that will be displayed in the wordsearch for the user to find
words_generated- The list of words that were automatically generated from a specific website

3.2.14.7 Methods
generateCode() - generates the code necessary to create the DWordSearch page

3.2.14.8 UtilityLibrary
3.2.14.8.1 Methods
GenerateWordList()- The list of words that was actually generated automatically from a predefined website
UtilityLibrary()- The creation of the utility library

---

**Figure 3-21**

---
3.2.14.9 Output

<table>
<thead>
<tr>
<th>Tool Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>- siteURL : String</td>
</tr>
<tr>
<td>- moduleName : String</td>
</tr>
<tr>
<td>- getDifficulty() : void</td>
</tr>
<tr>
<td>- undoHighlightLetters() : void</td>
</tr>
<tr>
<td>- removeWordFromList() : void</td>
</tr>
<tr>
<td>- addWordToList() : void</td>
</tr>
</tbody>
</table>

Figure 3-22

3.2.14.9.1 Tool Body

Javascript code that creates the grid of letters and the word list associated with it.

NewGame() starts a new game with a new word list
Difficulty() controls the level of difficulty for the game
Undo() removes the highlighting off the previous selected letters
Highlight() highlights the letters the user selects to find a word
StrikeThrough() strikes off the word that the user has finished
3.2.15 DMap

A component that provides a map with directions to a business or other location. Along with the map, the developer may also choose to add total travelled distance and time. Since routes may vary between different mapping services, the developer may choose between various sources.

3.2.15.1 Data

strAddress - The entire address, as a string of the location

totalTravel - A Boolean value that depicts if the total travel in time should be displayed along with the map

totalDistance - A Boolean value that shows the total distance from an address, if selected

engineSelected - An integer representing an index for which map engine to use

3.2.15.2 Methods

getCurrentLocation() - determines the user’s current address from the fields entered

generateCode() - generates the code necessary to create the DMaps page.

Figure 3-23

3.2.15.3 Output

Tool Body

userAddress : String
locationAddress : String

plotRoute(userAddress : String, locationAddress : String) : void
updateDisplay() : void
init() : void

Figure 3-24
3.2.15.3.1 Tool Body
Javascript code that creates a map with driving directions.

plotRoute(userAddress : String, locationAddress : String) – Provide directions from user to location
updateDisplay() – Updates the current display
init() – Initialize the map display
4 Design Features

4.1 Web Application

4.1.1 Login
When the user submits login information, a login request is sent to the Tomcat server. This login information is verified in the user database. The server then sets the session status to “logged in” and the page is reloaded.

4.1.2 Logout
When the user clicks “Logout”, a logout request is sent to the server. The session status is changed to “logged out” and the page reloads.
4.1.3 Select Wizard
The user clicks on the Tools menu. The user browses through the categories and clicks on a Tool. The server loads the appropriate wizard class and builds the page, then returns it to the browser.

4.2 Wizards
4.2.1 Generate Code
When the user clicks the “Generate Code” button of the wizard, a request is sent to the server for the output page. The server calls the generateCode() method of the wizard class and inserts the resulting code into the page, which it then returns to the browser.
5  Summary

5.1 Advantages of Design
By abstracting the wizard class, new wizards can easily be added. Also, by using the bridge pattern, each wizard class is able to be concisely defined as a set of required fields without regard to how these fields are presented. A separate renderer can then be used to display the wizard.

5.2 Disadvantages of Design
The abstraction of the wizard class and wizard renderer creates a more complex design.