The ePortfolio Generator™

Software Design Description

Author: Rahul Agasthya

Debugging Enterprises™

October, 2015

Version 1.0

Abstract: This document describes the software for The ePortfolio Generator™, an application for generating student ePortfolio Web sites.

Based on IEEE Std 1016™-2009 document format

1 Introduction

A well designed, informative and inviting personal Web site is an essential tool for any University student. A good personal site represents the student online, providing a succinct summary of the student's academic career accented with personal style.

The process of making a good ePortfolio can, in part, be automated, which is the point of this project. Using this tool, a well-designed, colorful, informative, site can be generated using content entered by a student user, where they may customize the site in all sorts of ways.

Note that this document format is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

The document is to serve as the blueprint for the construction of ***The ePortfolio Generator^{TM}*** application. This design will use UML class diagrams to provide complete detail regarding packages, classes, instance variables, class variables, and method signatures needed to build the application. In addition, UML Sequence Diagrams will be used to specify object interactions post-initialization of the application.

1.2 Scope

The ePortfolio Generator^{TM} is an application that the students of Stony Brook University use. This framework (or a set of frameworks), should be designed and constructed to be used to make this application a living piece of software that can easily accommodate change. This design contains design descriptions for the development of both the framework and the application. Note that Java is the target language for this software design.

1.3 Definitions, acronyms and abbreviations

Class Diagram – A UML document format that describes classes graphically. Specifically, it describes their instance variables, method headers, and relationships to other classes.

IEEE – Institute of Electrical and Electronics Engineers, the "world's largest professional association for the advancement of technology."

Framework – In an Object Oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

Java – A high level programming language that uses a virtual machine layer between the Java application and the hardware to provide program portability.

HTML – Short for Hyper Text Markup Language, is a standard markup language to create web pages.

CSS – Short for Cascading Style Sheets, is a style sheet language used for describing the presentation of a document written in a markup language.

Digication – A Web based software product currently used by Stony Brook University for students to create ePortfolios.

ePortfolio – Short for Electronic Portfolio, it is typically a Web site that summarizes ones work, as in an academic setting.

GUI – Short for Graphical User Interface, it includes graphical components and visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

1.4 References

IEEE Std 830™ – 1998 (R2009) – IEEE Standard for Information Technology – System Design – Software Design Descriptions.

The ePortfolio Generator™ SRS – Debugging Enterprises' Software Requirements Specification for the ePortfolio Generator application.

The Java Documentation™ – Oracle's documentation website for Java and JavaFX classes, packages and interfaces.

1.5 Overview

This Software Design Description document provides a working design for the ePortfolio Generator™ software application as described in the ePortfolio Generator™ Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to specify how the classes should be constructed. Section 4 provides the deployment information like file structures and formats to use. Section 6 provides a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using VioletUML™ editor.

2 Package-Level Design Viewpoint

As mentioned, this design will encompass the ePortfolio Generator application to be used in its construction. In building both we will heavily rely on the Java API to provide services. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

2.1 ePortfolio Generator Application Overview

The ePortfolio Generator Application will be designed and developed using the components which is described by Figure 1. Also, the figure shows the places of all classes in packages.

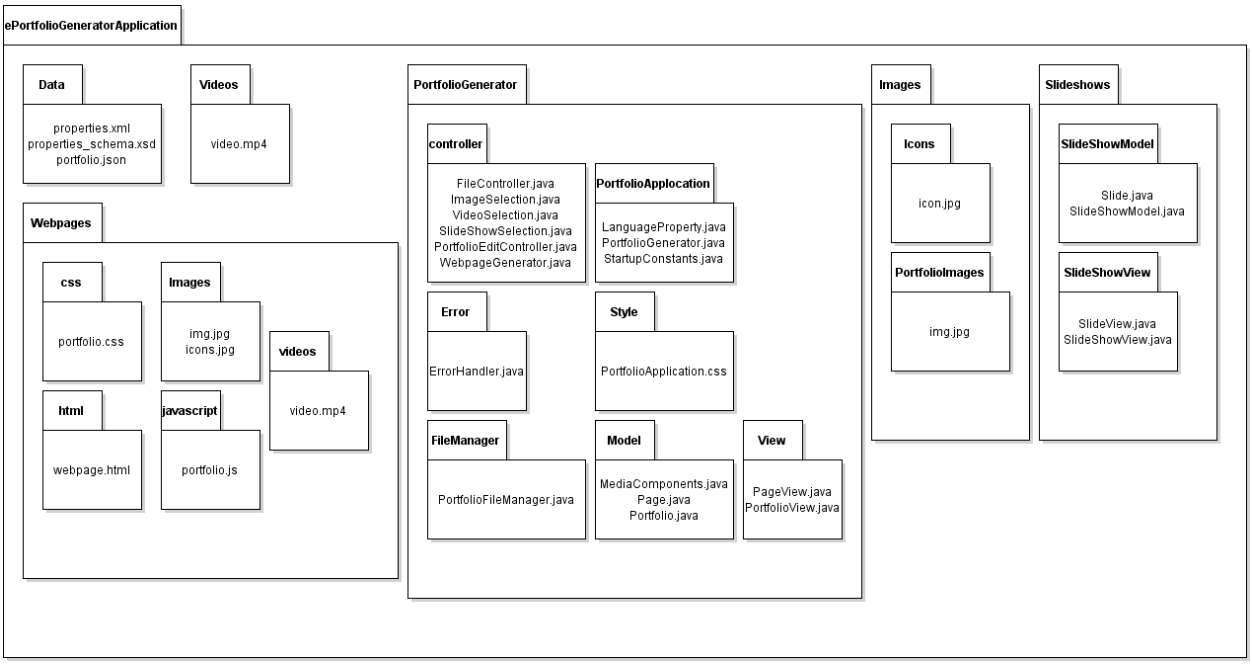

Figure 1 Design Packages Overview

2.2 Java API Usage

The Application will be developed using Java Programming Language, HTML and JavaScript, and will make use of the following classes specified in Figure 2.

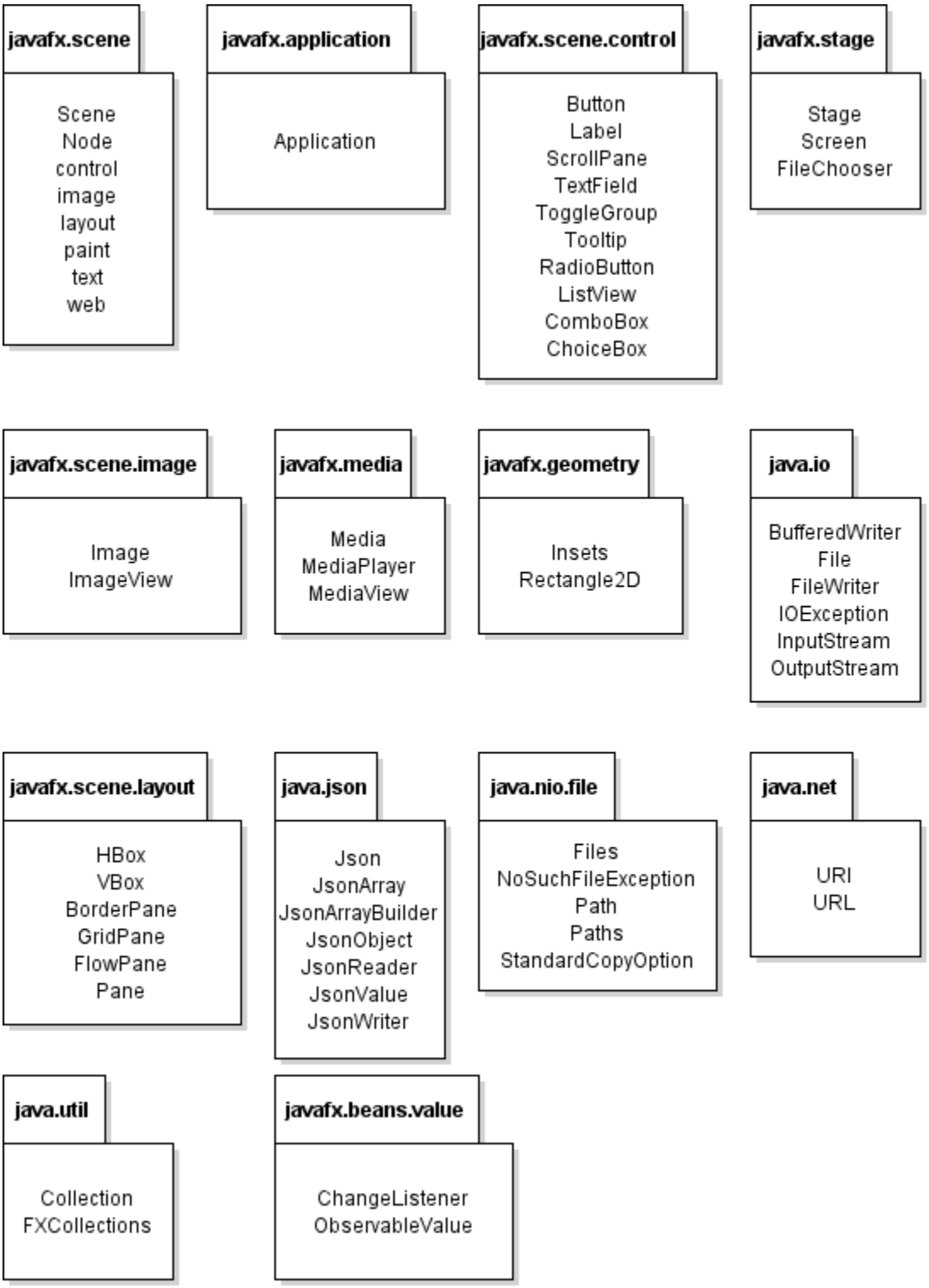

Figure 2 Java API Classes and Packages to be used

2.3 Java API Usage Descriptions

Tables 1-14 below summarize how each of these classes will be used.

Class/Interface	Use
Scene	For setting the scene of the window. It is a graph data structure.
Node	Each item in the Screen graph is a Node.
control	For every UI Control in the application.
image	For loading and displaying images.
layout	Provides classes to support user interface layout.
paint	Provides classes for colors and gradients used to fill shapes and backgrounds when rendering the scene graph.
text	Defines a node to display text.
web	Provides means for loading and displaying Web content.

Table 1 Uses for classes of the Java API's javafx.scene package.

Class/Interface	Use
Application	The entry point for JavaFX applications.

Table 2 Uses for classes of the Java API's javafx.application package.

Class/Interface	Use
Button	A simple Button control.
Label	A non-editable text control.
ScrollPane	Provides a scrolled, clipped viewport of its content.
TextField	Text input component that allows the user to enter text.
ToggleGroup	Provides a control of selection such that the items within this group can be selected one at a time.
Tooltip	Common UI elements which are typically used for showing additional information about the control, when hovered by the mouse.
RadioButton	A series of items where only one item can be selected.
ListView	Allows the display of horizontal or vertical list of items from which the user may select, or with which the user may interact.
ComboBox	A popup list is shown to users providing then with a choice that they may select them.
ChoiceBox	Presents a small set of predefined choices from which they may choose.

Table 3 Uses for classes of the Java API's javafx.scene.control package.

Class/Interface	Use
Stage	Top level JavaFX container which holds the Scene.
Screen	Characteristics of a graphics destination and the size of the stage.
FileChooser	Support for standard platform file dialogs

Table 4 Uses for classes of the Java API's javafx.Stage package.

Class/Interface	Use
Image	This class represents the graphical images and is used for loading images from a specified URL.
ImageView	A Node used for painting images loaded with the **Image** class.

Table 5 Uses for classes of the Java API's javafx.scene.image package.

Class/Interface	Use
Media	Represents a media resource.
MediaPlayer	Provides controls for playing the media
MediaView	A view for the **Media** being played by the **MediaPlayer**.

Table 6 Uses for classes of the Java API's javafx.media package.

Class/Interface	Use
Insets	Sets inside offsets for the four sides of a rectangular area.
Rectangle2D	Describes the bounds of an object.

Table 7 Uses for classes of the Java API's javafx.geometry package.

Class/Interface	Use
BufferedWriter	Writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.
File	An abstract representation of file and directory pathnames.
FileWriter	Convenience class for writing character files.
IOException	Signals that an I/O exception of some sort has occurred.
InputStream	This abstract class is the superclass of all classes representing an input stream of bytes.
OutputStream	This abstract class is the superclass of all classes representing an output stream of bytes.

Table 8 Uses for classes of the Java API's java.io package.

Class/Interface	Use
HBox	HBox lays out its children in a single horizontal row.
VBox	VBox lays out its children in a single vertical column.
BorderPane	BorderPane lays out children in top, left, right, bottom, and center positions.
GridPane	GridPane lays out its children within a flexible grid of rows and columns.
FlowPane	FlowPane lays out its children in a flow that wraps at the flowpane's boundary.
Pane	Base class for layout panes which need to expose the children list as public so that users of the subclass can freely add/remove children.

Table 9 Uses for classes of the Java API's javafx.layout package.

Class/Interface	Use
Json	Contains static methods to create JSON readers, writers, builders, and their factory objects.
JsonArray	Represent data types for values in JSON data.
JsonArrayBuilder	Create an object model or an array model in memory by adding values from application code.
JsonObject	Represent data types for values in JSON data.
JsonReader	Reads JSON data from a stream and creates an object model in memory.
JsonValue	Writes an object model from memory to a stream.
JsonWriter	Represent data types for values in JSON data.

Table 10 Uses for classes of the Java API's java.json package.

Class/Interface	Use
Files	This class consists exclusively of static methods that operate on files, directories, or other types of files.
NoSuchFileException	Checked exception thrown when an attempt is made to access a file that does not exist.
Path	An object that may be used to locate a file in a file system.
Paths	This class consists exclusively of static methods that return a Path by converting a path string or URI.
StandardCopyOperation	Defines the standard copy options.

Table 11 Uses for classes of the Java API's java.nio.file package.

Class/Interface	Use
URI	Represents a Uniform Resource Identifier (URI) reference.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.

Table 12 Uses for classes of the Java API's java.net package.

Class/Interface	Use
Collection	The root interface in the collection hierarchy.
FXCollections	Utility class that consists of static methods that are 1:1 copies of **java.util.Collections** methods.

Table 13 Uses for classes of the Java API's java.util package.

Class/Interface	Use
ChangeListener	It is notified whenever the value of an ObservableValue changes.
ObservableValue	It is an entity that wraps a value and allows to observe the value for changes.

Table 14 Uses for classes of the Java API's javafx.beans.value package.

3 Class-Level Design Viewpoint

As mentioned, this design will encompass the ePortfolio Generator Application. The following UML Class Diagrams reflect this. Note that due to the complexity of the project, the class diagrams are presented using a series of diagrams going form overview diagrams down to the detailed ones.

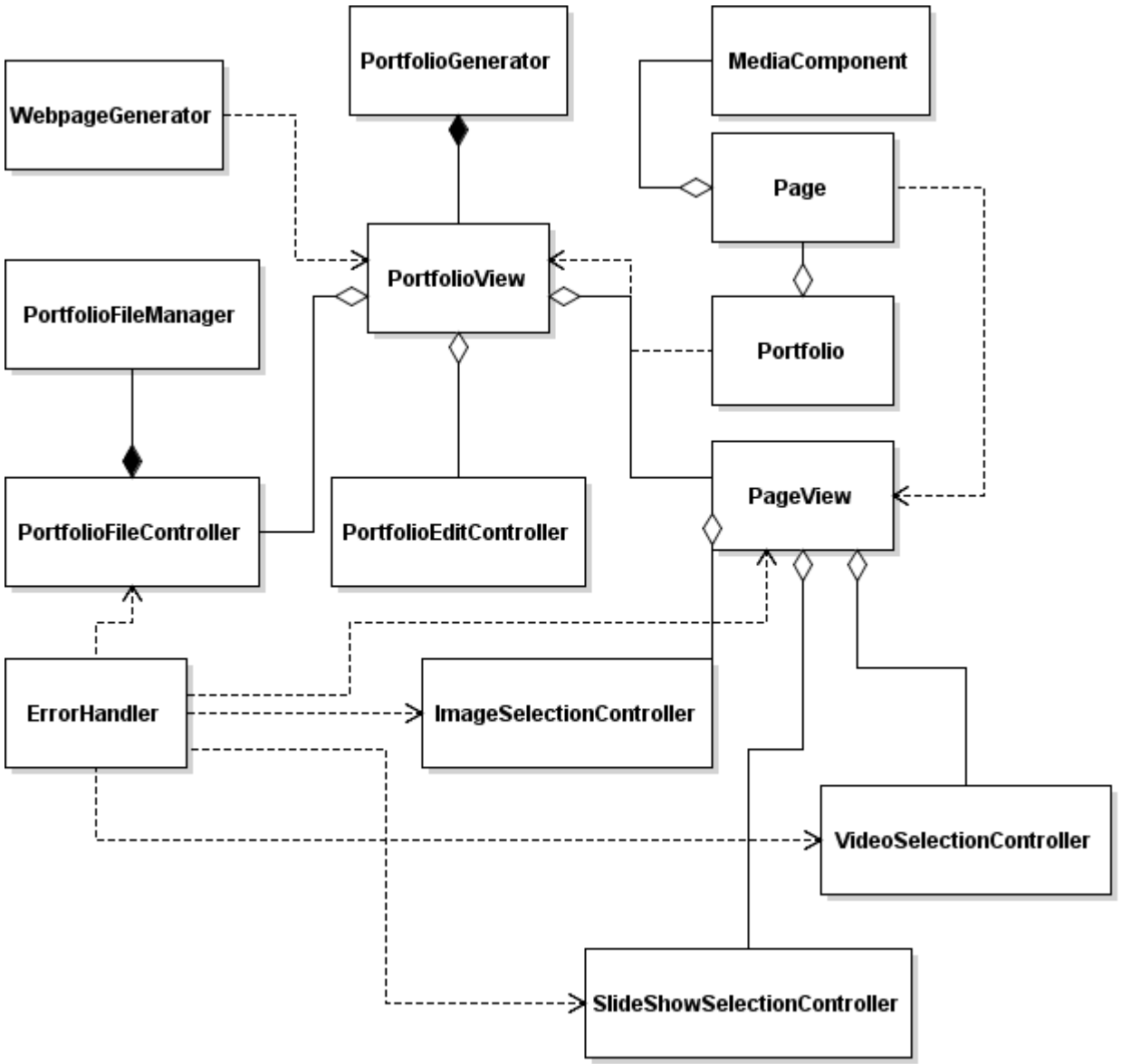

Figure 3 The ePortfolio Generator Overview UML Diagram

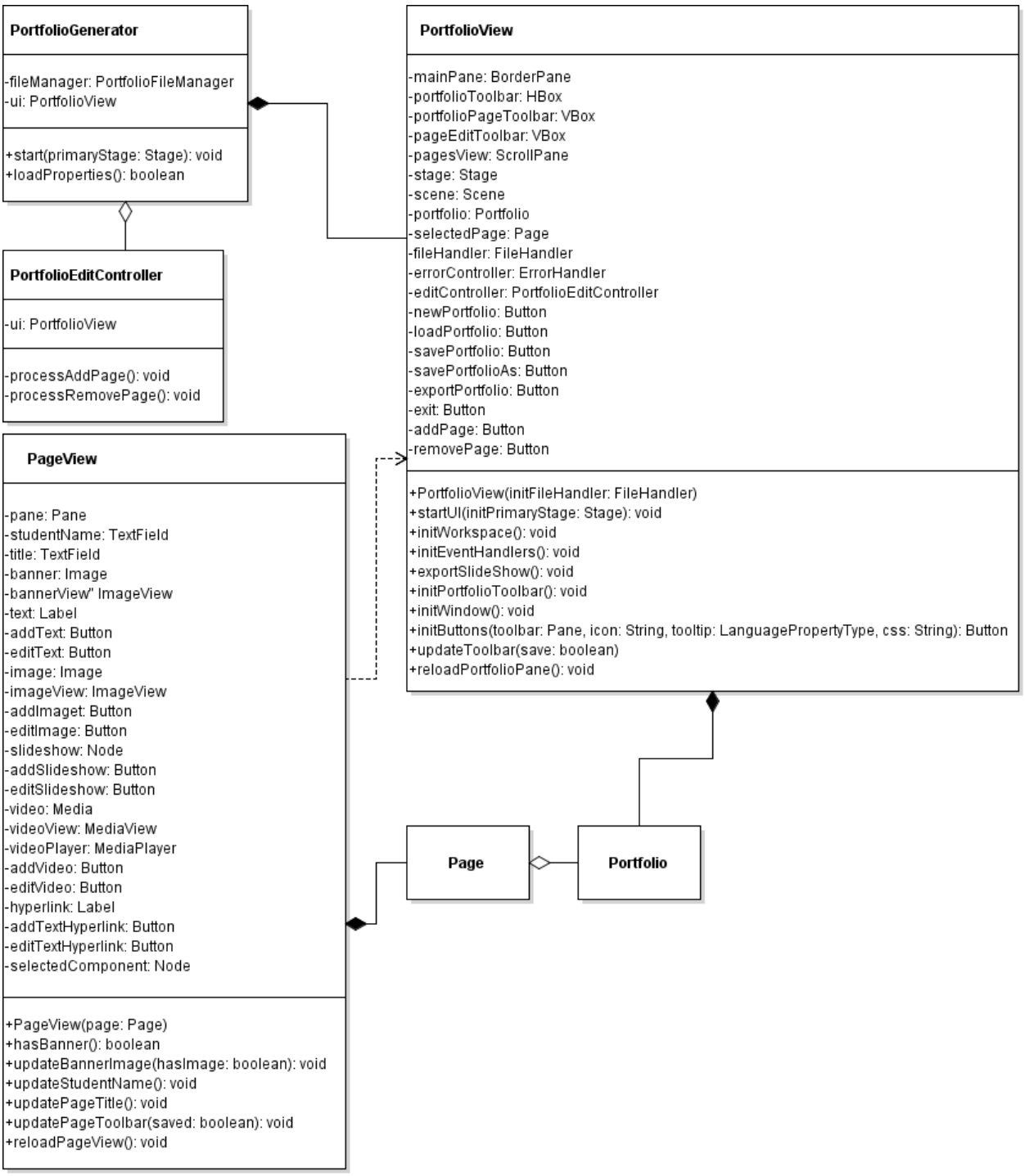

Figure 4 Detailed Portfolio Generator Class diagram, with related components

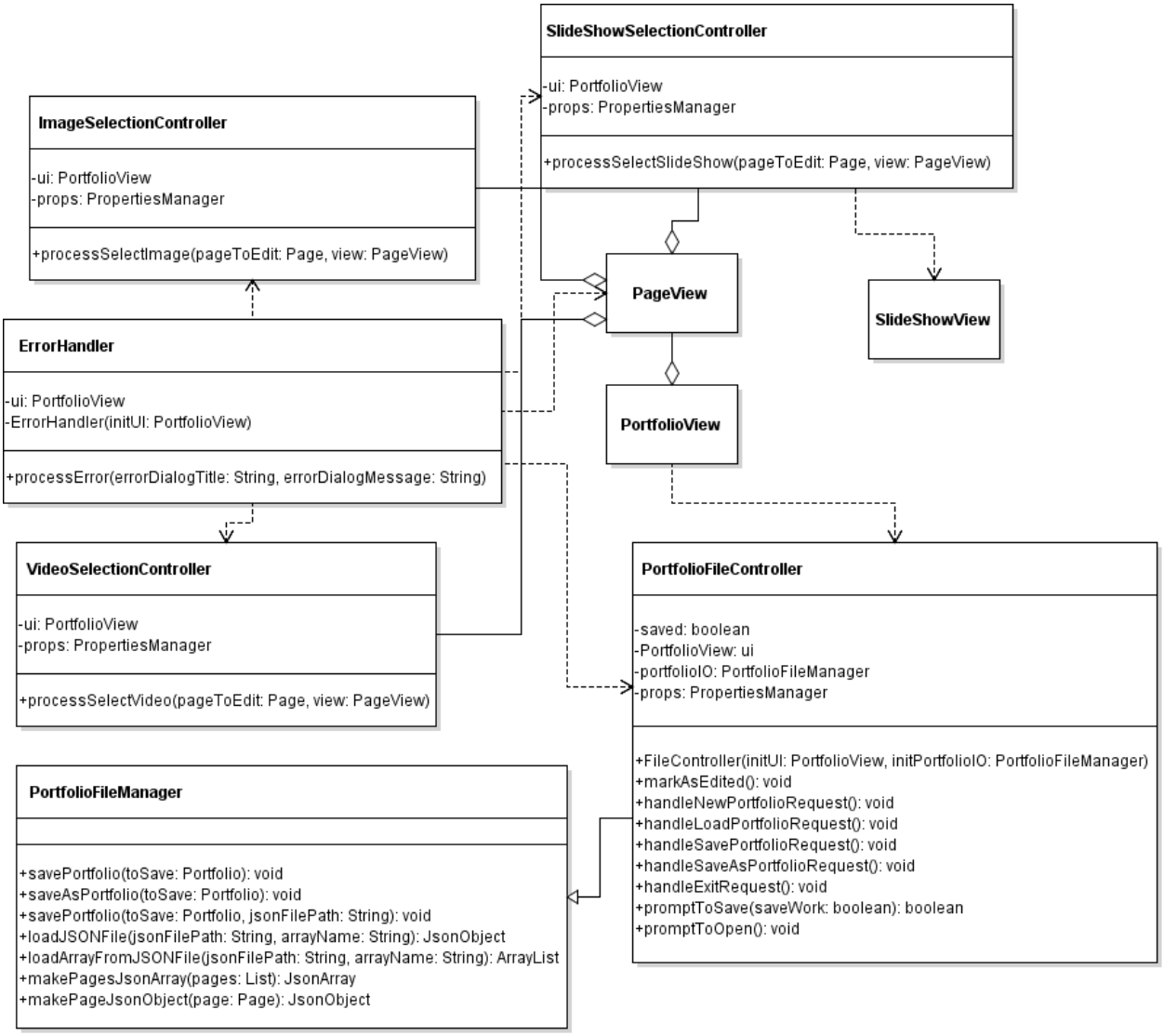

Figure 5 Detailed File Managers' Class diagram, with related components

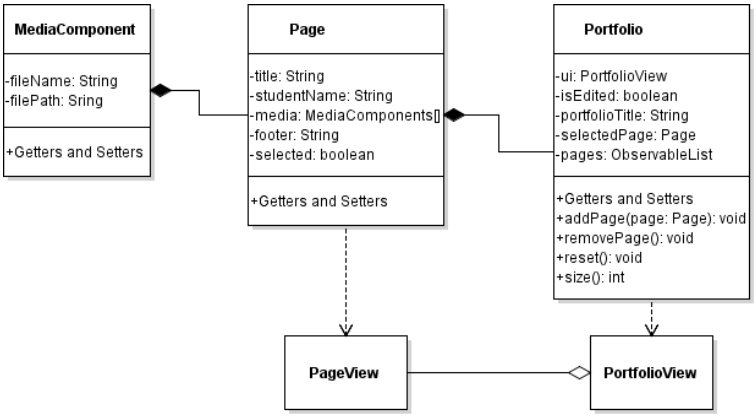

Figure 6 Portfolio Model Class Diagram

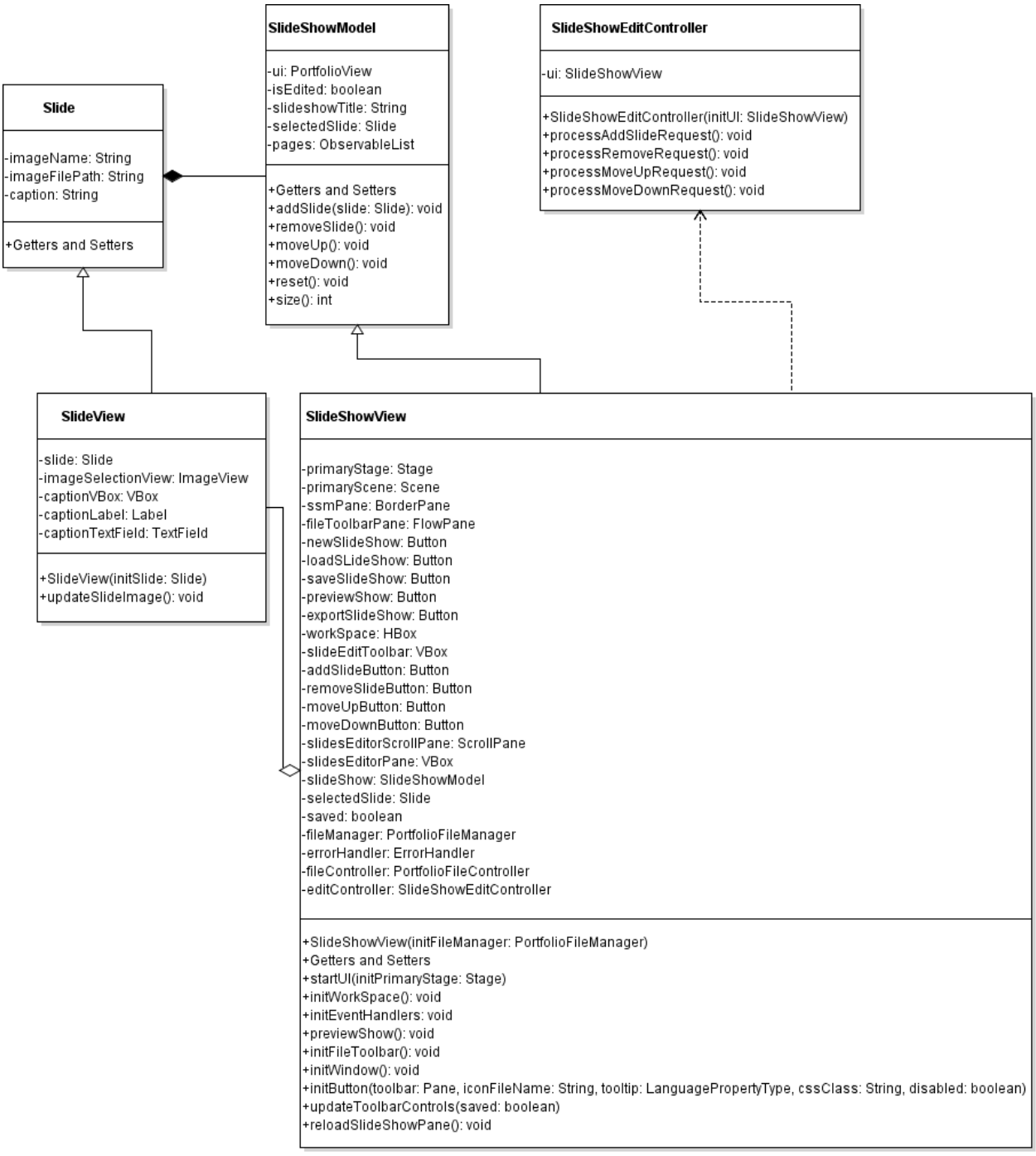

Figure 7 SlideShow Selection Control Class Diagram

4 Method Level Design View Point

Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.

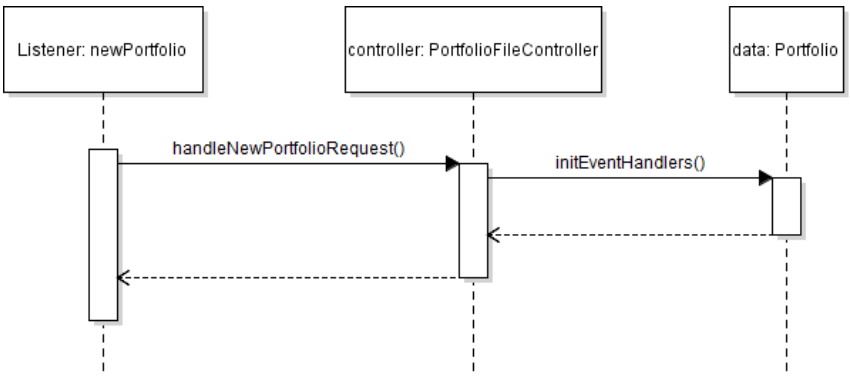

Figure 8 New Portfolio Event Sequence Diagram

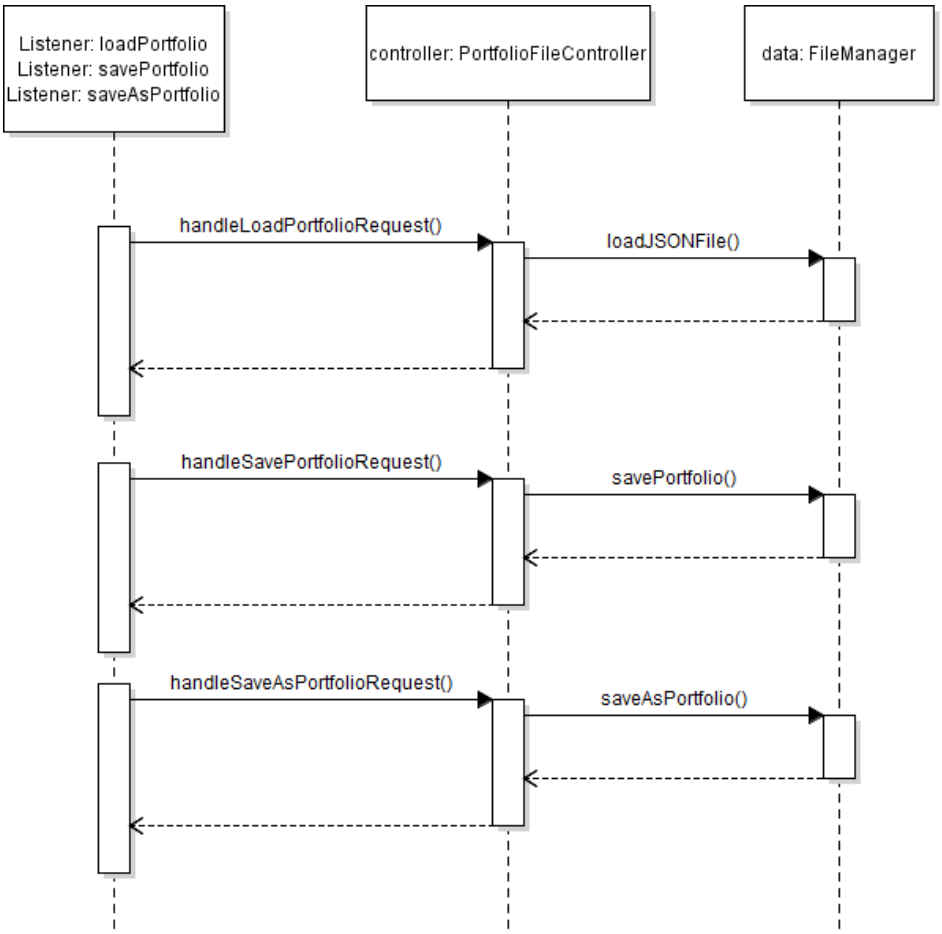

Figure 9 Load, Save and SaveAs Event Sequence Diagram

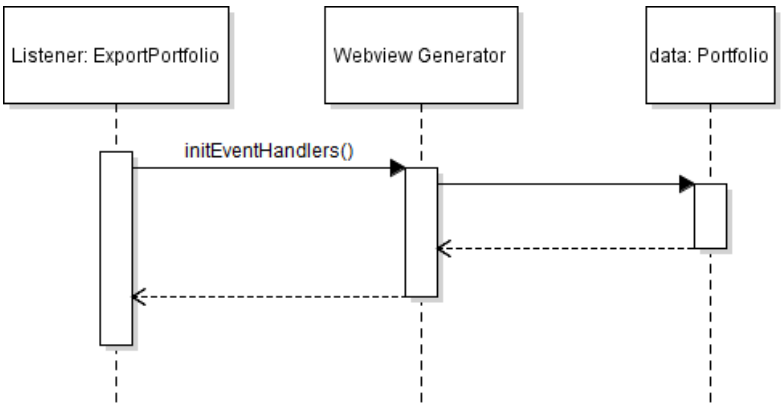

Figure 10 Export Portfolio Controller Sequence Diagram

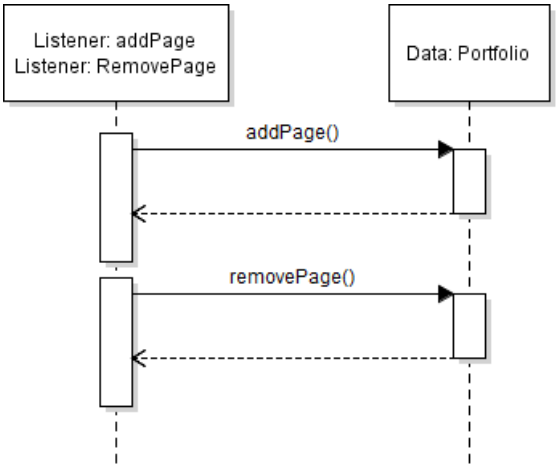

Figure 11 Number of Pages Controller Sequence Diagram

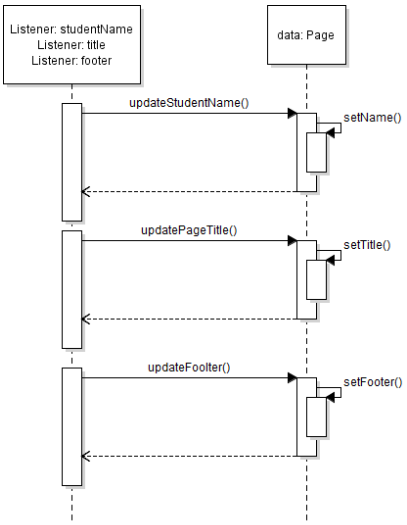

Figure 12 StudentName, Title, and Footer Sequence Diagram

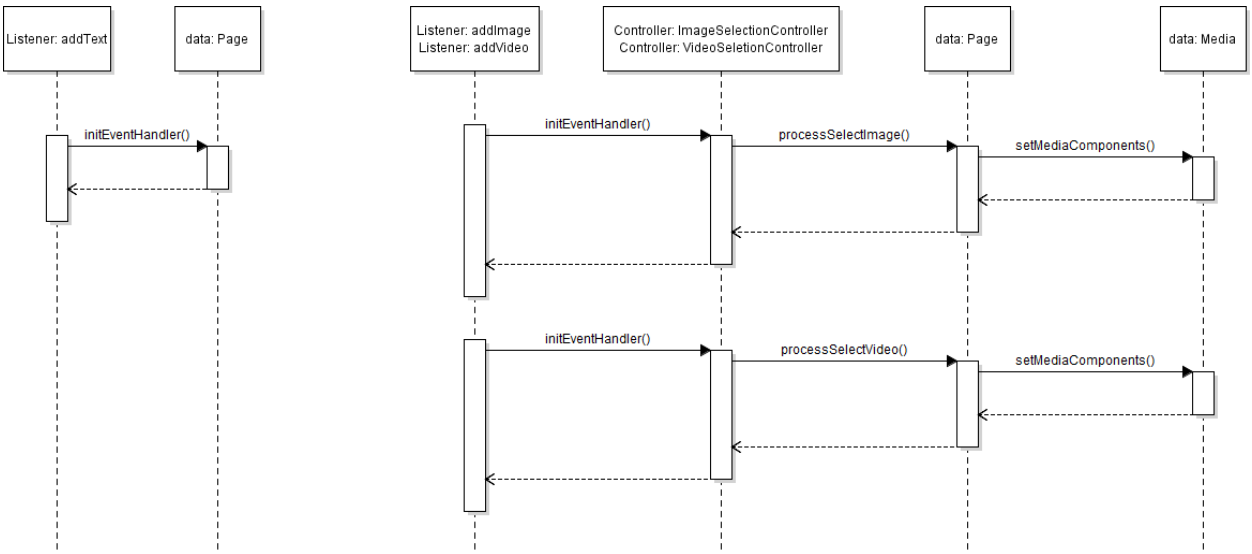

Figure 13 Adding Text, Image and Video Sequence Diagrams

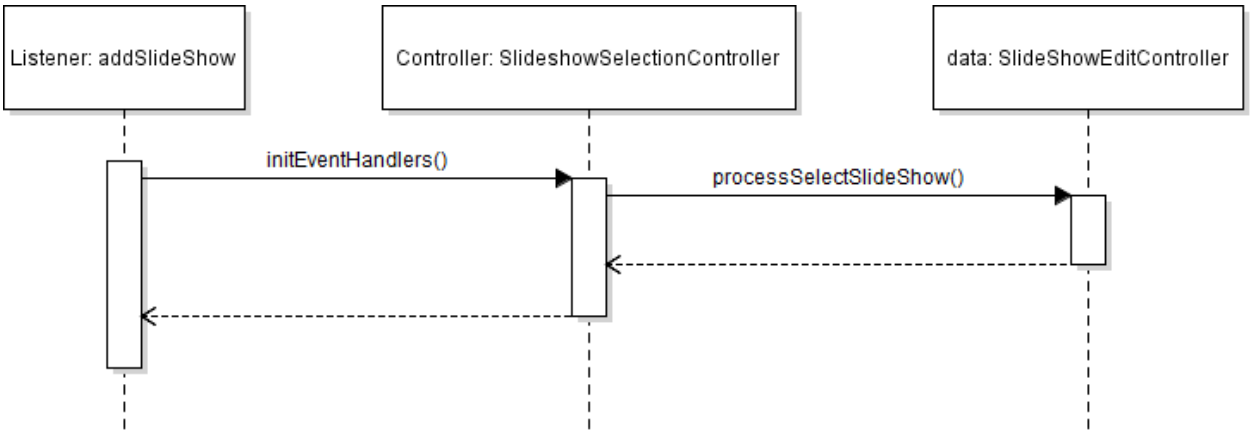

Figure 14 Slide Show Selection Sequence Diagram

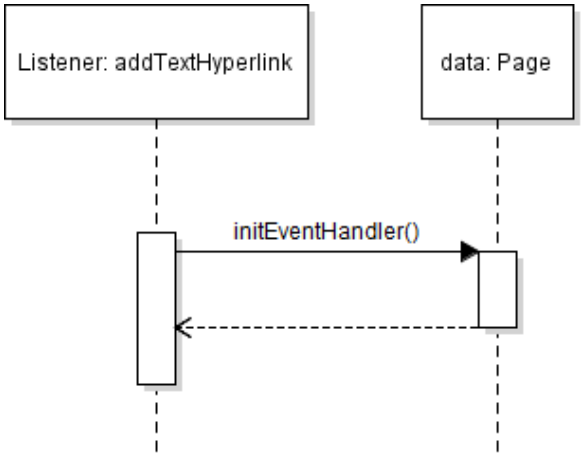

Figure 15 Text Hyperlink Sequence Diagram

5 File Structure and Formats

All the saved portfolios (and slideshows if present) shall be stored in JSON File Format. All data that has to be loaded will be loaded from the JSON File previously stored in. Language Property for the Button Tooltip will be loaded from XML Files. Additionally, for the generation of websites, a CSS and a JavaScript files shall be used. For the design of the main application, another CSS file shall be used, which will be similar to the one used for generating web pages

5.1 JSON File Format

```
{
	"title": "Rahul's ePortfolio",
	"pages": [
		{
			"title": "CSE 219",
			"studentName": "Rahul Agasthya",
			"media": [
				{
					"filename": "random.mp4",
					"filePath": "…media\\videos\\"
				}
			]
			"footer": "This is an example page.",
			"selected": "false"
		}
	]
}
```

5.2 XML File

```
<properties>
<property_list>
   <property name="TITLE_WINDOW"    value="ePortfolio Generator"></property>
   <property name="TOOLTIP_NEW"     value="New ePortfolio"></property>
   <property name="TOOLTIP_LOAD"    value="Load ePortfolio"></property>
   <property name="TOOLTIP_EXIT"    value="Exit"></property>
</property_list>
<properties>
```

5.3 For Web View

6.2.1 CSS File

```
heading {
        font-family: Times New Roman;
        font-weight: bold;
        font-size: 100%;
        background-color: #80080;
        color: #00800;
}
```

6.2.2 JavaScript File

```
var SlideShowSpeed = 3000;
var CrossFadeDuration = 3;
var Picture = new Array();
var Caption = new Array();
Picture[1] = 'img/BryceCanyonUtah.jpg';
Caption[1] = "Utah Canyons";
Picture[2] = 'img/OregonTrailNebraska.jpg';
Caption[2] = "Nebraska";
Picture[3] = 'img/YosemiteCalifornia.jpg';
Caption[3] = "Yosemite California";
var tss;
var iss;
var jss = 1;
var pss = Picture.length-1;
var preLoad = new Array();
for (iss = 1; iss < pss+1; iss++) {
    preLoad[iss] = new Image();
    preLoad[iss].src = Picture[iss];
}
var play = true;
```

```
        function playpause() {
                if (play) {
                        runSlideShow();
                        play = false;
                }
                else {
                        pause();
                        play = true;
    }
}
function runSlideShow() {
    document.getElementById("play").src = "img/Pause.jpg";
    if (document.all){
        document.images.PictureBox.style.filter = "blendTrans(duration=2)";
        document.images.PictureBox.style.filter =
"blendTrans(duration=CrossFadeDuration)";
        document.images.PictureBox.filters.blendTrans.Apply();
        }
        document.images.PictureBox.src = preLoad[jss].src;
    if (document.getElementById)
        document.getElementById("CaptionBox").innerHTML = Caption[jss];
    if (document.all)
        document.images.PictureBox.filters.blendTrans.Play();
    jss = jss + 1;
    if (jss > (pss))
        jss = 1;
        tss = setTimeout('runSlideShow()', SlideShowSpeed);
    }
function pause() {
    document.getElementById("play").src = "img/Play.jpg";
    clearTimeout(tss);
    }
```

6.2.3 HTML File

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Rahul</title>
  <link rel="stylesheet" type="text/css" href="css/slideshow_style.css">
  <script src="js/slideshow.js" type="text/javascript"></script>
</head>
<body>
  <heading>Rahul</heading>
</body>
</html>
```

6 Support Information

Note that this document should serve as a reference for those implementing the code, so we'll provide a table of contents to help quickly find important sections.

6.1 Table of Contents

6.2 Appendixes – JAR Files

6.2.1 PropertiesManager JAR Files

This class is used for loading properties from XML files that can then be used throughout an application. Note that this class is a singleton, and so can be accessed from anywhere. To get the singleton properties manager, just use the static accessor:

```
PropertiesManager props = PropertiesManager.getPropertiesManager();
```

Now you can access all the properties it currently stores using the `getProperty()` method. Note that the `properties_schema.xsd` file specifies how these files are to be constructed.

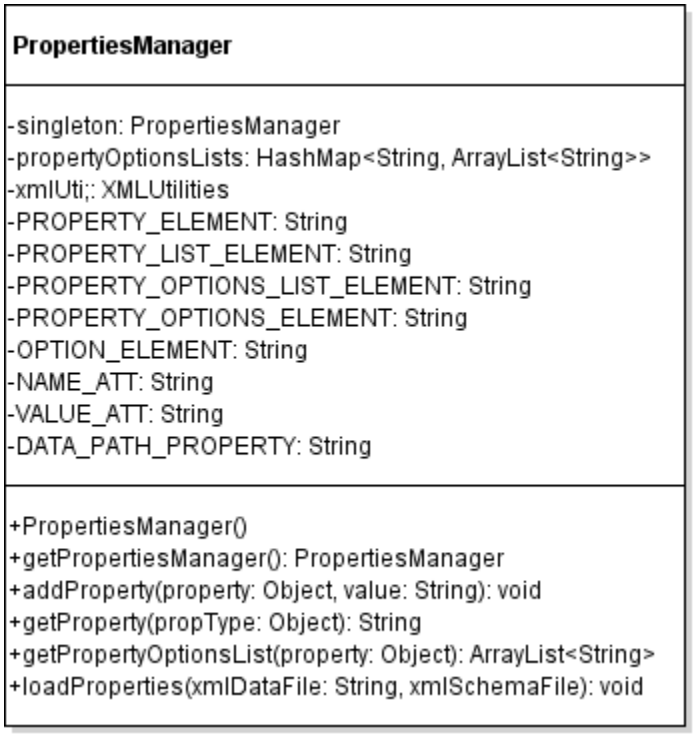

Figure 16 PropertiesManager Class Diagram

6.2.2 XMLUtilities

This class provides general purpose utilities for the loading and extracting of data from XML Files. This class is useful for the validation and loading of XML docs and for easy extraction of data from. Note that it to be a really useful library it would need to be extended a bit, with some more similar functionality added.

The Exception handled by this class is `InvalidXMLFileFormatException.`

The `InvalidXMLFileFormatException` is a checked exception that represents the occasion where an XML document does not validate against its schema (XSD).

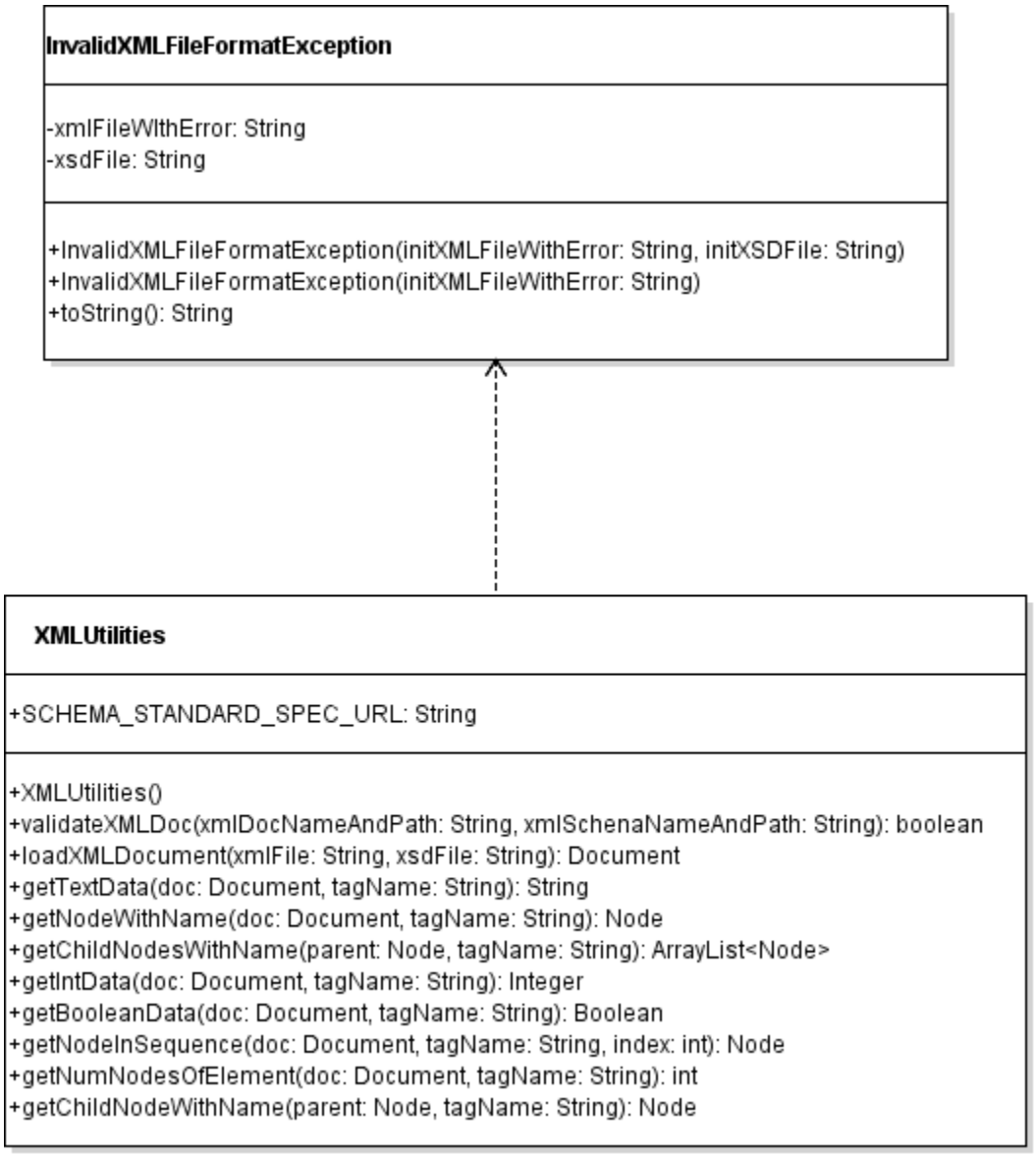

Figure 17 Class Diagram for XMLUtilities