**Course Site GeneratorTM**

**Software Design Description**

**Author:** Phuc Nguyen

SBU CSE219 TM

March, 2017

Version 1.0

**Abstract:** This document describes the software design for Course Site Generator, a utility for professors to handle their classes, projects, teaching assistants, and students.

**Based on IEEE Std 1016TM-2009 document format**

Copyright © 2017 SBU CSE219, which is a made up company and doesn’t really own Course Site Generator,

Please note that this is a fictitious document for hw3 via CSE219 students at Stony Brook.

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

**1 Introduction**

University courses are required to provide course materials at the start of a term that help students understand course requirements and plan their semesters. These come in the form of a course syllabus and schedule that list information like course policies and dates for exams and assignments. Many times instructors find the most convenient means to disseminate this information is by way of a course Web site. Course Web sites are typically published at the start of a semester and are updated as it progresses. Such a site keeps the students up to date on deadlines and provides a place for instructors to distribute things like lecture slides.

Places like Stony Brook University’s Computer Science Department have long required a Course Web Site for each taught course, and so every semester, the instructors teaching these courses generate and update this content. This process can be time consuming and tedious. In addition, many times instructors have more important things to do than build beautiful templates, and the result is each course Web site looks different, making it difficult for students to find the content they are after as they navigate differently arranged structures.

But why use course sites at all? Why not just use a tool like Blackboard for organizing course content? Well, sites that that have their own difficulties. They require time consuming login and navigation processes, they do many things, and so many things interfere with the quick retrieval for what a student is looking for, and they are general purpose sites, and so are cluttered with many things a course isn’t even using. For an instructor, building and maintaining a custom course Web page still provides the best service to its students.

The ***Course Site Generator*** application intends to automate the process of building and updating a course Web site in one easy to use tool. The sites produced by this application will look good and will be customizable in a number of different ways, but will exist within a common site and page structure.

**1.1 Purpose**

The purpose of this document is to specify how our ***Course Site Generator*** program should look and operate. The intended audience for this document is all the members of the development team, from the instructors to the software engineers and designers. This document serves as an agreement among all parties and as a reference for how the site creation tool should ultimately be constructed. Upon completing the reading of this document, one should clearly visualize how the application will look and operate as well as understand the way a generated site is setup.

**1.2 Scope**

For this project the goal is for instructors to easily make and update course Web sites. There will be a common structure to the pages and so there are limitations on customization, but the site should be usable for instructors teaching courses in any department at any University.

# 1.3 Definitions, acronyms, and abbreviations

**Document Object Model (DOM)** – a tree data structure maintained by the browser that contains all content for the currently loaded Web page.

**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.

**GUI** – Graphical User Interface, visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

**HyperText Markup Language** – a markup language used to describe Web pages. Web pages are text files encoded in HTML that can employ JavaScript and Stylesheets to build and style content.

**IEEE –** Institute of Electrical and Electronics Engineers, the “world’s largest professional association for the advancement of technology”.

**JavaScript** – the default scripting language of the Web, JavaScript is provided to pages in the form of text files with code that can be loaded and executed when a page loads so as to dynamically generate page content in the DOM.

**Stylesheet** – a static text file employed by HTML pages that can control the colors, fonts, layout and other style components in a Web page.

**UML** – Unified Modeling Language, a standard set of document formats for designing software graphically.

**Use Case Diagram** – A UML document format that specifies how a user will interact with a system.

# References

**IEEE Std 830TM-1998 (R2009) –** IEEE Recommended Practice for Software Requirements Specification

**1.5 Overview**

This document does not specify how to build the appropriate technologies, it is simply an agreement concerning what to build. Section 2 of this document will provide the context for the project and specify all the conceptual design. Section 3 will present how the user interface should be laid out. Section 4 provides a Table of Contents, an Index, and References. It contains all the UML models needed to create the program and is a conceptual design as stated above in which we will use to design a model without writing any programming code.

**2 Package-Level Design Viewpoint**

As mentioned, this design will encompass the DFramework, PFramework, and Generator 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.

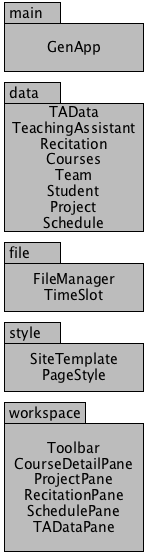
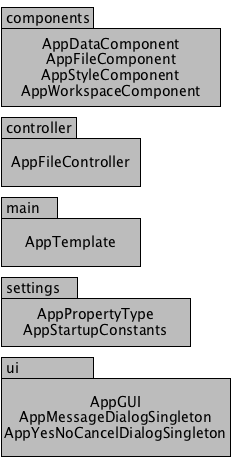
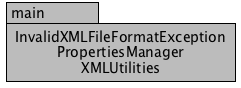
**2.1 DFramework, PFramework, and Generator overview**

The Generator and D/P framework will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in home packages.

Generator Framework

DFramework

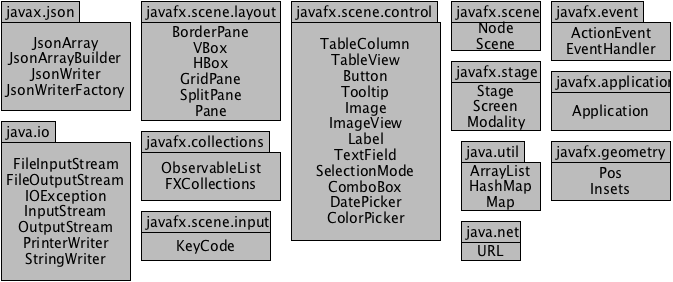
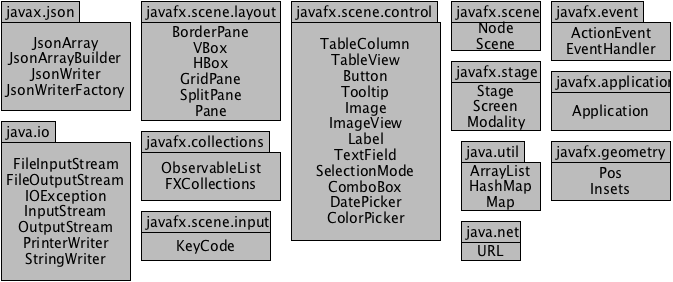
PFramework

****

**Figure 2.1: Design Packages Overview**

**2.2 Java API Usage**

All the frameworks will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.

****

**Figure 2.2: Java API Classes and Packages To Be Used**

**2.3 Java API Usage Descriptions**

Tables 2.1-2.7 below summarize how each of these classes will be used.

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **JsonArray** | For creating array of json items to save to json. |
| **JsonArrayBuilder** | For building arrays to save to json. |
| **JsonWriter** | For writing json files. |
| **JsonWriterFactory** | For building the written json file. |

**Table 2.1: Uses for classes in the Java API’s javax.json package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **FileInputStream** | For loading files by opening a stream. |
| **FileOutputStream** | For saving files by opening a stream. |
| **IOException** | For checking if there’s an error with input/output. |
| **InputStream** | For FIS (file input stream) to use to read data. |
| **OutputStream** | For FOS (file output stream) to use to output data. |
| **PrinterWriter** | For printing out chars/variables to be saved. |
| **StringWriter** | For writing string variables. |

**Table 2.2: Uses for classes in the Java API’s java.io package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ObservableList** | For storing sets of data that can be modified. |
| **FXCollections** | For storing sets of data but using its methods to create deep copy. |

**Table 2.3: Uses for classes in the Java API’s java.fx.collections package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Node** | For using via casting to check nodes. |
| **Scene** | For creating the scene for our Stage. |

**Table 2.4: Uses for classes in the Java API’s java.scene package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **TableColumn** | For creating columns for our table view. |
| **TableView** | For creating a table for our data sets. |
| **Button** | For executing events such as add/edit/remove. |
| **Tooltip** | For hover over assistance. |
| **ImageView** | For rendering images such as icons or button assets. |
| **Image** | For loading images via resource directory. |
| **Label** | For creating labels for our other nodes and descriptions. |
| **TextField** | For receiving input from the user to enter data. |
| **SelectionMode** | For getting TableView data to use in our statements. |
| **ComboBox** | For a selection of choices for the user to pick from. |
| **DatePicker** | For picking a specified date for a data set. |
| **ColorPicker** | For use when picking team color and text color |
| **Vector** | For storing data like the Strings for rendering debugging text. |

**Table 2.5: Uses for classes in the Java API’s java.scene.control package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **URL** | For access to URL formats and to navigate through them. |

**Table 2.6: Uses for classes in the Java API’s java.net package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ArrayList** | For storing sets of data and cycling through them |
| **HashMap** | For storing sets of data with variance in cycling. |
| **Map** | For storing sets of data with less variance of data. |

**Table 2.7: Uses for classes in the Java API’s javax.util package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **BorderPane** | For creating GUI with top and center properties. |
| **VBox** | For creating GUI panes that are added vertically. |
| **HBox** | For creating GUI panes that are added horizontally. |
| **GridPane** | For creating GUI panes that can be added by coordinates. |
| **SplitPane** | For creating a pane that can have respective left/right side nodes. |
| **Pane** | For checking through panes via casting. |

**Table 2.5: Uses for classes in the Java API’s java.scene.layout package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Stage** | For creating out stage in our application. |
| **Screen** | For functions pertaining to the screen of our application. |
| **Modality** | For functions such as having a parent window in front. |

**Table 2.6: Uses for classes in the Java API’s javafx.stage package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Application** | For creating our GUI application and executing it. |

**Table 2.6: Uses for classes in the Java API’s javafx.application package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **KeyCode** | For getting the keycode of a button press and creating an action. |

**Table 2.6: Uses for classes in the Java API’s javafx.scene.input package**

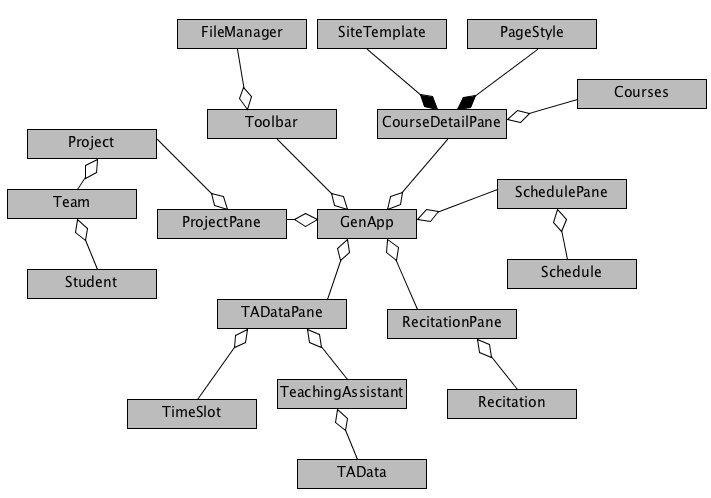
|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ActionEvent** | For creating actions via lambda expressions. |
| **EventHandler** | For handling our events used with our buttons. |

**Table 2.7: Uses for classes in the Java API’s javafx.event package**

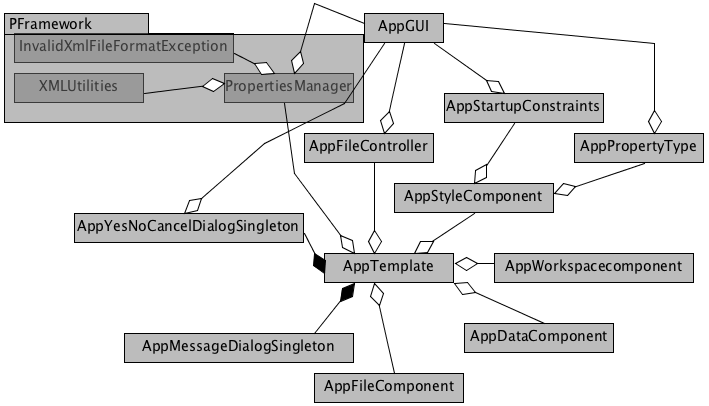
|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Pos** | For centering our nodes in our containers |
| **Insets** | For creating spacing between our nodes with our containers |

**Table 2.7: Uses for classes in the Java API’s javafx.Geometry package**

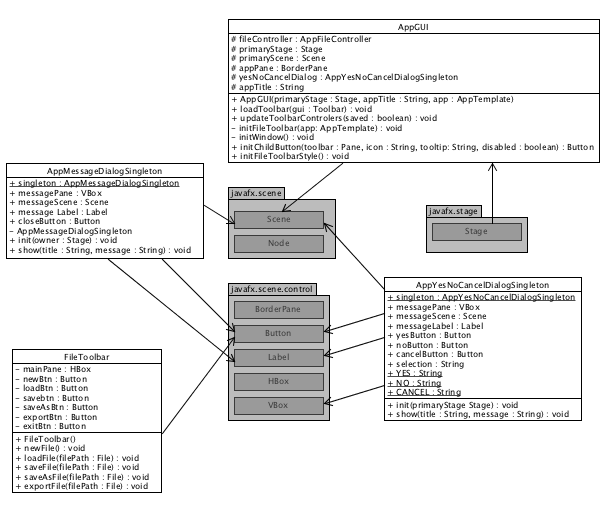
**3 Class-Level Design Viewpoint**

As mentioned, this design will encompass the Generator application and two supporting framework classes. The following UML Class Diagrams reflect this. Note that due to the complexity of the project, we present the class designs using a series of diagrams going from overview diagrams down to detailed ones. 

**Figure 3.1: Generator Framework Overview UML Class Diagram**

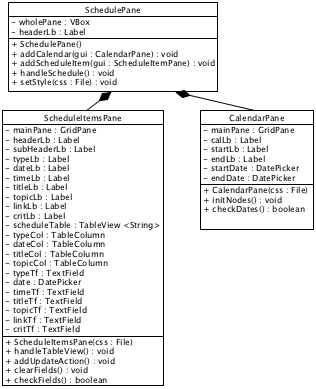


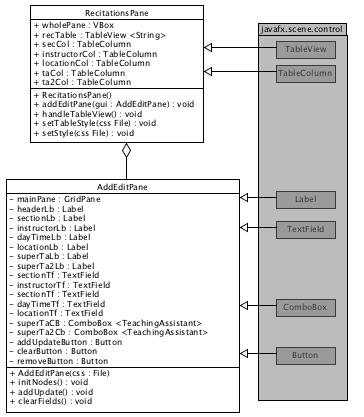
**Figure 3.2: DFramework & PFramework Overview UML Class Diagram**

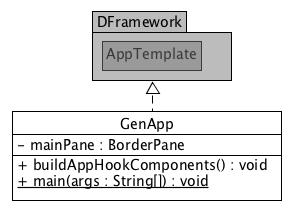
****

**Figure 3.3: Detailed DFramework with packages UML Class Diagram**

9

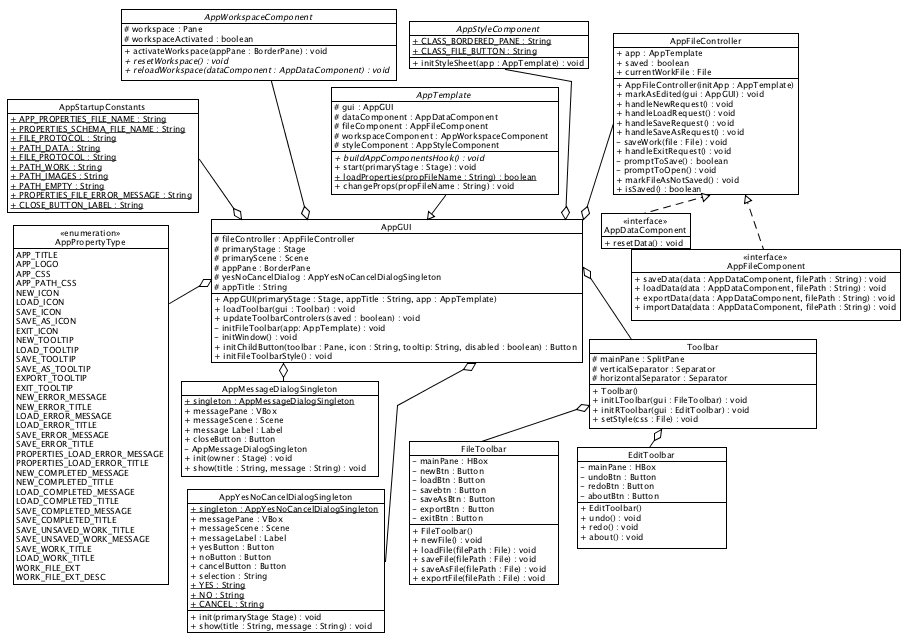






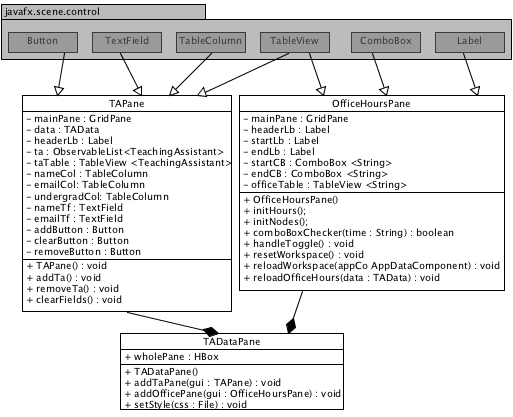
**Figure 3.4: Detailed GeneratorApp UML Class Diagram**

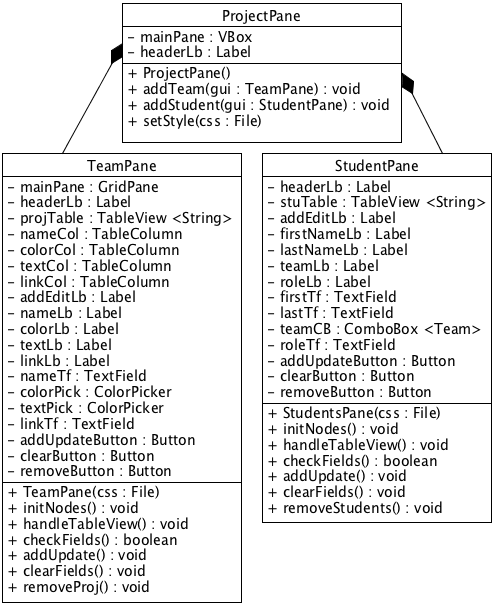
10



**Figure 3.5: DFramework UML Class Diagram**

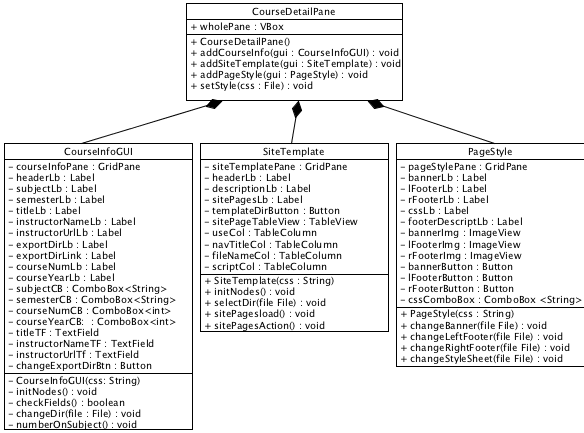
11



****

**Figure 3.6: Detailed ProjectPane and TAPane UML Class Diagram**

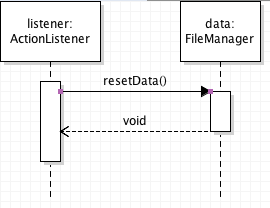
12



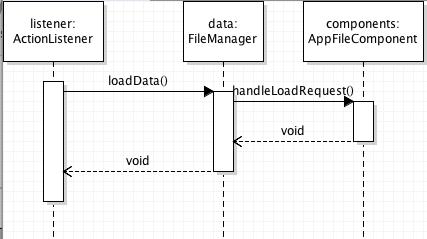
**Figure 3.7: CourseDetailPane UML Class Diagram**

**4 Method-Level Design Viewpoint**

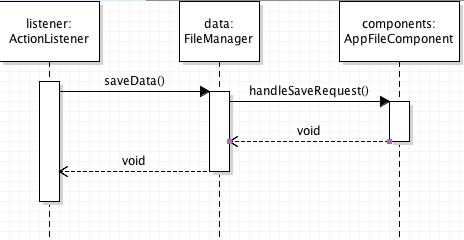
Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.



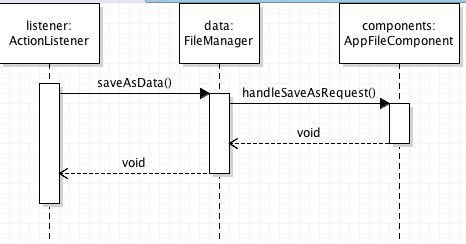
**Figure 4.1: Create New Course Site UML Sequence Diagrams**



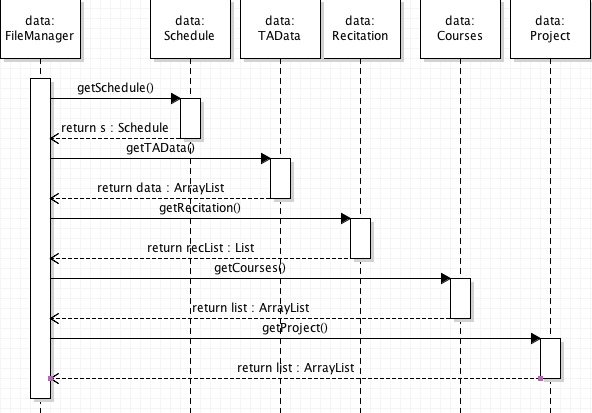
**Figure 4.2: Load Course Site UML Sequence Diagrams**

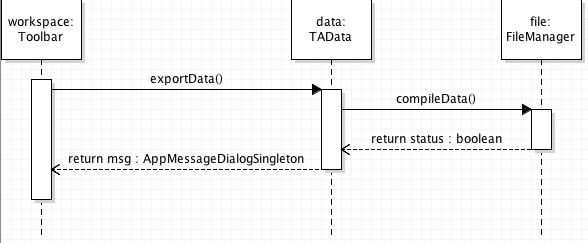


**Figure 4.3: Save Course Site UML Sequence Diagrams**

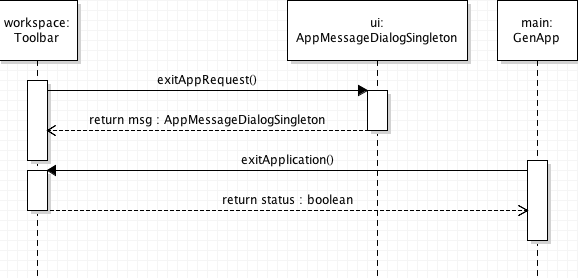


**Figure 4.4: Save As Course Site UML Sequence Diagrams**

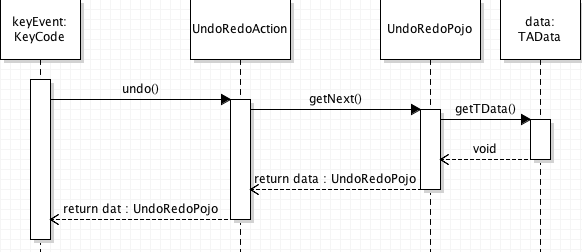
Optional Figure shows how save works:



**Figure 4.5: Export Course Site UML Sequence Diagrams**

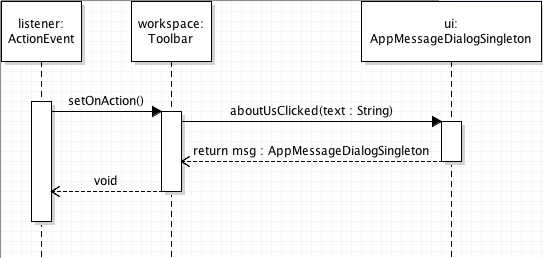
****

**Figure 4.6: Exit Application UML Sequence Diagram**

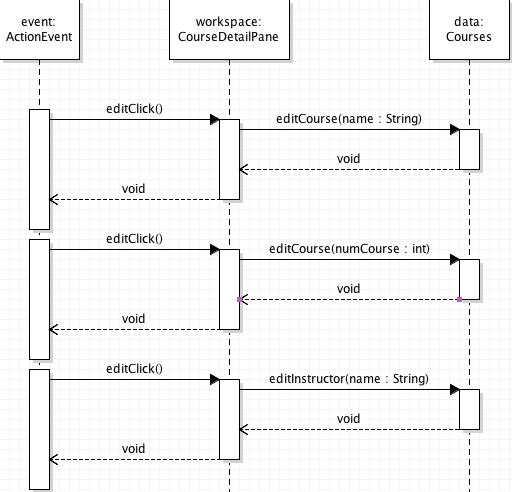
****

**Figure 4.7: Undo UML Sequence Diagram**

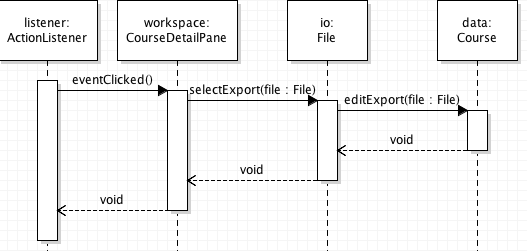
**Figure 4.8: Redo UML Sequence Diagram**

****

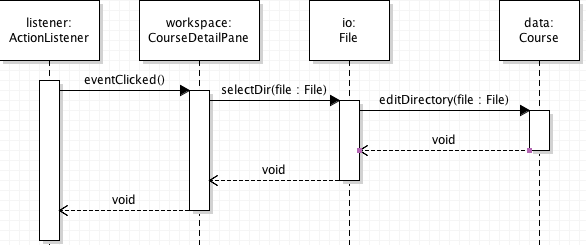
**Figure 4.9: About UML Sequence Diagram**



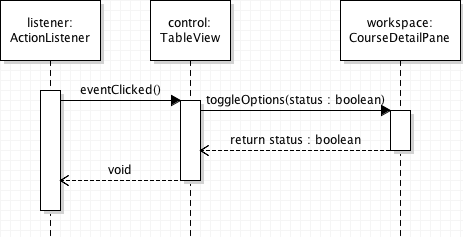
**Figure 5.0: Edit Course Info UML Sequence Diagram**



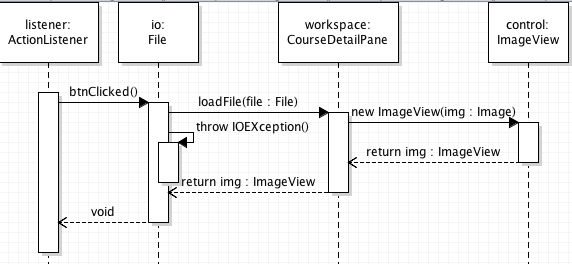
**Figure 5.1: Select Export Directory UML Sequence Diagram**



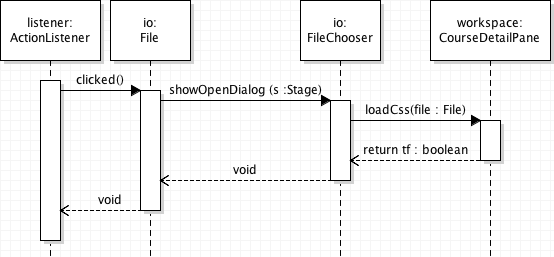
**Figure 5.2: Select Template Directory UML Sequence Diagram**



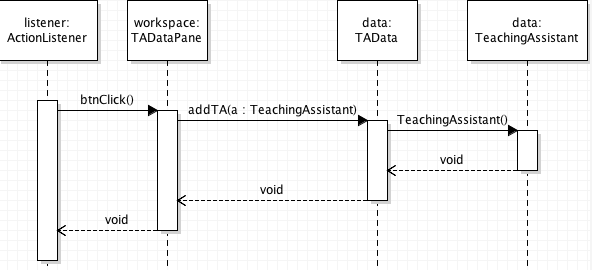
**Figure 5.3: Toggle Use Template Page UML Sequence Diagram**



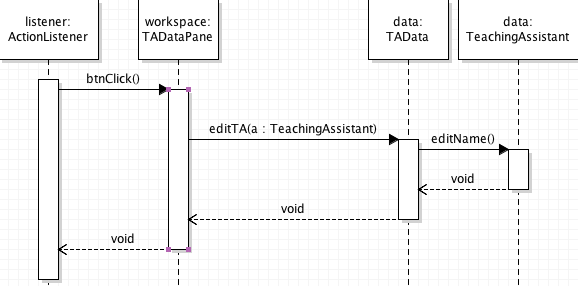
**Figure 5.4: Select Branding Images UML Sequence Diagram**



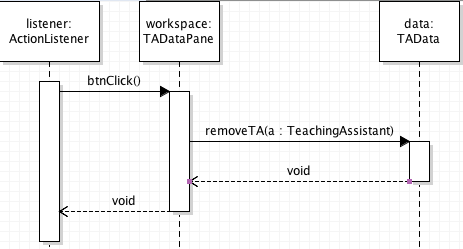
**Figure 5.5: Select Stylesheet UML Sequence Diagram**



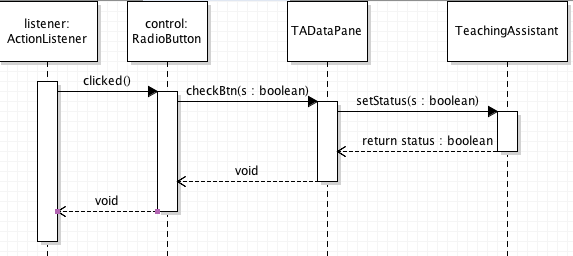
**Figure 5.6: Add Teaching Assistant UML Sequence Diagram**

****

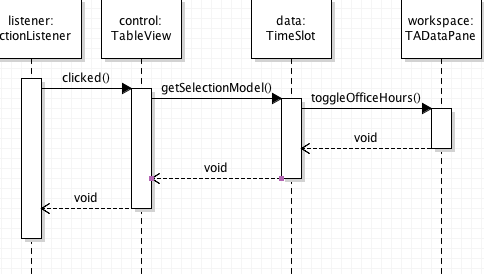
**Figure 5.7: Edit Teaching Assistant UML Sequence Diagram**



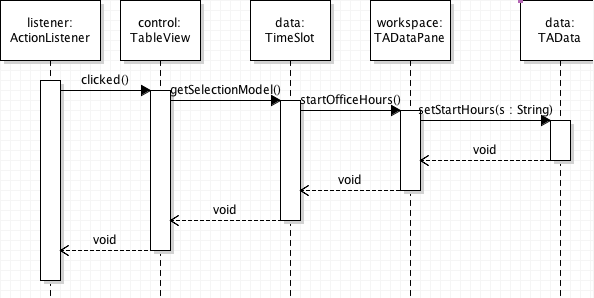
**Figure 5.9: Remove Teaching Assistant UML Sequence Diagram**

****

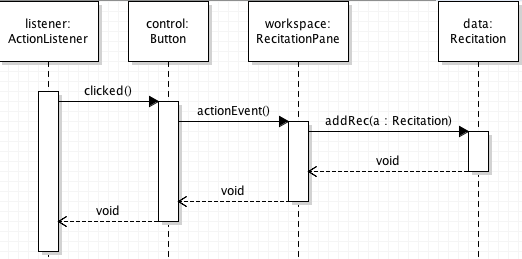
**Figure 6.0: Toggle TA Undergrad UML Sequence Diagram**



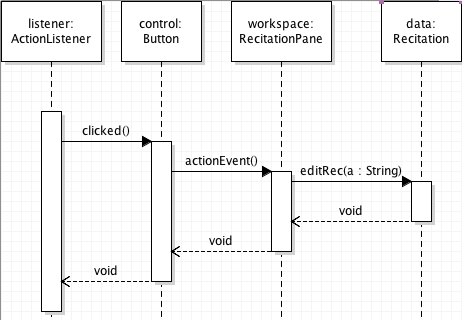
**Figure 6.1: Toggle TA Office Hours UML Sequence Diagram**



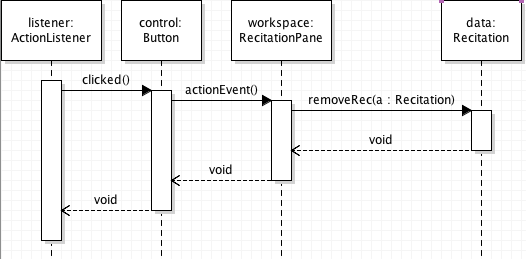
**Figure 6.2: Change Start/End Office Hours UML Sequence Diagram**



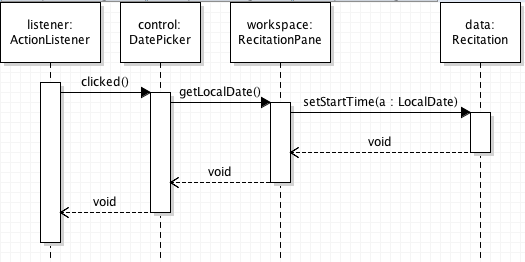
**Figure 6.3: Add Recitation UML Sequence Diagram**

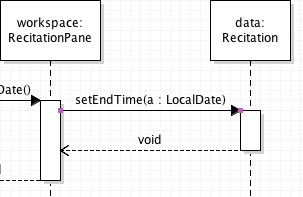
****

**Figure 6.4: Edit Recitation UML Sequence Diagram**

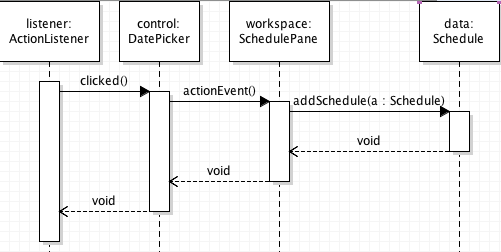


**Figure 6.5: Remove Recitation UML Sequence Diagram**

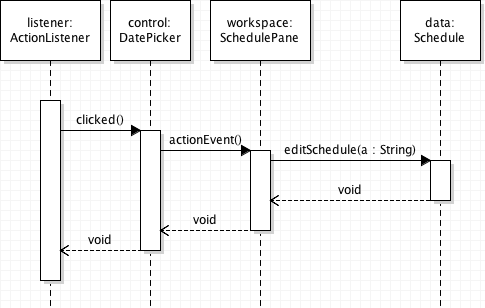




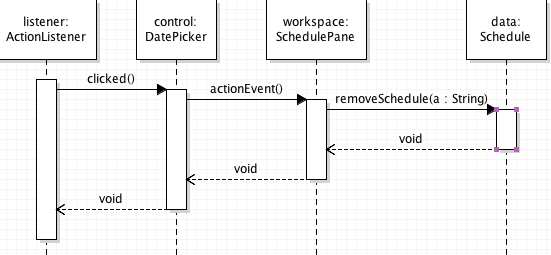
**Figure 6.6: Edit start and end dates UML Sequence Diagram**

****

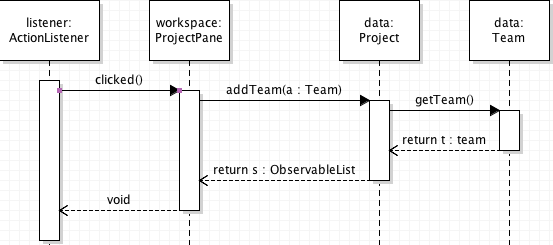
**Figure 6.7: Add Schedule Item UML Sequence Diagram**

****

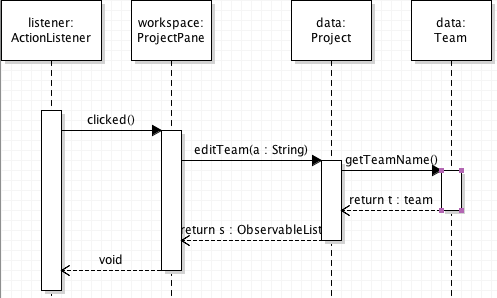
**Figure 6.8: Edit Schedule Item UML Sequence Diagram**



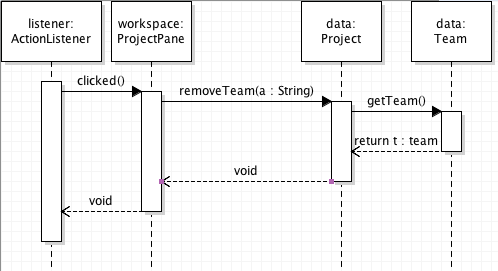
**Figure 6.9: Remove Schedule Item UML Sequence Diagram**



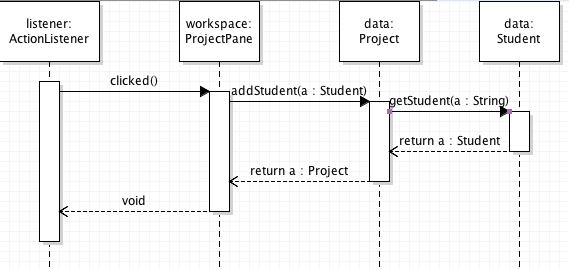
**Figure 7.0: Add Team UML Sequence Diagram**



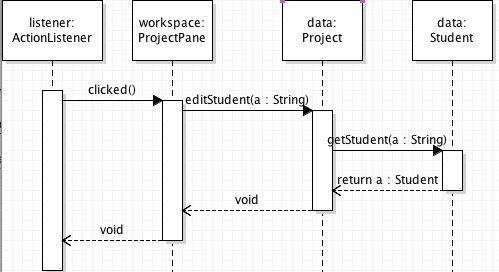
**Figure 7.1: Edit Team UML Sequence Diagram**

****

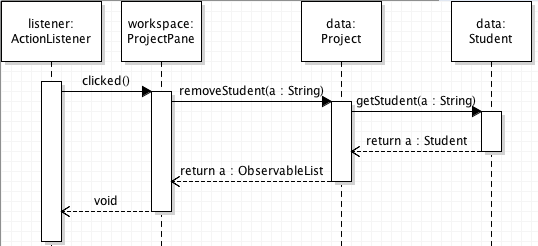
**Figure 7.2: Remove Team UML Sequence Diagram**



**Figure 7.3: Add Student UML Sequence Diagram**



**Figure 7.4: Edit Student UML Sequence Diagram**



**Figure 7.5: Remove Student UML Sequence Diagram**

16

**5. File Structure and Formats**

Note that the DFramework and PFramework (Desktop and Properties) will be provided inside the GeneratorFramework.jar so a java archive file will encapsulate the entire framework. This is needed for the project. This should be imported in the project folder for the Generator application and will be a necessary dependency for the program. The application will be an executable JAR file titled SiteGenerator.jar. All the assets and data needed to accompany the program are included in the resource folder. All images should go to the “resource/images/” folder and all data/asset folders should stay within the program’s relative source (therefore, there will be errors if you move the asset folders out of range of the program as it only sees the scope of its local folder).

**Figure 5.1: Zombiquarium File Structure**

The SiteGenerator.json provides the file and data values for our sets such as Recitation, Courses, TAData, and OfficeHours. Any saved data will be stored in this SiteGenerator.json however we also have subdata files that save other sections of data such as when we export the site, it will generate the htmls templates which we save.

Each section can be found in this order:

**SiteGenerator**

**OfficeHours\_filePath,**

**TAData\_filePath,**

**Recitation\_filePath,**

**Course\_filePath,**

We can define these values

**OfficeHours** – the filepath to our officehours saved data. This is the start/end and init data values.

**TAData** – this file has all of our asset loading file paths as well as the ArrayList of our grouped sets.

**Recitation** – this file has the path towards where the recitation data sets are and all related fields.

**Courses** – this file has a path to the different courses we can use to load in respectively.

**6. Supporting 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**

1. Introduction 2

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | | Purpose | 2 |
| 2. | | Scope | 2 |
| 3. | | Definitions, acronyms, and abbreviations | 2 |
| 4. | | References | 3 |
| 5. | | Overview | 3 |
| 2. | Package-Level Design Viewpoint | | 4 |
|  | 1. Generator and Framework overview | | 4 |
|  | 2. Java API Usage | | 5 |
|  | 3. Java API Usage Descriptions | | 5 |
| 3. | Class-Level Design Viewpoint | | 8 |
| 4. | Method-Level Design Viewpoint | | 14 |
| 5. | File Structure and Formats | | 17 |
| 6. | Supporting Information | | 18 |
|  | 1. Table of contents | | 18 |
|  | 2. Appendixes | | 18 |