Course Site Generator

Software Design Description

**Author:** Jonathan Mathai

Debugging EnterprisesTM

Based on IEEE Std 830TM-1998 (R2009) document format Copyright © 2017 Debugging Enterprises

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. **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.

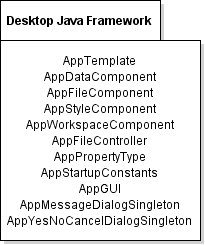
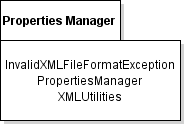
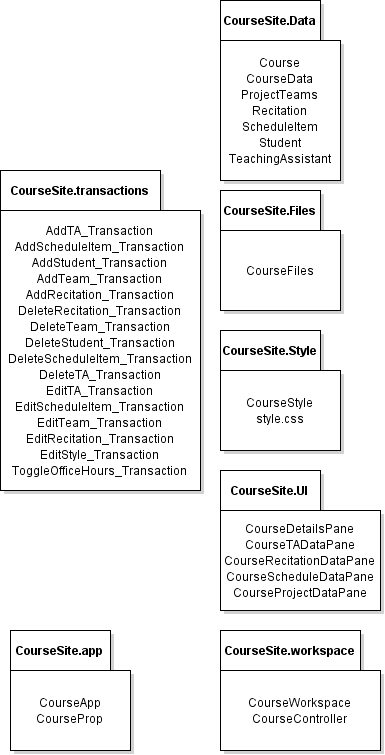
**UML** – Unified Modeling Language, a standard set of document formats for designing software graphically.

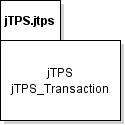
**Use Case Diagram** – A UML document format that specifies how a user will interact with a system.

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

**2.1 Desktop Java Framework and Property Manager**

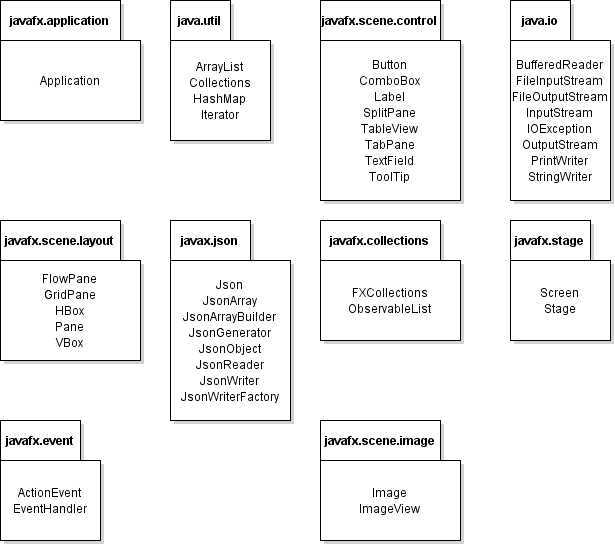
The Desktop Java Framework and the Property Manager Framework will be used in this project to create the template for the application that will be built for the Course Site Generator. The Course Site Generator will follow a design scheme which will break up application into data, files, style, user-interface and workspace components.





**2.2 Java API Usage**

The following classes described below are classes that will be used from the Java API to help provide support for this application.



|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| Application | Serves as the entry point class for the Course Site Generator Application |

Figure 2.2.1: API usage of classes in javafx.application package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| ArrayList | For storing temporary data before it is rendered in the application. |
| Collections | For storage in addition to the static methods for sorting data. Will be used to store TA names for a course alphabetically. |
| HashMap | A data structure that will be used for storing data. Will be used for storing office hours data and the data for site pages. |
| Iterator | For iterating through all the items that are stored within a given HashMap data structure. In order to perform operations more quickly. |

Figure 2.2.2: API usage of classes in java.util package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| Button | For checking if a user selected a certain option in a transaction. Used to initiate transactions in the user interface. |
| ComboBox | For providing the user with a provided list of inputs to choose course details, supervising TA and other data. |
| Label | For displaying text in the panes that are displayed in the user interface of the application. |
| SplitPane | For dividing components of a tab into two sections. This is used in the TA Data section for the TA data and office hours. |
| TableView | For displaying the TA information in the TA Data tab. Used to display TA names and emails. |
| TabPane | For storing the multiple panes that will be used in the user-interface for the application. |
| TextField | For providing a means of asking for user input to change data in the application. Used in many adding and editing operations. |
| ToolTip | For providing a common toolbar which will allow the user to perform actions throughout the application. |

Figure 2.2.3: API usage of javafx.scene.control package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| BufferedReader | For reading from lines from a file when importing in json files for the stored data in the application. |
| FileInputStream | For importing data from a json file to populate data fields in application. |
| FileOutputStream | Used in the process of writing to a file. This will be used when saving to a json file when changes have been made in the application. |
| InputStream | Also used in the process of importing from a file. |
| IOException | Used to catch any input and output errors that may take place during the application. |
| OutputStream | Used in aid of to writing to a json file when saving changes in the application. |
| PrintWriter | For writing to a json file when saving changes in the application. |
| StringWriter | For constructing strings when developing labels or other output texts in the application. |

Figure 2.2.4: API usage of java.io package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| FlowPane | Used to display interface components in a horizontal or vertical manner. |
| GridPane | For displaying user-interface components in a two-dimensional grid. |
| HBox | For placing components in a horizontal box. |
| Pane | The general pane used to store user-interface components in the application. This will also be used as the base pane for each tab in the TabPane. |
| VBox | For placing components in a vertical box. |

Figure 2.2.5: API usage of javafx.scene.layout package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| Json | The factory class that will be used to create json objects to interact with data that is read from the json files. |
| JsonArray | Create an array to store lists of data that are read from json files such as TA names and emails. |
| JsonArrayBuilder | An interface for building and constructing arrays to store data from json files. |
| JsonGenerator | For assisting in writing data to a json file that will be used by the application. |
| JsonObject | For creating an object that will help with interacting with the json data in a given file. Helps to display and manipulate data stored in a json file. Will be used to interact with data dealing with TAs, recitations and course data. |
| JsonReader | For reading data from a given json file to be stored in a json object or json array. |
| JsonWriter | For assisting in writing data to particular json files. This will be used when making changes to a course in the application. |
| JsonWriterFactory | A factory interface which will be used in creating instances of a json writer to write data to json files. |

Figure 2.2.6: API usage of javax.json package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| FXCollections | For assisting in creating the lists that will be used as the choices in combo boxes throughout various selections in the user-interface. |
| ObservableList | Another data structure for storing data in the application. Will be used for storing the list of teaching assistants and perhaps schedule items. |

Figure 2.2.7: API usage of javafx.collections package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| Scene | A container for holding panes that will display the user-interface in the application. |
| Stage | The largest container that also contains the scene and holds all interface components |

Figure 2.2.8: API usage of javafx.stage package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| ActionEvent | Used to deal with actions that take place in the application. Such as when clicking on a button or performing a transaction |
| EventHandler | For dealing with different transactions throughout the application. |

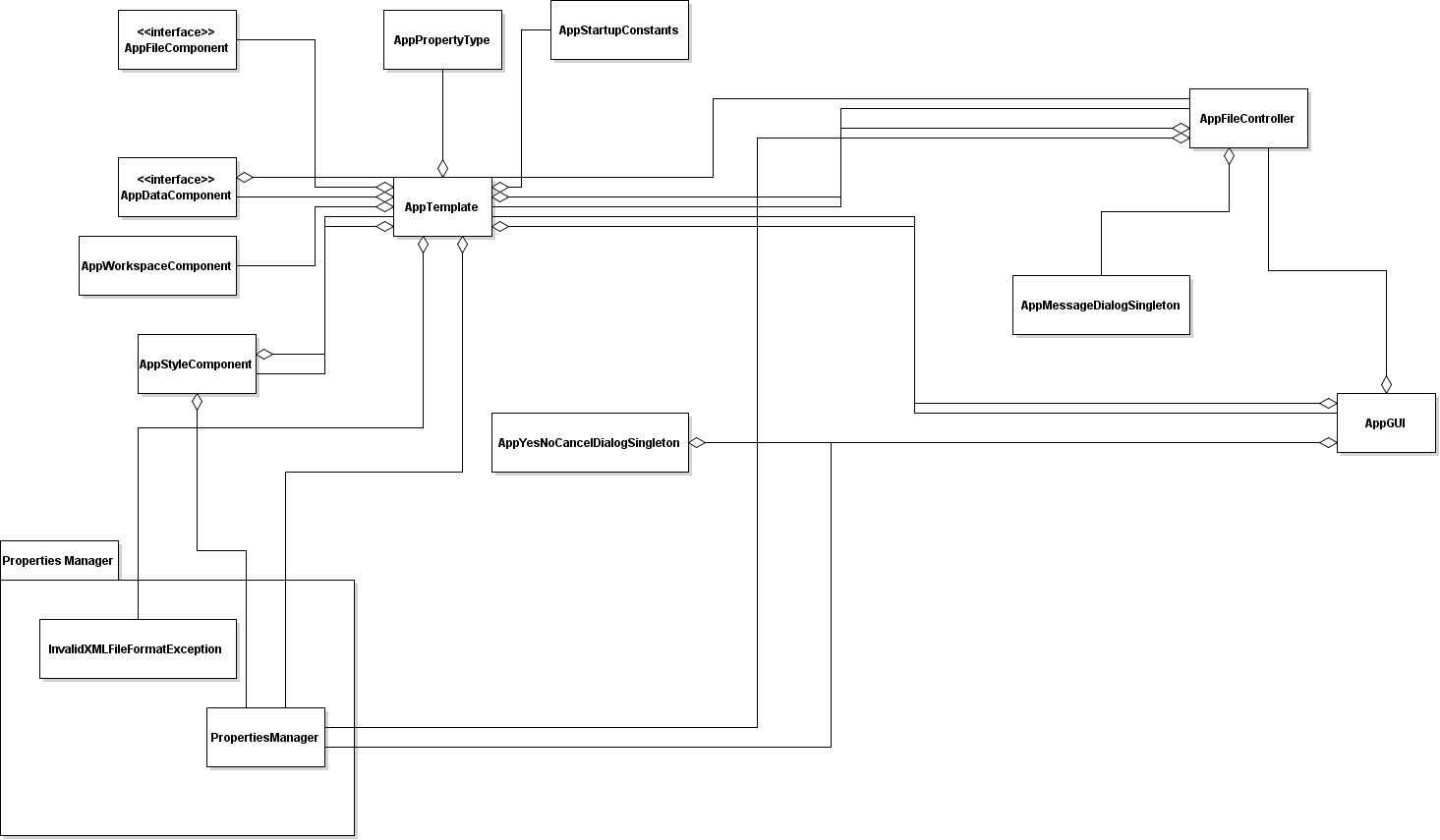
Figure 2.2.9: API usage of javafx.event package

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| Image | Used to help store an image that can be used in the application. |
| ImageView | For displaying an image in the application. |

Figure 2.2.10: API usage of javafx.scene.image package

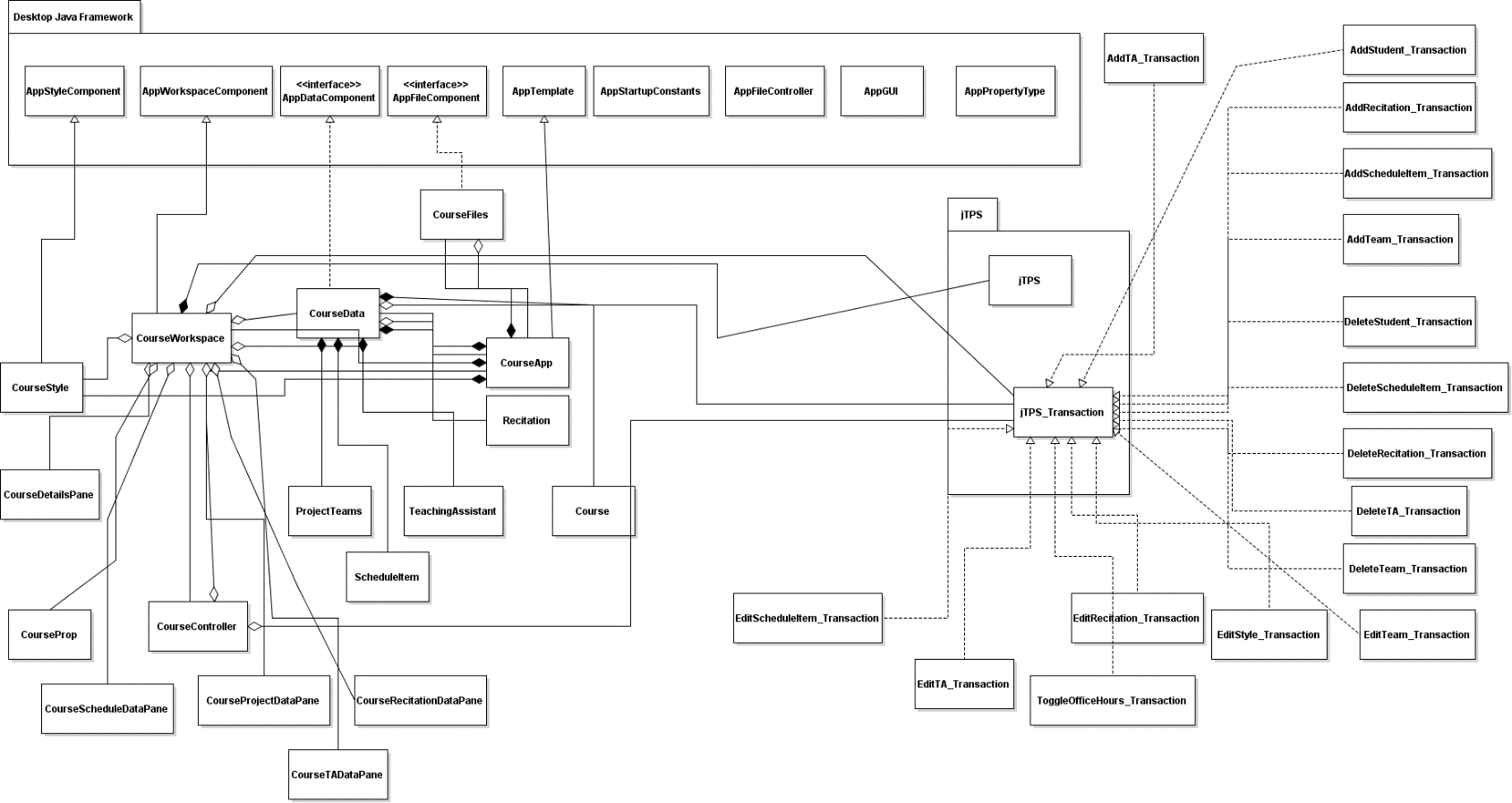
**3 Framework Class-level Design Viewpoint**

This application uses both the Desktop Java Framework and the Properties Manager in this application. The diagram below details out the class design for these frameworks and how the classes will work together.

Figure 3.0: Desktop Java Framework and Properties Manager Framework UML

**3.1 Course Site Generator class-design**

This is an overview of the class relationship of the classes in the Course Site Generator application. It illustrates how each of the various components in the application such as the files, data and workspace work together throughout the entire application.

Figure 3.1: The class diagram of classes in the Course Site Generator application.

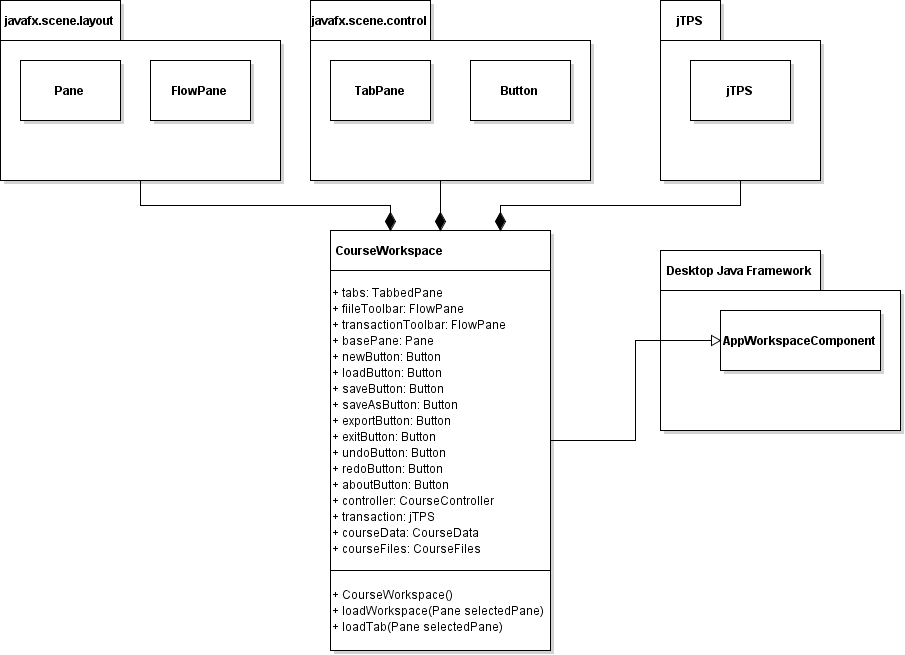


Figure 3.2: More detailed class diagram of the workspace.

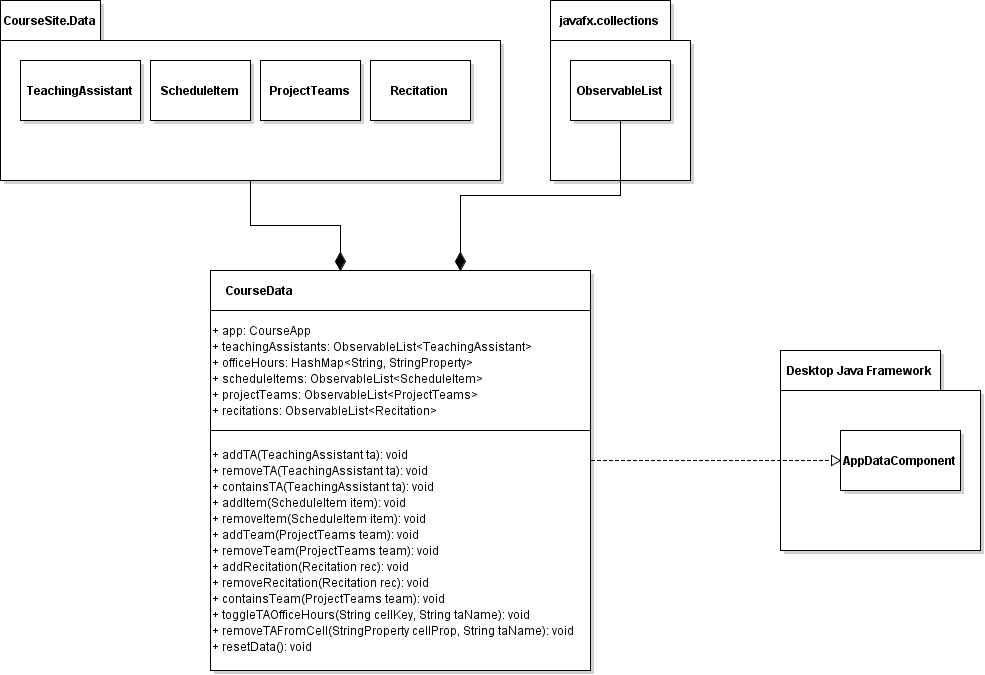
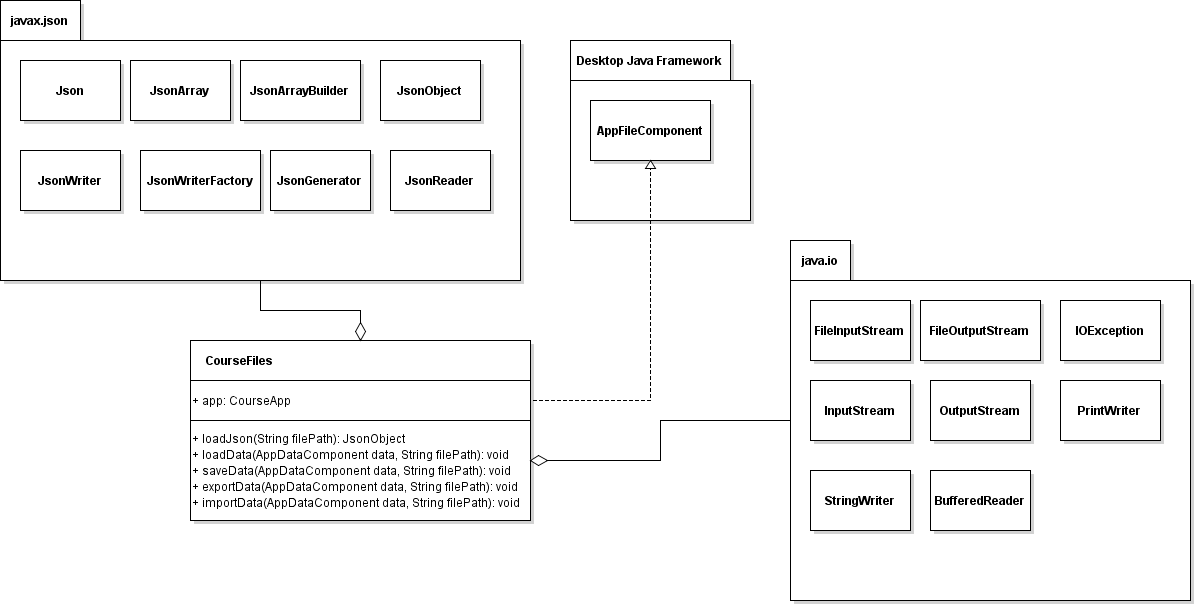


Figure 3.3: More detailed Class diagram of course data.

Figure 3.4: More detailed Class diagram of course files.

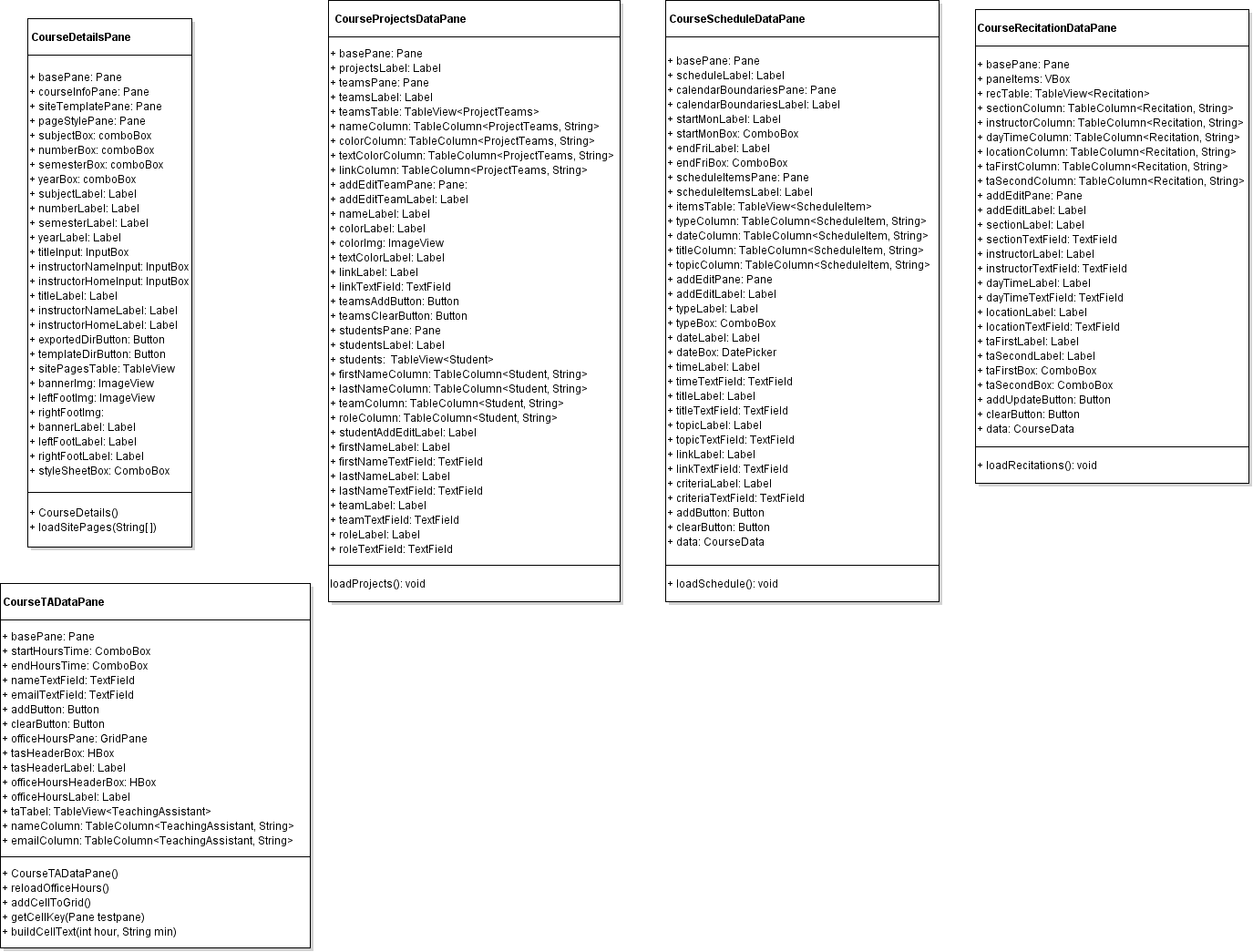


Figure 3.5: More detailed look at the different graphical user-interface classes that will be displayed to the user in each tab.

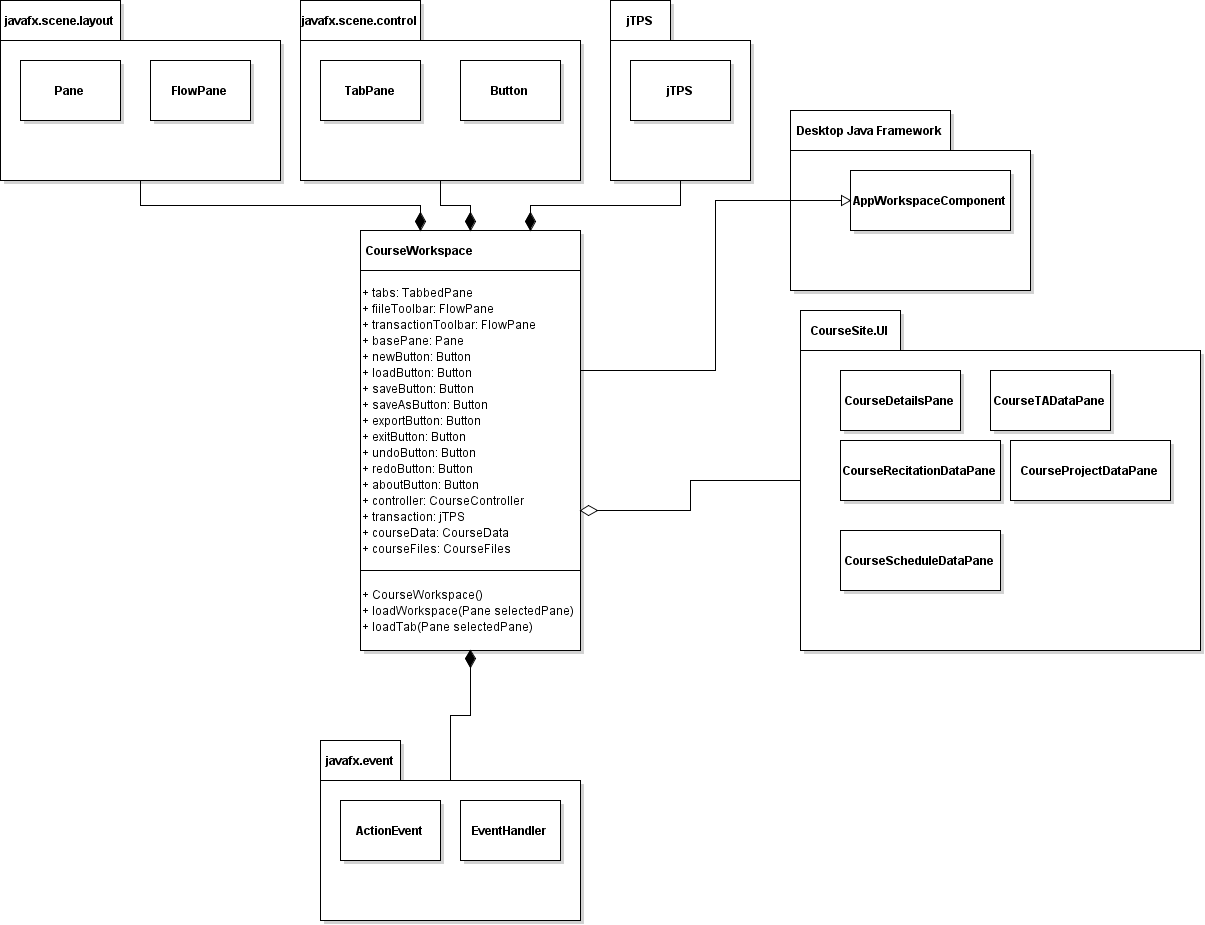
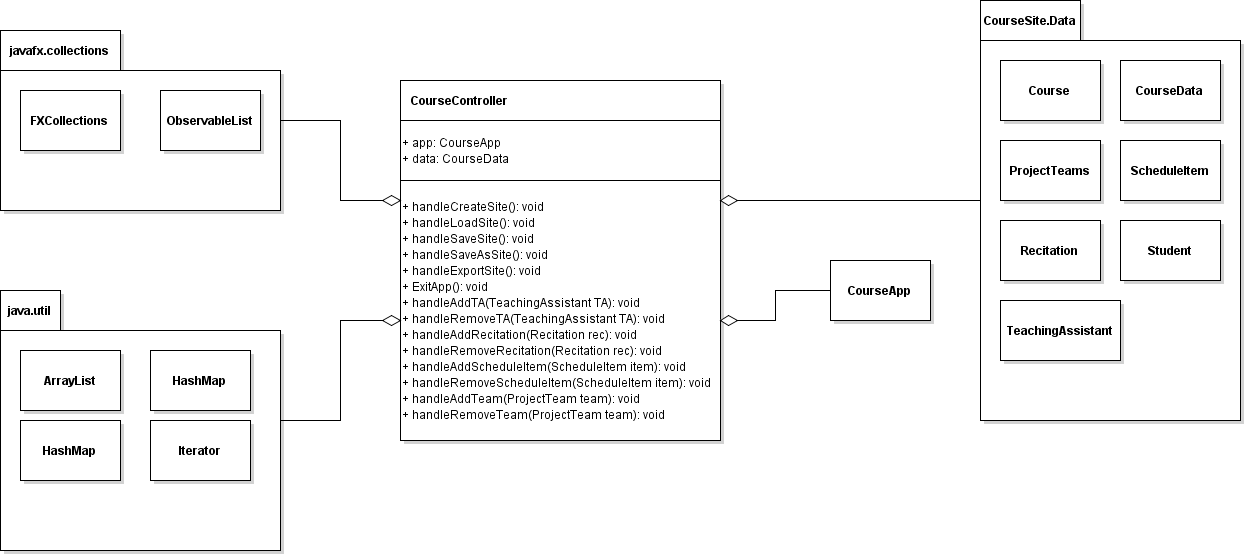


Figure 3.6: Detailed look at workspace along with event handler and gui classes.

Figure 3.7: Detailed look at course controller

**4 Method-Level Design Viewpoint**

These diagrams below specify the method calls that will be taken when certain actions are invoked in the application. All of the specified diagrams below outline the method calls that are defined for each use case.

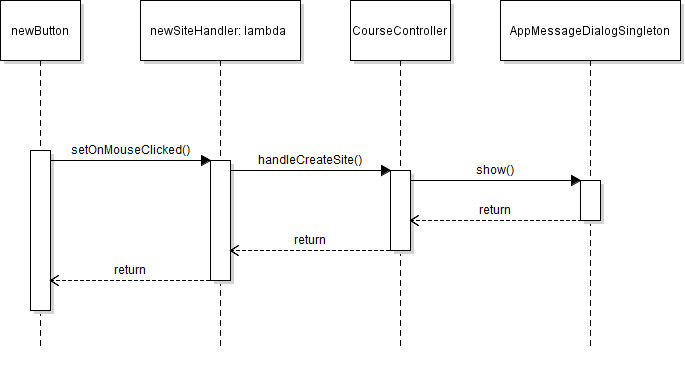


Figure 4.1: NewButton event handler uml diagram

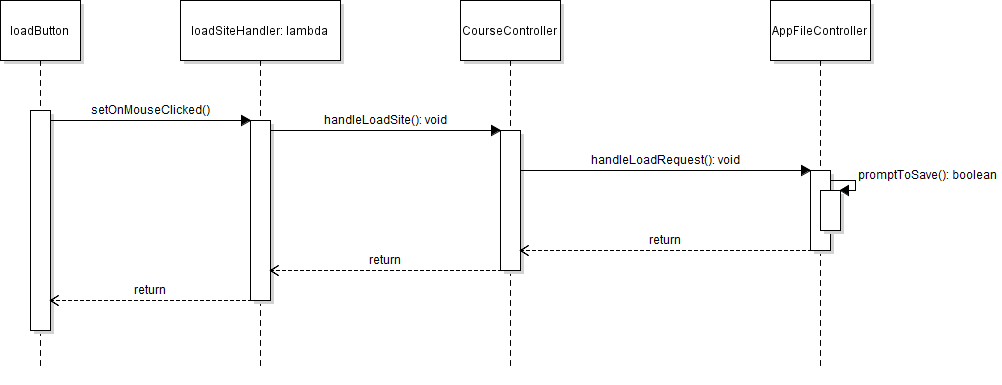
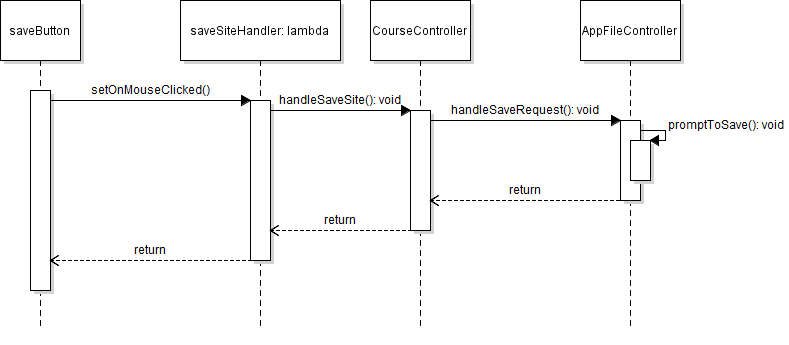


Figure 4.2: LoadButton event handler uml diagram

Figure 4.3: SaveButton event handler uml diagram.

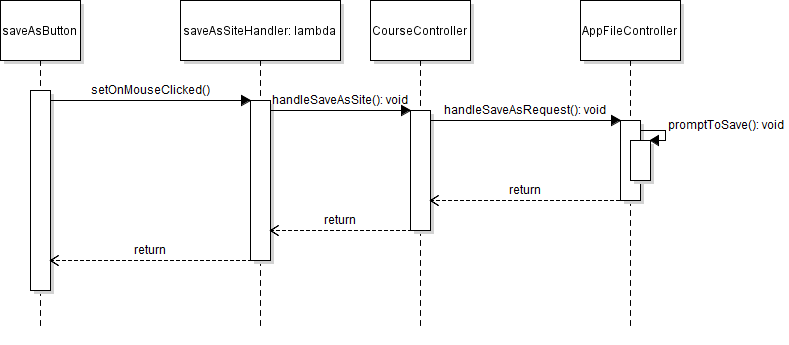
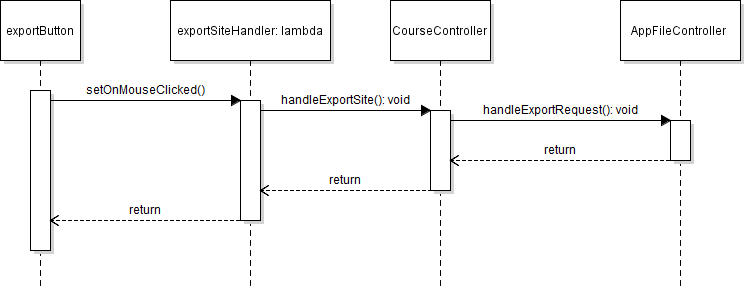


Figure 4.4: SaveAsButton event handler uml diagram.

Figure 4.5: ExportButton event handler uml diagram.

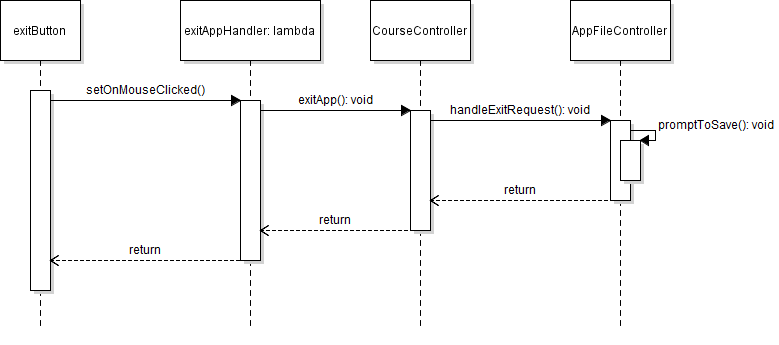


Figure 4.6: ExitApplication event handler uml diagram.

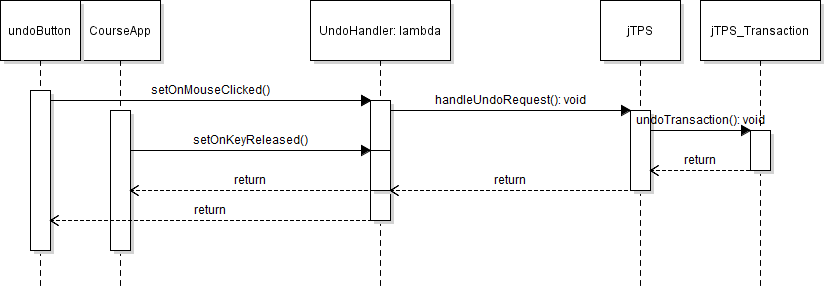


Figure 4.7: Undo event handler uml diagram.

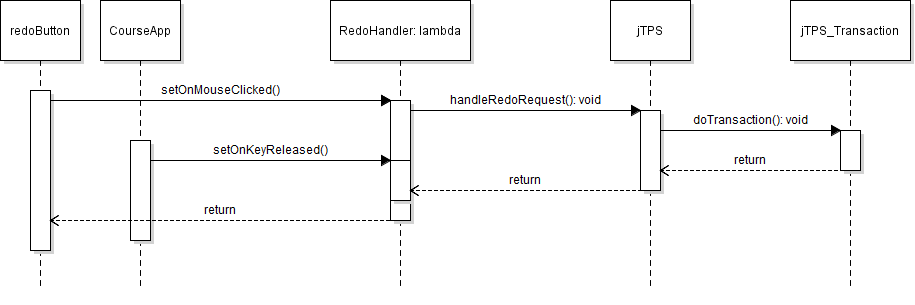


Figure 4.8: Redo event handler uml diagram.

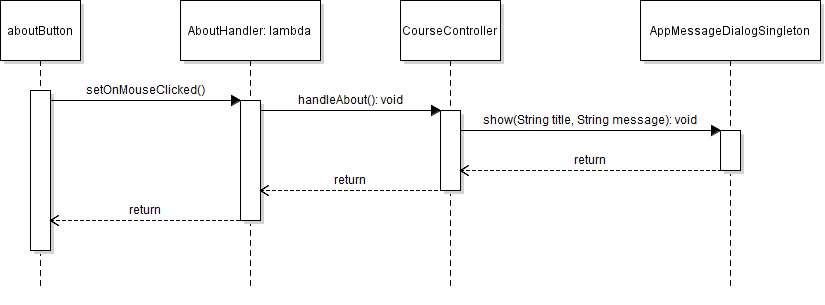


Figure 4.9: About event handler uml diagram.

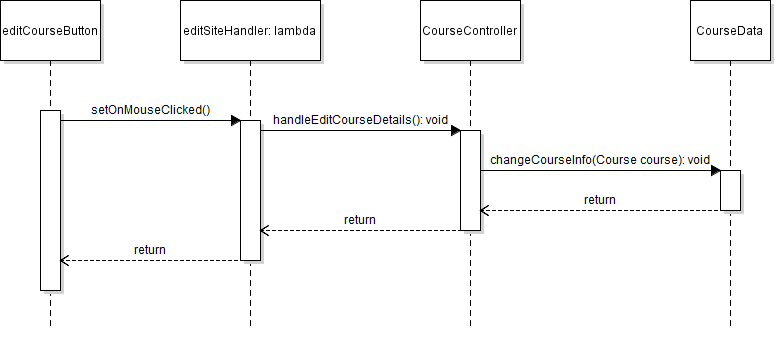


Figure 4.10: Edit course info uml diagram.

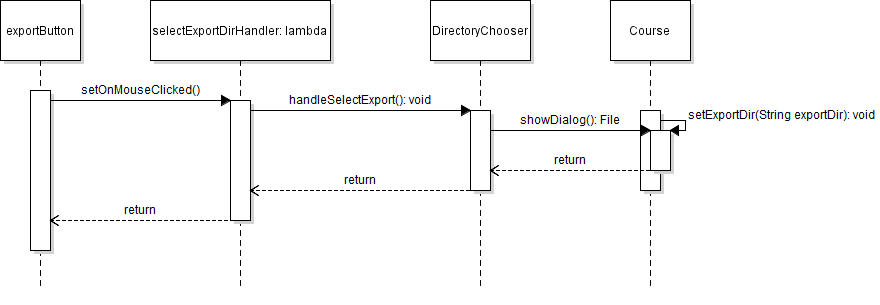


Figure 2.11: Select export directory uml diagram.

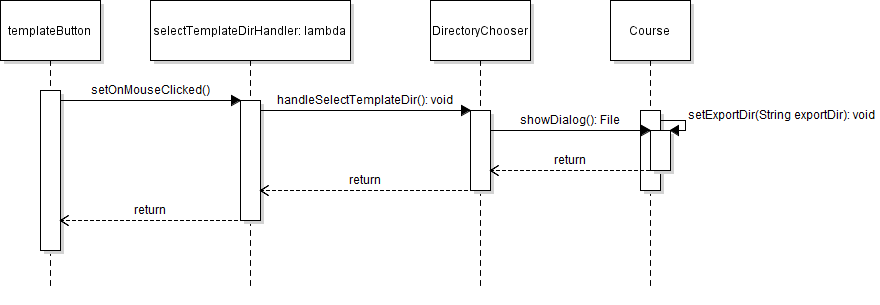


Figure 2.12: Select template directory uml diagram.

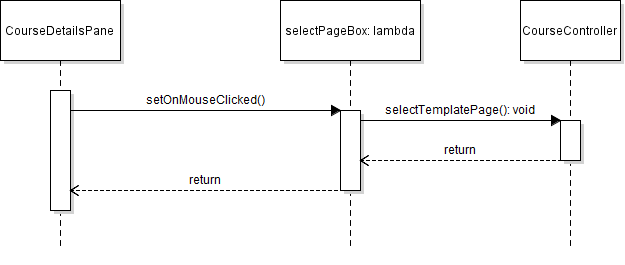
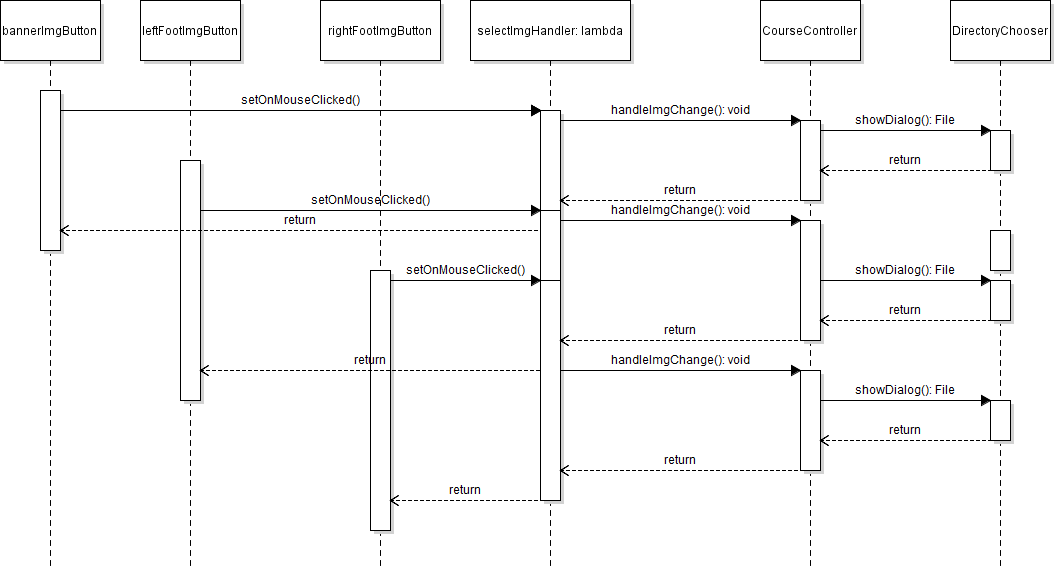
Figure 4.13: Toggle template page uml diagram.

Figure 4.14: Select branding images uml diagram.

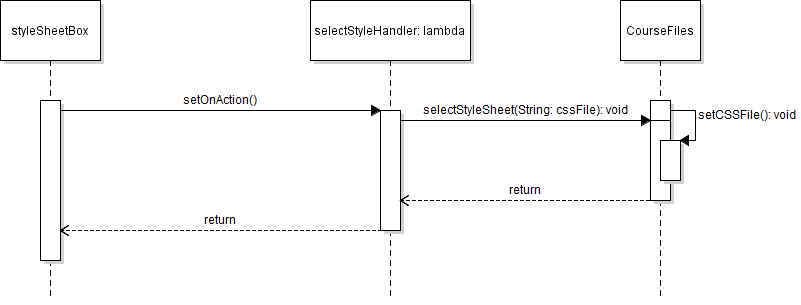


Figure 4.15: Select stylesheet uml diagram.

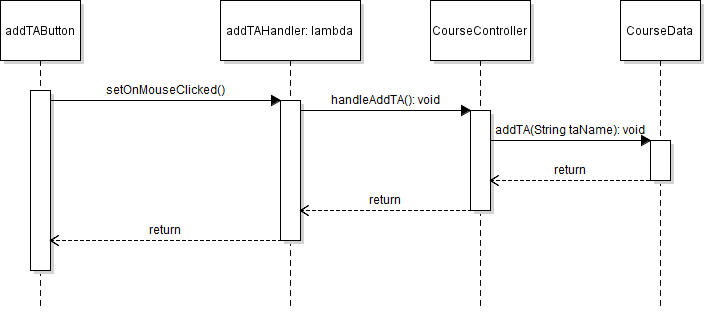


Figure 4.16: Add teaching assistant uml diagram.

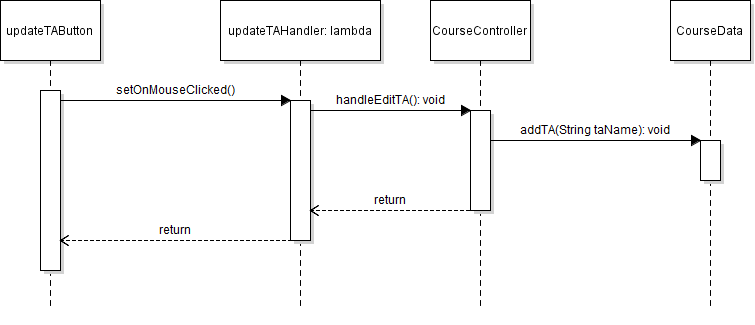


Figure 4.17: Edit a selected teaching assistant uml diagram.

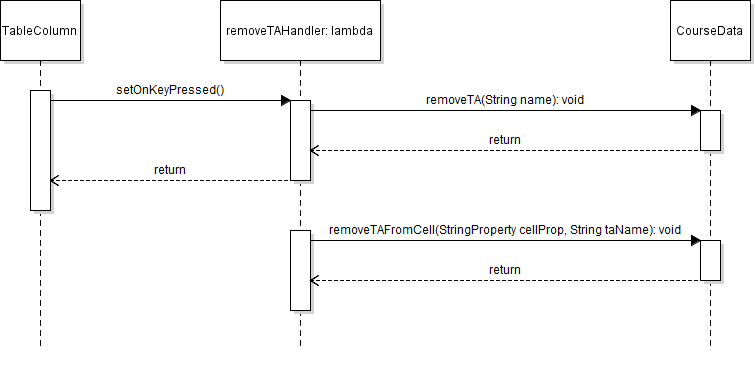


Figure 4.18: Remove teaching assistant uml diagram

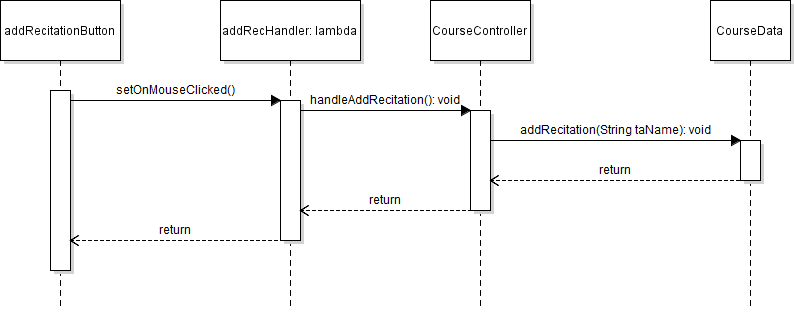


Figure 4.22: Add recitation uml diagram

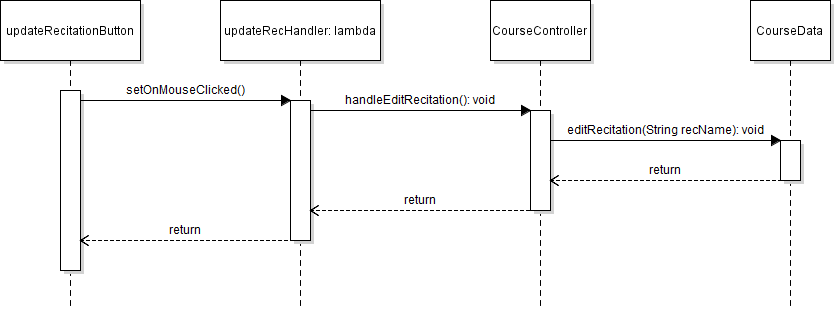
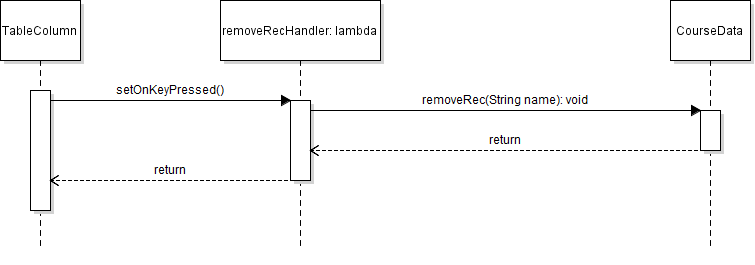


Figure 4.23: Edit recitation uml diagram

Figure 4.24: Remove recitation uml diagram.

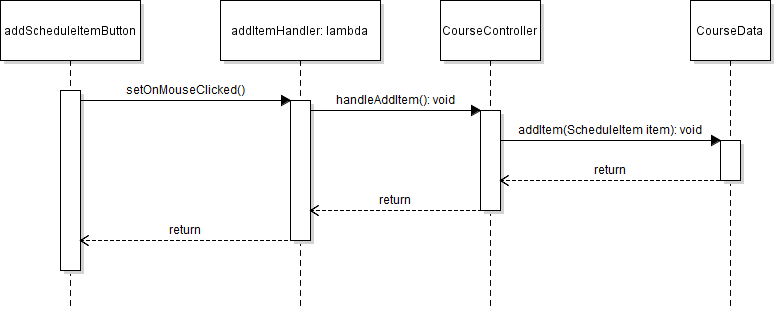


Figure 4.26: Add schedule item uml diagram

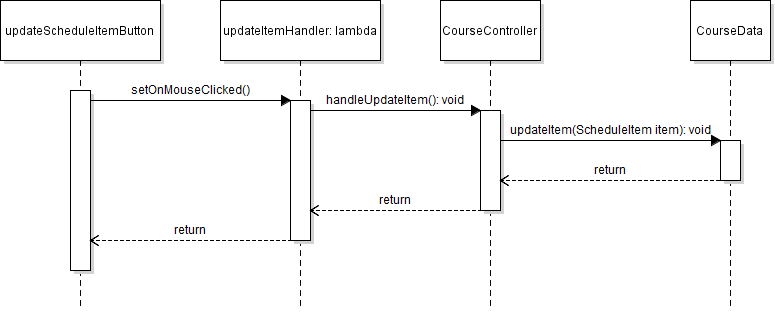


Figure 4.27: Edit schedule item uml diagram

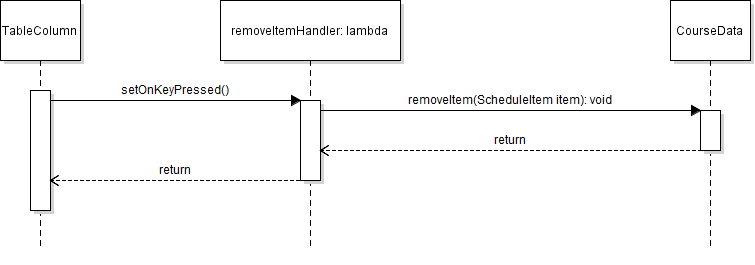


Figure 4.28: Remove schedule item uml diagram

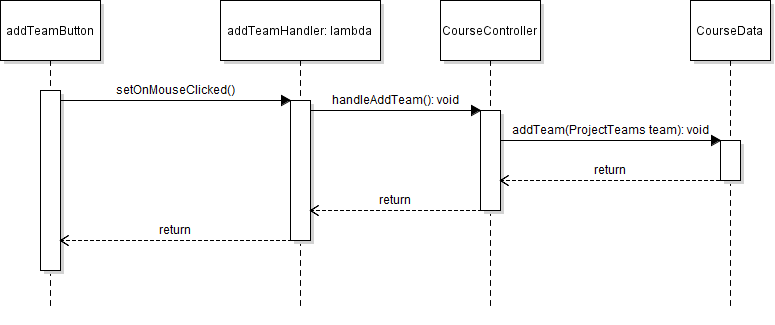


Figure 4.29: Add team uml diagram

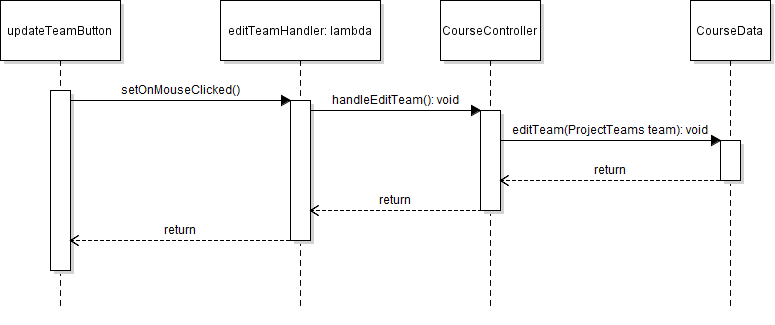
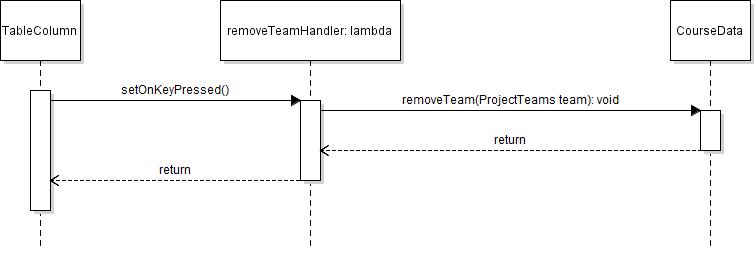


Figure 4.30: Edit team uml diagram

Figure 4.31: Remove team uml diagram

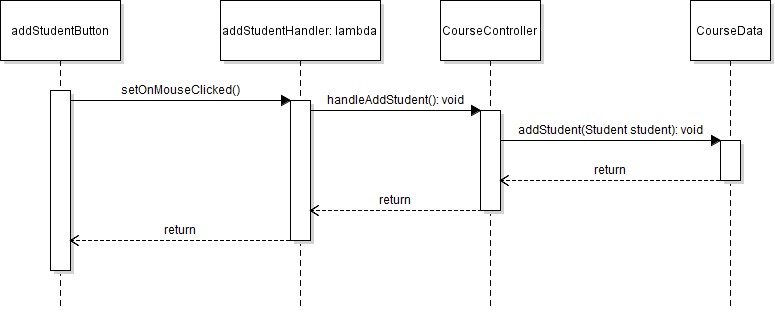


Figure 4.32: Add student uml diagram

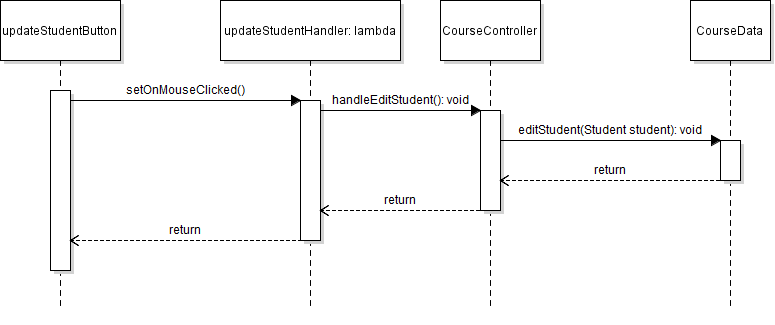


Figure 4.33: Edit student uml diagram

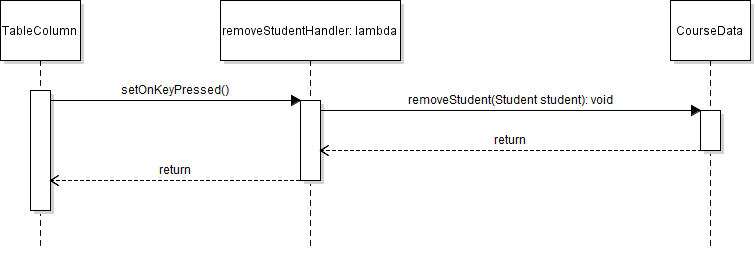


Figure 4.34: Remove student uml diagram.

**5.1: Course Site Generator File Structure (JSON)**

The saved data components in this application involve the objects that are created within the data component of this application. These objects include project teams, recitations, schedule items, teaching assistants and office hours data.

**TeachingAssistant JSON structure:**

**{**

**"startHour":"start\_hour",**

**"endHour":"end\_hour",**

**"undergrad\_tas":[**

**{**

**"name":"ta\_name",**

**"email":"ta\_email"**

**}**

**],**

**"grad\_tas":[**

**{**

**"name":"ta\_name",**

**"email":"ta\_email"**

**}**

**],**

**"officeHours":[**

**{**

**"day":"office\_hours\_day",**

**"time":"office\_hours\_time",**

**"name":"office\_hours\_name"**

**}**

**]**

**}**

**Terms:**

* startHour: defines the start hour for the office hours grid in the TA Data section
* endHour: defines the end hour for the office hours grid in the TA Data section
* undergrad\_tas: the array of all the undergrad teaching assistants in the course
  + name: the name of the undergraduate teaching assistant
  + email: the email of the undergraduate teaching assistant
* grad\_tas: the array of all the graduate teaching assistants in the course
  + name: the of the graduate teaching assistant
  + email: the email of the graduate teaching assistant
* officeHours: the array of all the office hours selections
  + day: the day for the office hours reservation
  + time: the time for the office hours reservation
  + name: the name in the pane of the office hours reservation

**Recitation JSON structure:**

**{**

**"recitations":[**

**{**

**"section": "number",**

**"instructor":"ta\_name",**

**"day/time": "day/time",**

**"location": "location",**

**"teaching\_assistants": [**

**{**

**"name": "name"**

**}**

**]**

**}**

**]**

**}**

**Terms:**

* recitations: the array of all the recitations for that particular course
* section: the particular section number for that particular course
* instructor: the instructor for that recitation.
* Day/time: the time and day for that particular recitation
* Location: the location of the recitation
* Teaching\_assistants: the array of teaching assistants for that recitation

**Schedule Item JSON structure:**

**{**

**"schedule\_items":[**

**{**

**"type": "type",**

**"date": "date",**

**"title": "title",**

**"topic": "topic"**

**}**

**]**

**}**

**Terms**:

* Schedule\_items: the array of all the items for that course
* Type: the type of item for that course
* Date: the date of the event
* Title: the title of the item
* Topic: the topic of the event

**Project Teams JSON structure:**

**{**

**"teams":[**

**{**

**"name":"name",**

**"color":"color",**

**"text\_color": "text\_color",**

**"link": "link",**

**"students": [**

**{**

**"first\_name": "name",**

**"last\_name": "name",**

**"team": "team",**

**"role": "role"**

**}**

**]**

**}**

**]**

**}**

**Terms:**

* Teams: the array of project teams in the course
* Name: the name of the team
* Color: the color for the team
* Text\_color: the text color for the team
* Link: the hyperlink for each team site
* Students: the array of all the students in the team
* First\_name: the first name of the student
* Last\_name: the last name of the student
* Team: the team the student is on
* Role: the role of the student on the team (i.e. lead designer, lead programmer, etc)