CS242 - *Advanced Programming Concepts in Java*

Fall 2015

Final Project

Website Designer

David Mosto

Thanh-Phong Ho

Oumar Ly

James Dapp

December 11, 2015

1 Introduction

The project was to develop a template-driven website designer as a JavaFX desktop application. The application is designed so that a user can create a webpage without any programming or web design knowledge. Similar applications are already available but are encumbered by many features that most users will never use. These extra features overwhelm and confuse the user which ruins the user experience. Our application focuses on simplicity by letting the pre-made templates format and drive the website. The user only needs to add content and the application will do the rest. The application accepts content such as text and pictures and outputs the webpage as an HTML file. The application includes pre-made CSS files to style the website. The user can choose which style they like and the application will pair the CSS file with the HTML.

2 Problem Description

- Design simple and intuitive UI that gives the user enough control to design a webpage
- Get content from the user (text / pictures)
- Categorize content by type (section, image, paragraph, header, footer, etc)
- Allow user to add / edit / and delete content with some flexibility
- Write to HTML file such that it can be re-read reliably
- Allow the user to edit the website and see changes in real-time
- Allow the user to make edits in the middle of an action
- Allow the user to save the project and pick up later
- Create modular UI to describe a modular website
- Implement error handling to minimize probability of a program crash, such as when the user decides to cancel an operation or enters an invalid input

3 Methodology

The design approach was to divide the project into two components: the background code (logic, functions, etc) and the frontend code (UI). This was decided based on the idea that both topics could be developed in parallel. Since this was a fairly large project with a complicated UI and many functions, a Model-View-Controller design pattern was implemented to organize the code, simplify the development process, and create an easier way to debug a large program.

Since the project mainly focused on adding content, such as text and pictures, the UI was heavily based on the concept of a form. Forms are used everywhere to gather information in a simple, systematic, and organized way. Since most users are familiar with forms (usually from filling out physical forms for an organization), this concept was used to develop the rest of the UI. Since all websites are different and many users will want different content on their website with different number of paragraphs, pictures, etc, it was decided that allowing the user

to create their own “form” using UI buttons would be the simplest and best way to populate a website.

HTML differentiates content by their tags in the HTML document. However, the user only sees content as text and images. An object model was created for HTML components so that the application could tell the difference between a heading, paragraph, footer, etc. This decision was made based on object oriented design principles. Because everything is an object, functionality can easily be extended along with allowing code reuse.

4 Implementation

4.1 Task Allocation

David Mosto - Team Leader, HTML Parser / File Handling
Thanh-Phong Ho - UI Design / User Interaction
James Dapp - Project Directory Creator / HTML File Writing / CSS Templating
Oumar Ly - CSS Templating

4.2 Development Tools

Gitlab
IntelliJ version 15
JavaFX Scene Builder version 2
Apache Commons IO - FileUtils

4.3 Code Layout

The source code is wrapped in a package called “master”. The master package has sub-packages where each part of the application is divided. The subpackages are:

4.3.1 **model** - Contains helper classes to perform background information manipulation

a. ApplicationManager - Contains the instance of WebPage and information stored with int. Mainly used so that information can be passed to the WebPage from anywhere in the application (different controllers, stages, scenes, etc)
b. JavaToHTML - Contains the HTML parsing code required to communicate with the HTML files that will be written and editted. It contains code that can read from a *.html file and store the separate parts within a String, List, or List<List> depending on the type of value. Also it contains code that can write to a *.html file the new values that have been generated from the user interface.
c. HTMLStringDefintions - Contains a number of different required String values that have all of the different comments that JavaToHTML will use to parse a *.html file.
d. HTMLObject - Object model for HTML web components. Used to differentiate between different types of input from the user. All of the objects inherit from this

object. Some objects below have been extended to include their corresponding HTML tags. The UI dumps the user input into these objects.

i. HTMLHeader
ii. HTMLList
iii. HTMLImage
iv. HTMLSection
v. HTMLParagraph

e. WebPage - Object model of a webpage. A webpage contains a header, footer, and multiple sections. This object stores the webpage information as the user modifies it.
f. HTMLObjectWriter - Converts the HTMLObject data into actual HTML code using the JavaToHTML class.

4.3.2 view - FXML files generated by SceneBuilder 2.0. All custom dialogs have their own FXML file.

a. MainEditorOverview - UI for the main application window. Contains the webview, listview, add header button, add section button, etc.
b. TempateSelectorOverview - UI for the template selector window. Shows the user the different templates and gives them the buttons to change the templates.
c. HeaderEditor - UI to add the header. Header can be text / image / or both
d. SectionEditor - UI that lets the user create the layout of the website. This UI changes as the user adds content.
e. editorViewStyle (CSS) - Styling for the main application

4.3.3 controller - Contains the event handlers / controllers for each FXML file

a. MainViewController - Event handler for the UI buttons / menu buttons / webview / listview in the MainEditorOverview FXML file
b. SectionSelectorController - Event handler for the SectionEditor FXML file
c. TemplateSelectorController - Event handler for the TemplateSelectorOverview FXML file
d. HeaderSelectorController - Event handler for the HeaderEditor FXML file
e. MainApp - Run this file to start the whole application

4.3.4 images - Contains the template preview images. These images are displayed in the template selector as “preview” images.

4.3.5 path - Contains the different templates available to the user. When a template is selected, the application will copy the corresponding CSS and HTML file from this pool into the user’s working directory.

4.4 Testing Scheme

The program was tested by essentially by going through each button and each menu combination in an attempt to design the website. If anything crashed, then that error would be fixed. Other testing procedures include trying to enter letters in a box that only expects numbers, trying to add a non-image file as the header, and canceling dialogs. Most of these testing procedures printed out Exceptions into the console. The code was then revised by wrapping "dangerous" operations in try-catch blocks and writing appropriate Exception handlers.

5 Discussion

David Mosto:

The main challenge I faced was figuring out a way to structure the HTML file that will always allow the JavaToHTML parser to break down the file whenever it is told to do so. This structure would need to be written so that the program can read from the HTML file, break down different parts for easy use within the GUI, receive new values from the GUI, and change the HTML file accordingly whenever there is a content change. For the structure of the HTML file I used HTML commenting, where an example would be *<!--$headerBegin-->* and *<!--$headerEnd-->*. *$headerBegin* would mark within the file where the parser should look for the beginning of the content for Header and *$headerEnd* would mark the content's ending. This allows for each individual part to always be the surrounded by the same code no matter which HTML file is read, as long as the structure is followed.

Originally I had planned that the content within the HTML file would simply make up of a title, navigation, and basic content. Now the HTML file can contain a title; navigation; header; different sections that contain the bulk content of the webpage and can in itself contain section headers, paragraphs, unordered lists, ordered lists, and images; and a footer. When covering the sections, I had to plan out extensively all of the possible options that could be added to the HTML file, and the way I found most efficient to store them would be to store all of the section headers within its own Array List and to store the rest of the content within a List of ArrayLists. Each section will mark a new list and within each list is the individual content values. Initially instead of lists I had stored everything within Arrays because of my C & C++ background. This proved to be an inefficient process since Java allows for dynamically allocated Lists, so I transitioned my plans to work with Lists instead.

Thanh-Phong Ho:

The main challenges I faced was trying to figure out how SceneBuilder works along with how Java deals with FXML files. Initially when the project first started I thought the UI would be simple enough and that the time needed to learn SceneBuilder would not be worth it. I attempted to use SceneBuilder but learning the program was frustrating since the tutorials and the UI were not as straightforward as expected. I gave up shortly only to return to SceneBuilder. After attempting to draw up the UI using straight Java code, I found it extremely tedious to make minor changes. The hardest part was learning how Java wants people to use SceneBuilder with JavaFX applications.

Another challenge was trying to manage multiple stages and multiple scenes. JavaFX does not have a built in tool to handle multiple stages and multiple scenes. Most solutions on Google involved writing a custom framework using a Map to flip each scene/stage. After seeing how much code and background was required (and how little time was available) I decided to dump every event handler into one controller. I soon discovered that Java creates a new instance of the controller each time it reloads the FXML file. This became a problem while trying to save important information within the controller. After more Googling, I found a solution that used the MVC design pattern. Not only did MVC make it extremely easy to debug the UI, it also fixed most of the problems I encountered such as some UI elements not being displayed properly and controllers being re-instantiated when not told to do so.. If I had gone with the smarter design choice (MVC), this application development process would have been much smoother and faster.

James Dapp:

I designed the controller code to tie together the GUI with the JavaToHtml code. The main challenge with this is there was no overall diagram or design notes other than a drawing of the GUI prototype. So it became difficult to connect everything together when new features were always being added or modified and I wasn't sure where they were supposed to go in the first place.

Another challenge was the Version Control Software. For our VCS, we chose to use git with IntelliJ, an IDE that I have never used before. I am used to Eclipse, and IntelliJ was very intimidating the first time that I used it. IntelliJ has many features that Eclipse doesn't, including significantly more options when dealing with Git. I haven't used Git too much, and I wasn't sure what would happen in the event of conflicts or a merge. More than once our group became confused when we thought something was pushed when it wasn't, or if something was pushed that caused many conflicts. I favored using the Shelve feature to resolve conflicts. I could shelve my changes, pull the new code, and then using the diff view I could re-import my changes one file at a time to make sure everything merged correctly.

Oumar Ly:

I designed the CSS templates 3 and 4.