EasyFx

Project Submitted In the Context of the Course
Info3301 **Software Engineering**

Developed By

Jad Mrad

George Raed

LEBANESE UNIVERSITY
FACULTY OF SCIENCES I
DEPARTMENT OF COMPUTER SCIENCES

2018 - 2019

Acknowledgements

We would like to express our deep gratitude to Professor Kamal Baydoun, the course instructor, for providing a solid and thorough basis on which to build our project and subsequent report on in addition to guiding us through the software engineering process step by step. Furthermore we would like to acknowledge with much appreciation the support provided by the informatics department lab assistants with special thanks going out to Doreid Dagher for providing access to the labs in which we conducted much needed testing.

Abstract

The purpose of this project is to provide to the Java community a platform to build JavaFx projects with a degree of flexibility never before seen. EasyFx is a program that allows users to create, in a drag & drop fashion, complex UI with no coding needed. We seek to also solve a problem that so far remained unaddressed which is the lack of a proper visual scripter that provides a choice of output format. To achieve our set goals we set out to get the target functionalities with version 0.x (EZFXLite) then later build from the ground up on our main release in versions 1.x and beyond.

Table of Contents

Table of Figures

Table of Tables

Chapter I - Introduction

I.1. The business domain

This project fits into the software development domain and is specifically directed towards Java programmers working on JavaFx or GUIs in general. We believe this is an ideal community to target as it is sizable with 3 Billion devices running java and its sizable market share at 22.9% as can be seen in figure 1. We also chose JavaFx as a platform to build upon as it is the most recent officially released framework for creating graphical user interfaces with updates coming frequently and as recent as 2 hours[1] before this report was written.

Worldwide, Jan 2018 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.9 %	-0.9 %
2		Python	21.0 %	+5.6 %
3		PHP	8.6 %	-1.9 %
4	↑	Javascript	8.4 %	+0.3 %
5	↓	C#	8.2 %	-0.6 %
6		C	6.7 %	-1.1 %
7	↑	R	4.0 %	+0.3 %
8	↓	Objective-C	3.9 %	-1.1 %
9		Swift	3.2 %	-0.3 %
10		Matlab	2.3 %	-0.7 %

Figure 1 Programming Language Market Shares - Early 2018

In an ever-expanding community we believe to have stumbled upon a requirement that no one has provided yet based on our research. As will be discussed in detail below, any Java developer will in one way or another need our program be-it for large scale collaborations or a simple personal project.

[1] JavaFx GitHub Release

I.2. About the modeled application

The program is intended to be used by anyone from beginners to experienced developers to design and implement JavaFx completely code-free. It can also be utilized by anyone who doesn't know how to code which is useful on clients during the requirement gathering stage. With many more possibilities, the community needs such a code-free program not to mention the blueprints system that is so far unseen for JavaFx.

2.1. Users Persona

Some key cases where our product would be optimal:

- Java developers creating GUI assets or prototypes with integrated logic
- Designers setting up a projects layout without requiring any application code
- Newcomers to JavaFx that wish to explore its potential
- Clients to show Software Engineers what they want
- Swing and/or AWS users switching over to JavaFx syntax

2.2. Analysis of the Existing Similar Programs

There are quite a few programs and frameworks that could be considered alternatives or competitors to ours yet none of them carry the functionalities that EasyFx provides, mainly the handler management blueprints.

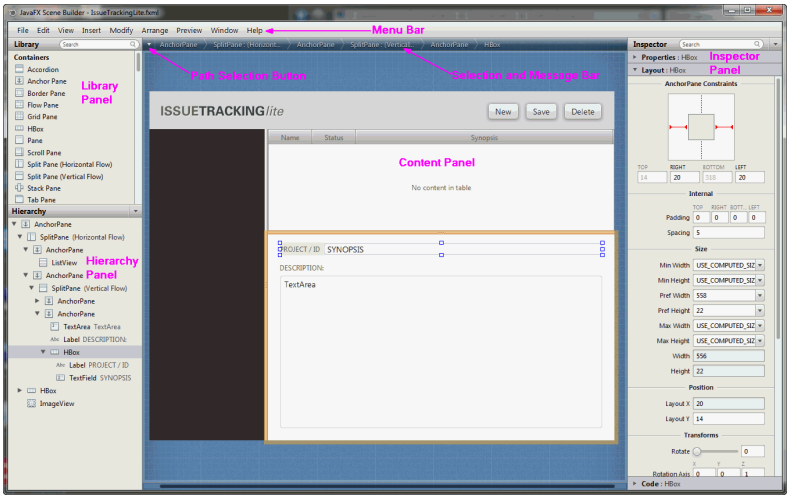


Figure 2 - Oracle SceneBuilder 1.1 Early Release

Oracle began SceneBuilder in late 2013 with the 2.1 release of JavaFX as a means of lightening the load of having to compile and execute code to view UI progress and check for any faults. It came at an early stage of Javas rise to dominate the programming world and thus was unopposed.

Though we view this as a gift to the community from Oracle and a major improvement that certainly made many projects easier, their decision to push their own Markup Language FXML, released in 2011, as the only possible output type is what prompted us to plan and go through with EasyFx.

	Last Update	Up to Java version	Accepted Inputs	Output	Blueprints
Oracle Scene Builder 1.1	October 2013	Java 7	None	FXML	No
Oracle Scene Builder 2.0	April 2014[2]	Java 8	FXML	FXML	No
Gluon SceneBuilder	June 2018	Java 10	FXML	FXML	No
EasyFx	January 2019	Java 10	EzML / FXML	JavaFx / EzML / FXML	Yes

Table 1 Comparison of Existing Similar Programs & Frameworks

As is apparent in Table 1, our only proper competition is the Gluon continuation of Oracle's SceneBuilder which took over post-2014 releases after Oracle halted development and favored a source-code only release. Though the developers over at Gluon have been doing a great job maintaining it, they continue to rely solely on FXML and have not provided a way for experienced developers to extract a proper JavaFx variation of their design.

We simly managed to create a more time efficient solution that provides a wider range of functionalities [See Setion IV.3 for Benchmarks] for the community.

[2] Oracle SB 2.0 Release notes

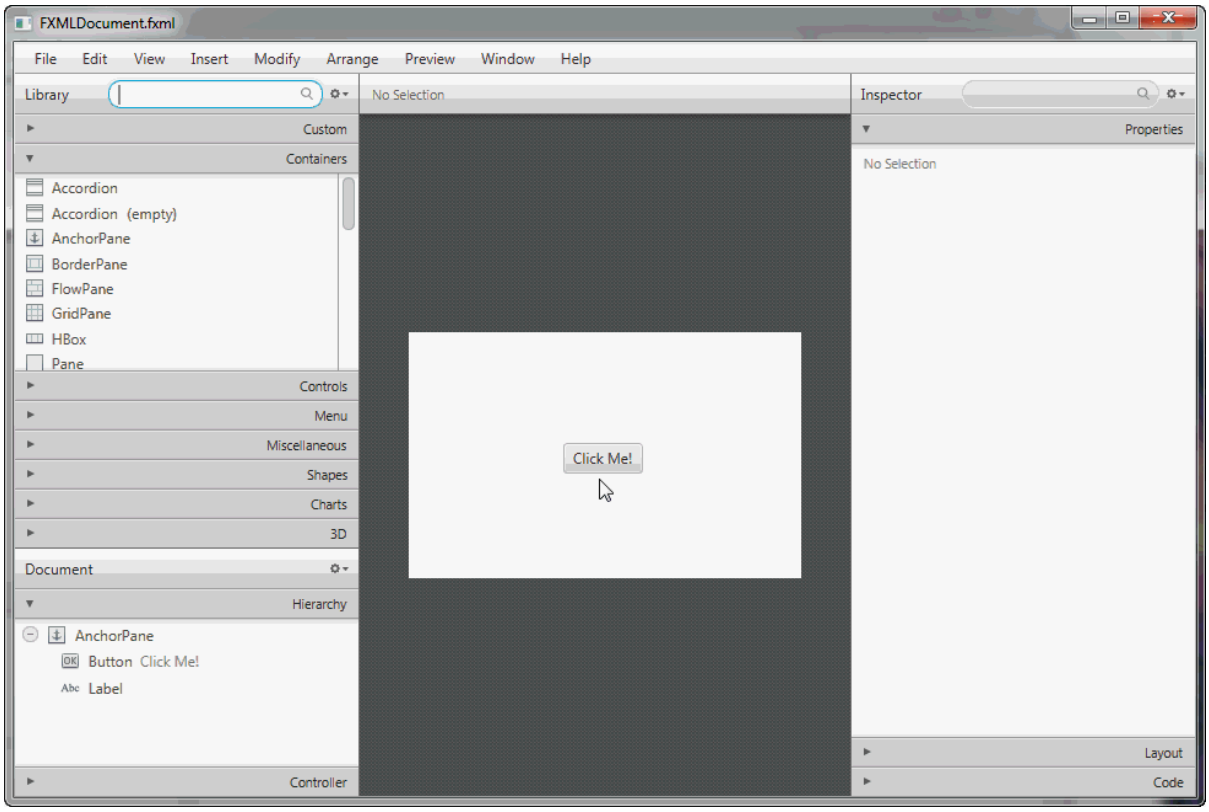


Figure 3 - Gluon SceneBuilder 8

FXML simply takes away the extended libraries and functionalities that make JavaFx unique among its outdated rivals by replacing intricate Object Oriented syntax with a Markup Language. Overall what makes our program diverse is allowing the user a choice of output and integrating an essential part of development, event handling, in the form of blueprints.

I.3. Plan of the document

This document goes into detail on all stages of designing and maintaining this product from the basic requirements engineering process through development, prototyping, testing, and preparation for an official release. Future plans and mappings of projected evolution will also be included as well as extended versions or requirement gathering procedures. All these will be discussed in detail below.

Chapter II - Requirement Analysis and Specification

II.1. Introduction

The following is a mildly modified condensation of the requirements collected through the initial meeting **[see addendum M1]** and follow-up meeting, phone calls, and questionnaire **[see addendums M2 through M4 and MQ]**. Unnecessary data has been omitted for the sake of simplicity. Note that the client has been fictionated to better simulate the requirements gathering process. Our true client remains the Java developer community as a whole.

"EasyFx is a program that makes programming with JavaFx easier and code-free. Users should be able to drag and drop buttons, fields, and shapes onto the canvas. The user can edit the details of these items and the items can then be dragged around to change their position and can also be deleted. Selecting one of them sends the user to a blueprint-like page where he can manage what happens when interacting with each item (hovering, clicking, etc...).

Users can start on an empty canvas, use a pre-built template, or import their own projects that can be FXML or previous projects made on the program. Closing the program will first prompt the user to save their changes or cancel. Saving is done into a custom file type made for the program, as FXML, or a text file holding extracted JavaFx.

All this can be done using a guest account. Users can also register a local account with an email address. This will allow them to save their progress directly onto the local account instead of a file for ease of use. They are prompted upon creating an account to sign up for the newsletter to receive information about future releases. Registered users can access a settings tab to manage their account details, logout, clear data, or sign up/out of the newsletter. They can also switch between a light and dark theme.

The system should ensure that user data, encrypted locally, can only be accessed by the user himself and no one else. Passwords should never be viewable at the point of entry or any other time. Users shall receive notification of any profile change or unauthorized login via email.

Use a custom file extension to save created projects. The program itself shouldn't be very big and should fit in a portable, executable JAR file."

II.2. Requirements Analysis

The following is the extraction of functional and non-functional requirements from the passage.

“Custom file type made for the program” has been replaced with “EZML”, our program’s own markup language that constructs the UI.

2.1. Functional Requirements

Guests can:

- Register a local account with an email address. **Feasible**
- Drag and drop buttons, fields, and shapes onto the canvas. **Feasible**
- Edit the details of these items. **Feasible**
- Items can then be dragged around to change their position. **Not Feasible** **[see addendum M3]**
- Delete items. **Feasible**
- Manage events with blueprints functionality. **Feasible**
- Start an empty project. **Feasible**
- Start a template project. **Feasible**
- Import EZML or FXML project. **Feasible**
- Close the program. [Customized closing sequence] **Feasible**
- Save/Extract as EZML, FXML, or JavaFx. **Feasible**

Registered users can:

- Save their progress directly onto the local account. **Feasible**
- When creating an account sign up for the newsletter to receive information about future releases. **Feasible**
- Access a settings tab to manage their account details, logout, clear data, or sign up/out of the newsletter. **Feasible**
- Receive notifications of any profile change or unauthorized login via email. **Not Feasible** **[see addendum M4]**
- Change the products theme. (light/dark) **Feasible**

2.2. Non-functional Requirements

The system should:

- Allow users to drag & drop items. **Usability**
- Ensure that user data, encrypted locally, can only be accessed by the user himself and no one else. **Confidentiality**
- Passwords should never be viewable at the point of entry or any other time. **Access Security**
- The program itself shouldn't be very big and should fit in a JAR file. **Portability**
- Run without using high maintenance servers **Development Requirement**

II.3. Specification

3.1. Use Case

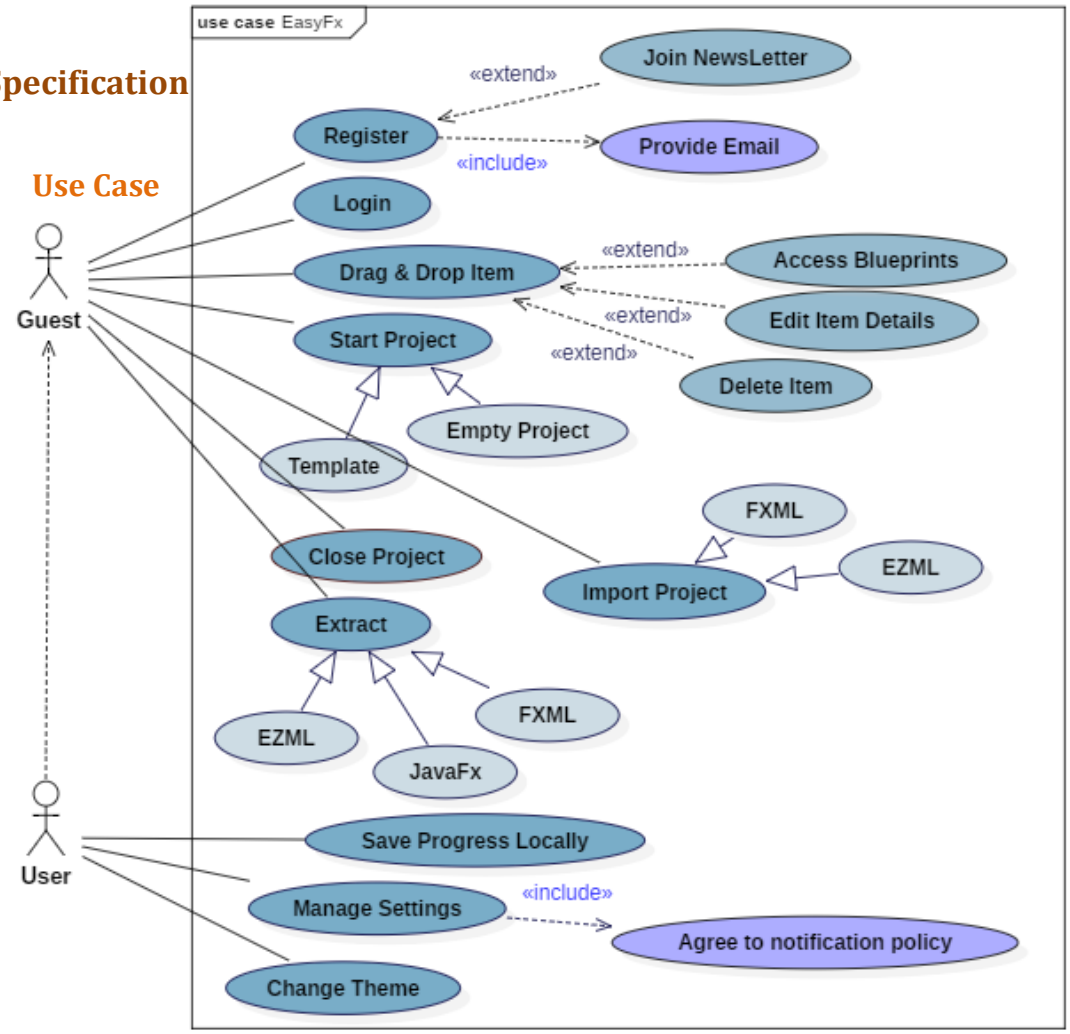


Figure 4 - Use Case Diagram

3.1. Use Cases Textual Description and Sequence Diagrams

Priority		Details
1	Cosmetic	- Serves no purpose
2	Add On	- Extension that runs alongside main functionalities
3	Neutral	- Secondary functions
4	Semi Essential	- Required for proper functionality. Failure won't cause error
5	Essential	- Failure will most probably cause error state

Table 2 - Priority Description

1 "Import Project" Use Case:

Number	#UC_DTD_001	
Name	Import Project	
Summary	Extract EZML project from input	
Priority	4	
Post-conditions	Project is constructed and opened	
Primary Actor	User	
Trigger	Import option from File menu	
Main Scenario	**Step**	**Action**
	1	User browses for target file
	2	System validates file extension
	3	System constructs UI from project
Extensions	**Step**	**Branching Action**
	2a	File extension is invalid
	2b	System notifies user: "Invalid File Type"
	2c	Import cancelled
	3a	System runs into error extracting data
	3b	User is notified of the error
	3c	Import cancelled

Table 3 - DTD of the "Import Project" Use Case

2 “Extract” Use Case

Number	#UC_DTD_002	
Name	Extract	
Summary	Export created project into a specific format	
Priority	4	
Preconditions	**N/A**	
Post-conditions	User now has a file in a specific format of his creation	
Primary Actor	User	
Trigger	Export option from File menu	
Main Scenario	**Step**	**Action**
	1	User selects save location (Browse)
	2	User selects format (EZML, FXML, or raw JavaFx)
	3	User enters file name
	4	System checks if a file of that name/extension exists in target
	5	User provides preferred save location
Extensions	**Step**	**Branching Action**
	3a	File with same name exists, prompt user to over-write file
	3b1	User agrees. File is over-written
	3b2	User denies. Cancel extraction

Table 4 - DTD of the “Extract” Use Case

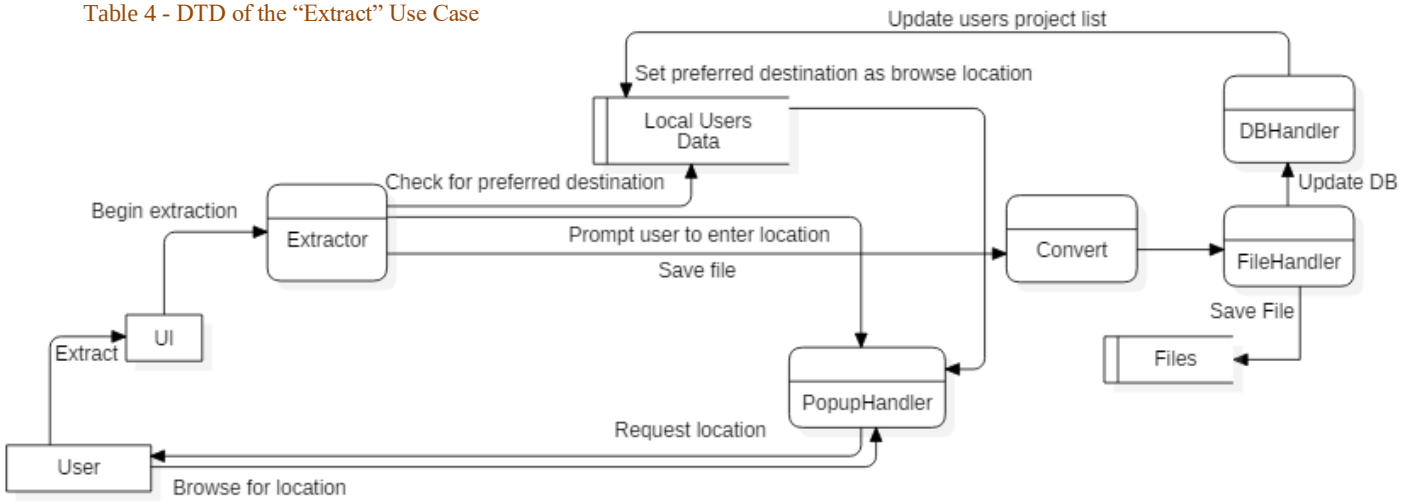


Figure 5 - DFD of "Extract" Use Case

3 "Register" Use Case:

Number	#UC_DTD_001	
Name	Register	
Summary	Register local account and input details	
Priority	2	
Post-conditions	User has an active local account in the database	
Primary Actor	User	
Trigger	Register button from Login scene	
Main Scenario	**Step**	**Action**
	1	User provides name and email
	2	Email is verified (?)
	3	User provides a valid password
	4	System checks password strength
	5	User provides preferred save location
	6	System prompts user to join newsletter
	7	User accepts
	8	User added to news list
Extensions	**Step**	**Branching Action**
	2a	System notifies user: "Email is not valid"
	2b	User prompted to retry step 1
	3a1	System notifies user: "Password is weak"
	3b1	Passwords do not match
	3a2/3b2	User sent back to step 3
	6a	User denies
	6b	User set to be notified about newsletter on next login
Open Issues	I001_1	Email verification system not confirmed functional yet

Table 5 - DTD of the "Register" Use Case

II.4. Addendums

4.1. Meetings:

The fictional company InsertCompanyNameHere.inc will henceforth be referenced as ICNH

The team working on the project EasyFx will henceforth be referenced as "the developers", "the development team", and "devs".

"The program", "The system", "The framework", and "The project" are all references to EasyFx.

1 M1: Initial conception meeting

Location: Head Office of ICNH

Date: 22/10/2018

Time: 10:00 AM to ~12:00 AM

Attendance:

- EasyFX Team (2/2)
- ICNH CEO, CSO, & IT Manager (3/3)

Agenda Items:

- Discuss EasyFX general functionalities & expectations [Success]
- Plan EasyFX UI design [Partial success, finalization postponed see addendum M2]
- Agree on timeline [Success]

Action Items:

- The developers will provide an in-depth documentation report on the software including all meetings conducted
- The developers are tasked with providing a handy user-manual alongside the product
- The agreed upon functionalities (see II.1) are to be achieved by the devs before the agreed upon dates
- ICNH will provide full access to their own development team for testing purposes
- ICNH will provide limited access to their database facilities for integration purposes

2 M2: Follow-up in depth meeting

Location: Starbucks

Date: 28/10/2018

Time: 02:00 PM to ~03:00 PM

Attendance:

- EasyFX Team (2/2)
- Selected ICNH Developers (2/3)
- ICNH Graphic Designer (1/1)

Agenda Items:

- Plan EasyFX UI design [Success, see II.5 Design Process]

Action Items:

- The program will feature a night/dark mode
- The primary colors of the program will be an office grey on red brick tint
- The design will be similar to that of [Name Omitted For Trademark Reasons]

3 M3: Phone call regarding feasibility

Action Items:

- Drag-ability of nodes outside set containers has been omitted for technical reasons
- More containers were requested to fill drag-ability gap

4 M4: Final pre-release meeting

Action Items:

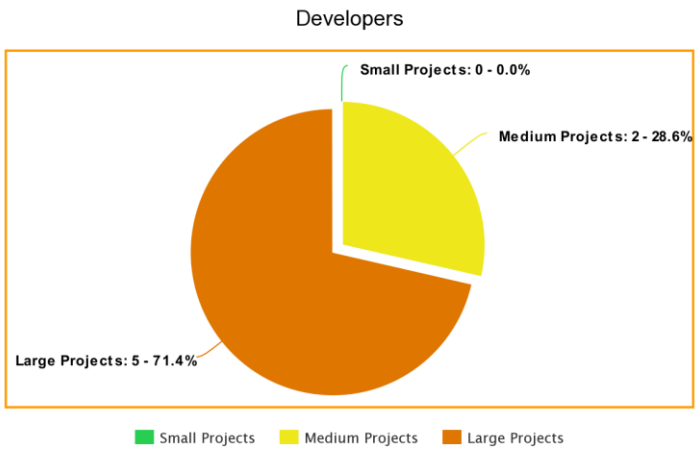
- E-mail configuration removed from deliverable(s) to be added in by ICNH teams later
- Demos tested and approved
- DB integration test run and approved

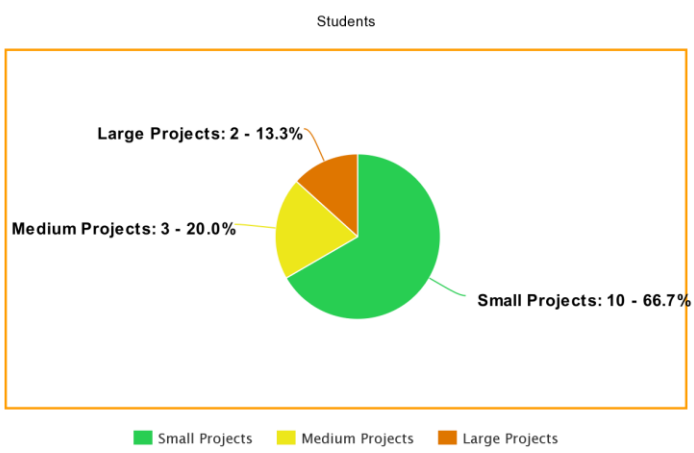
4.2. Questionnaires:

1 MQ: JavaFx developers

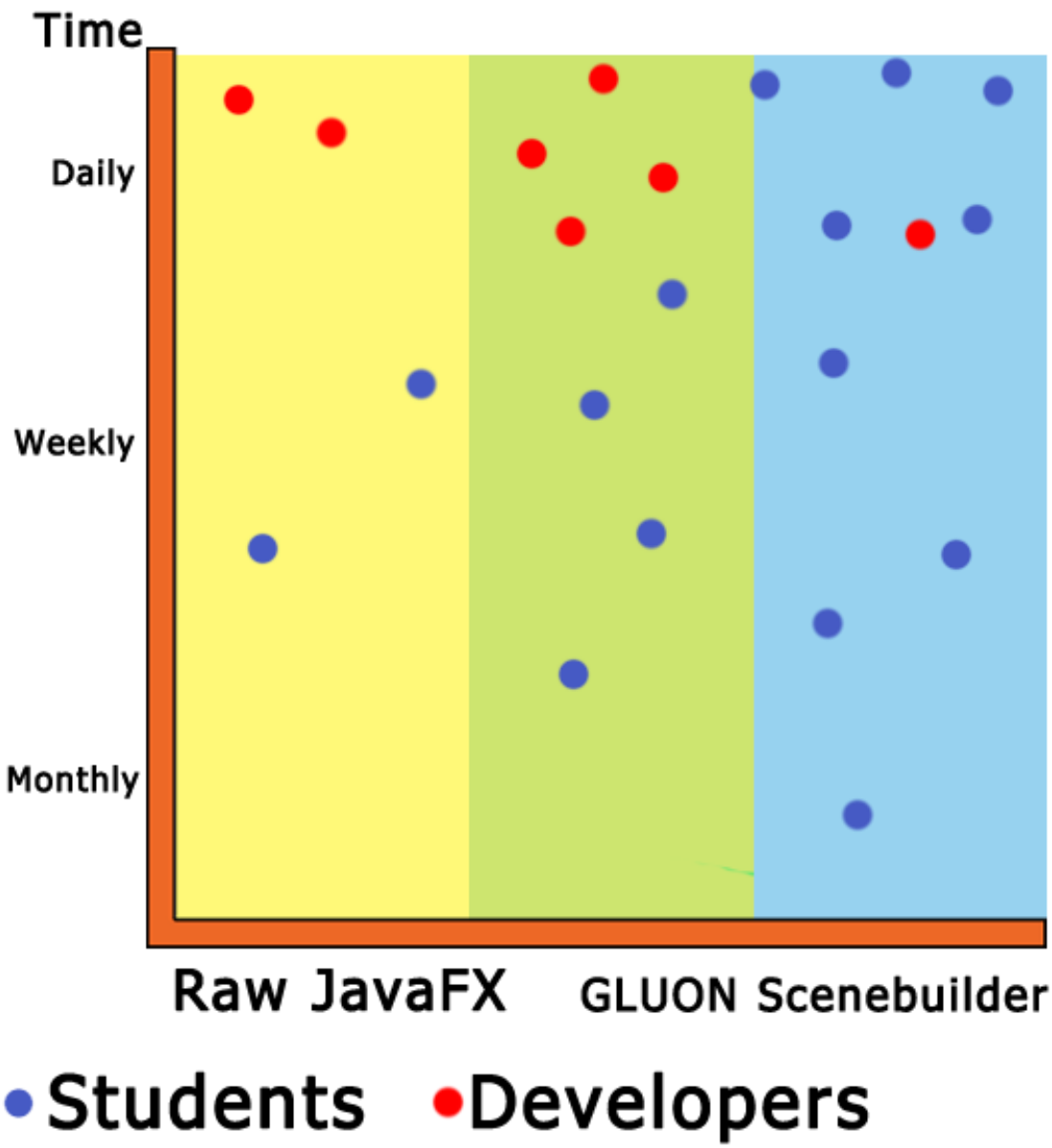
- **How often do you develop with JavaFx**
  - Once or twice a month
  - Weekly
  - 2-3 Times a week
  - Daily
- **Do you use any of the following (check all that apply)**
  - Raw JaxaFX
  - Oracle SceneBuilder
  - Gluon SceneBuilder
  - JFoenix Library
- **What scale of projects do you work on?**
  - 100-2000 Lines of code
  - 2000-10000 Lines of code
  - 10000+ Lines
- **Which of the following would you consider yourself**
  - Beginner
  - Student
  - Freelancer
  - Developer (Employed)

MQ was distributed on 22 FX developers of varying backgrounds for more diverse results and to aid comparison. The following is a compilation of these results:





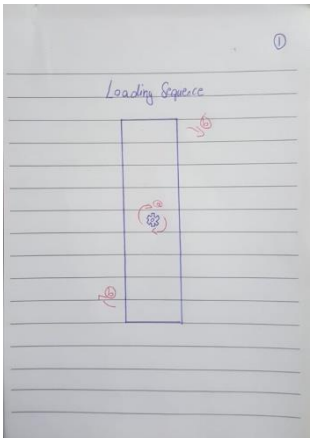
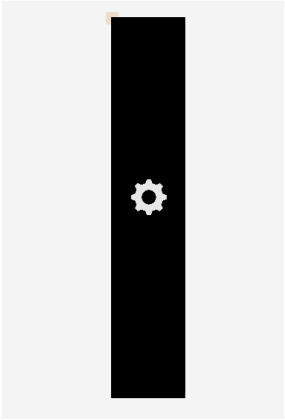
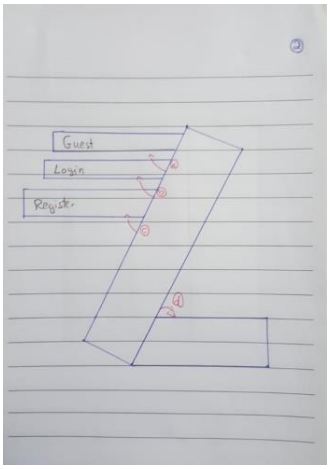
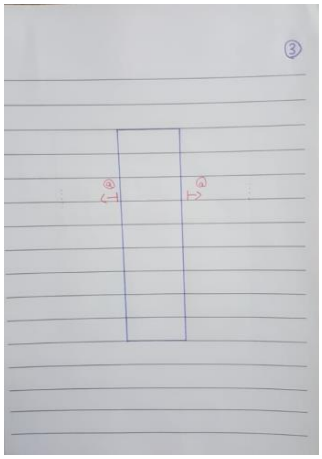
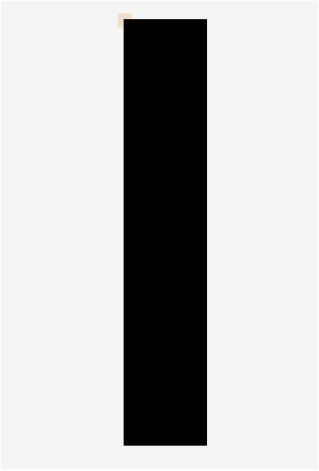
As expected, developers tend to work on larger projects whereas students take up smaller ones, the latter being in their learning phase. With this data in mind we move on to our second input evaluation:

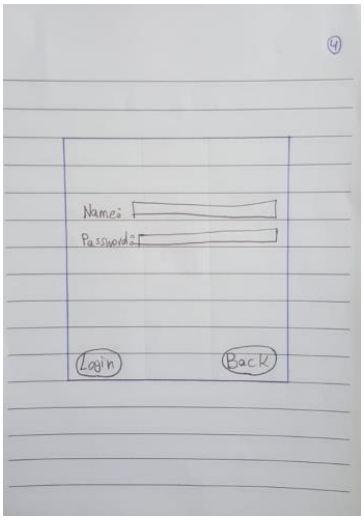


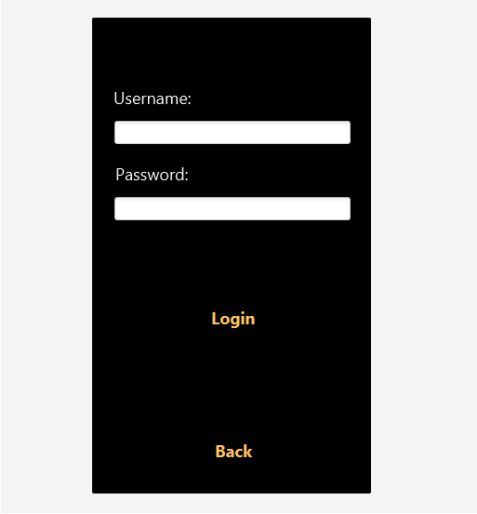
We thus deduce that larger projects require the developers to be working directly on the raw JavaFX code for optimization and flexibility purposes. Based on this assessment we chose to further push EasyFX's usability as a conversion tool.

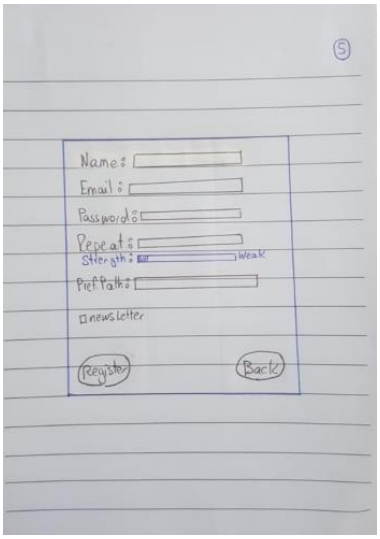
II.5. The Design Process

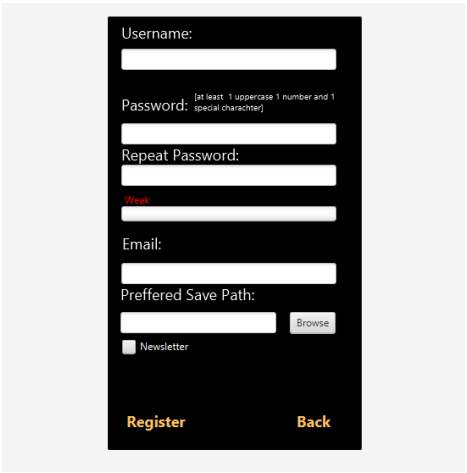
Table 6 Front End Sketches

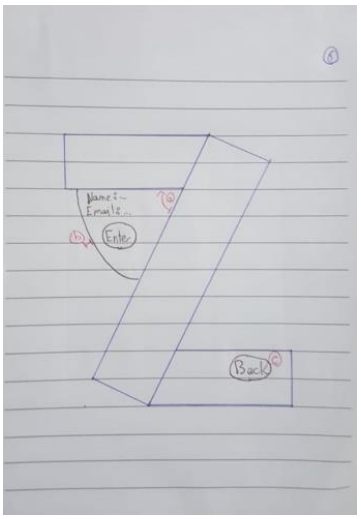
	
	
	


4
Name:
Password:
Login
Back


Username:
Password:
Login
Back


5
Name:
Email:
Password:
Repeat:
Strength:
Weak
Pref Path:
newsletter
Register
Back


Username:
Password:
[at least 1 uppercase 1 number and 1 special charachter]
Repeat Password:
Weak
Email:
Preffered Save Path:
Browse
Newsletter
Register
Back


6
Name:-
Email:-
Enter
Back


Name: Guest
Email: N/A
Back

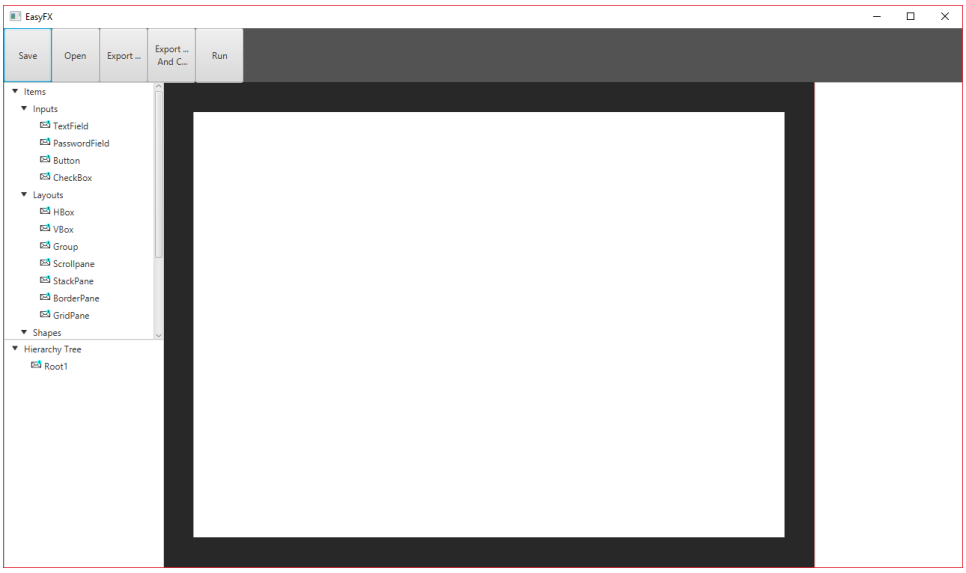


Figure 6 Builder Scene

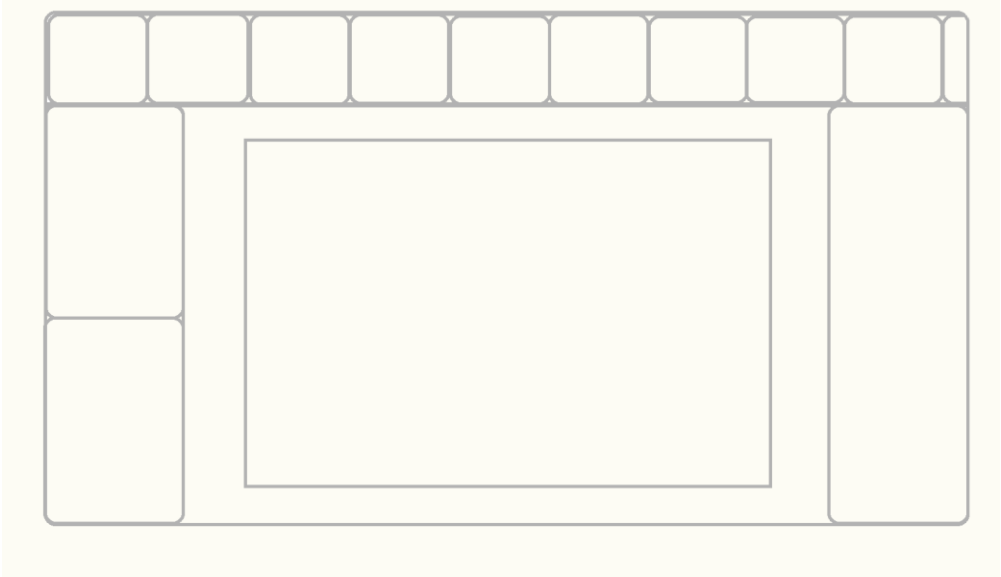


Figure 7 Builder Sketch

II.6. Conclusion

As is apparent from the sketches provide, we opted to map out the entire UI before any coding began. This allowed us to manage our time efficiently without having to worry about anything that might come up unexpectedly. In comparison, the final product is very much similar to the sketches due to our process of building the GUI onto a decal of the intended sketch. The final product came out exactly as was intended from the UI side which allowed us to instate our planned functionalities without much surprises.

Chapter III - Application Conception

III.1. Introduction

With the project planned out and a proper foundation set up, next come the steps to bring EasyFx to life. Below is the detailed description of our coding process and how we transformed a shower thought into a usable application.

III.2. UML Class Diagram

2.1. Front End & Login

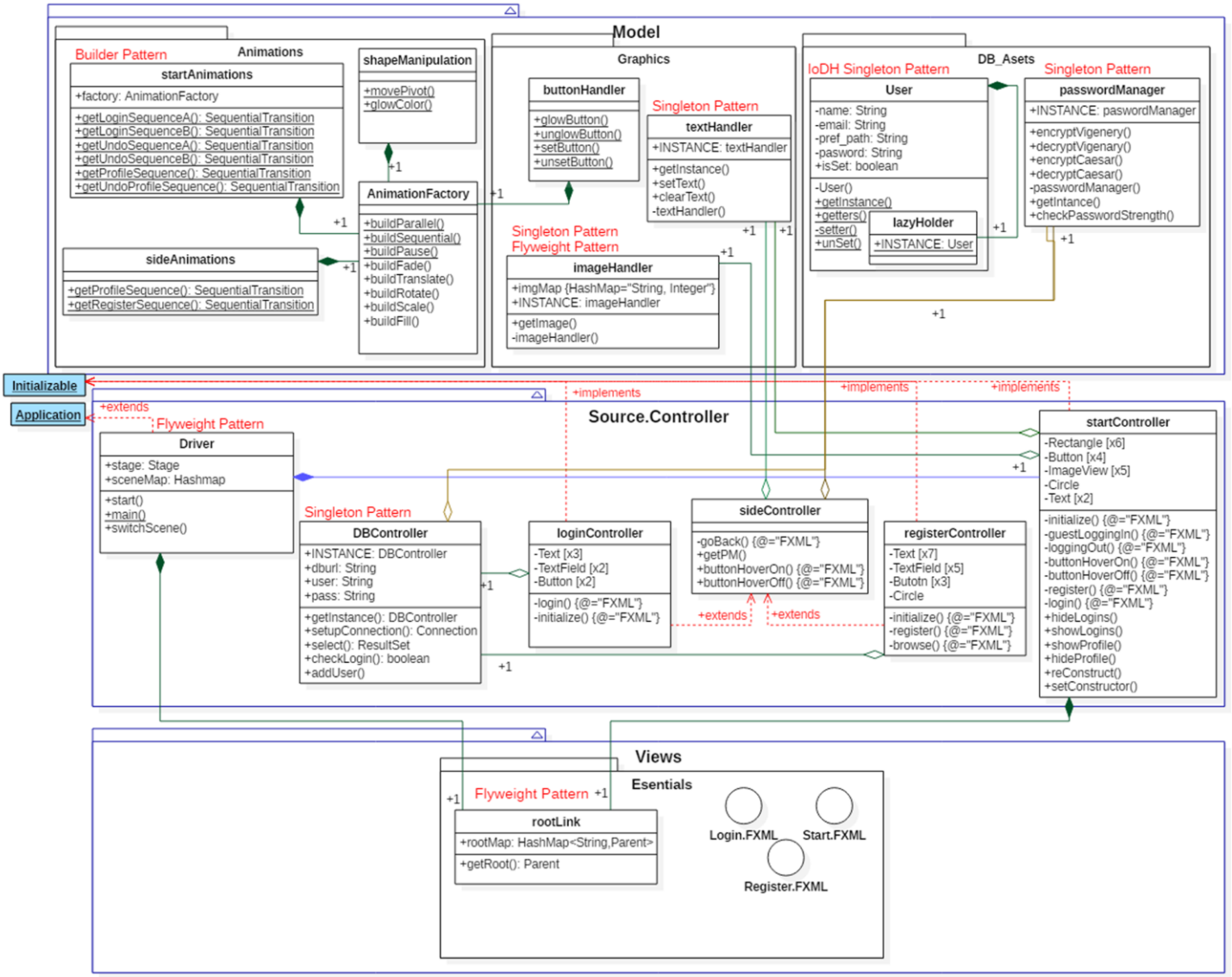


Figure 8 - Class Diagram Front End & Login Sequence

2.2. Back end & Builder

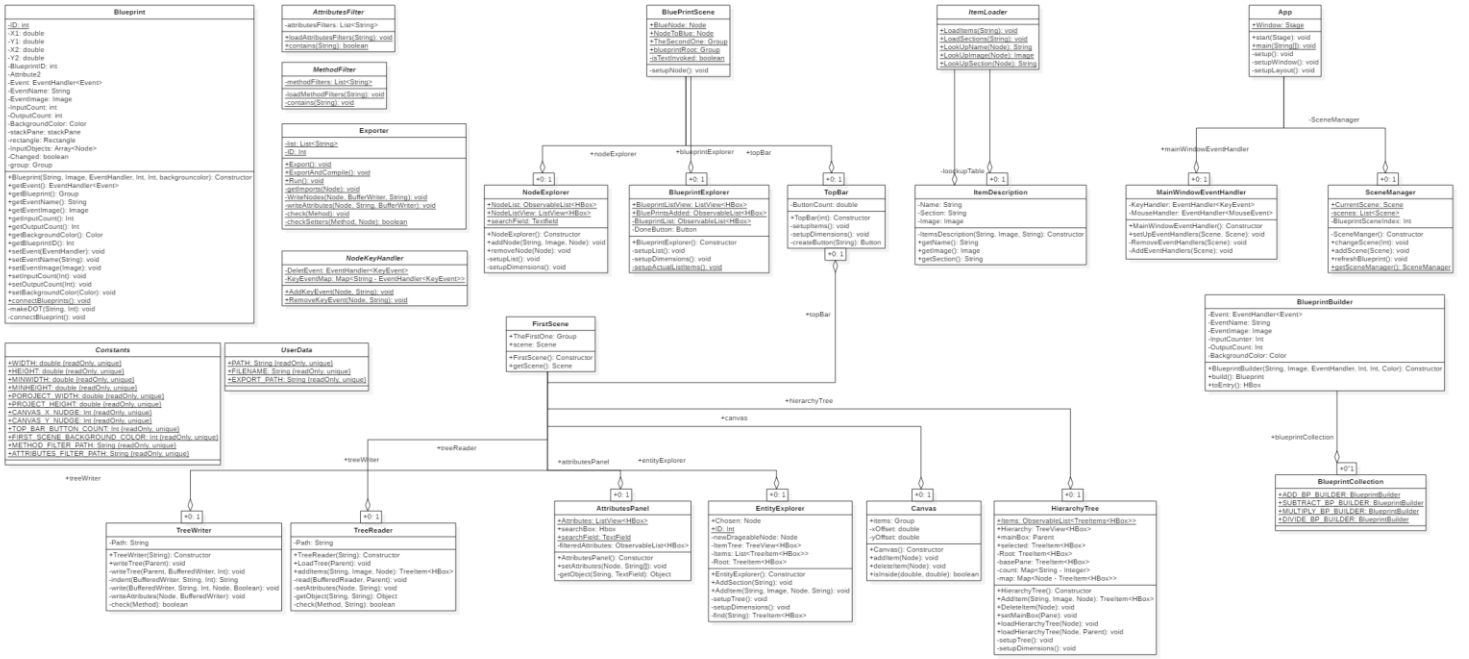


Figure 9 Back End Class Diagram

Our class diagram lies on the larger side when it comes to diagram sizes. We believe that we managed to clearly map out the entire code that needed to be written ahead of the development process in a way that provided the highest possible efficiency. Separating the front & back end prove to be a good call due to differences in representation when it comes to the two.

Front end programming is oriented towards how controllers manage the UI and thus is more accepting of design patterns, specifically the MVC pattern, which managed to cut our development time in half. On the other hand, the back end code is mostly one big Model with interloping functionalities that run depending on the user's input.

III.3. Sequence Diagrams

3.1. Initial access sequences

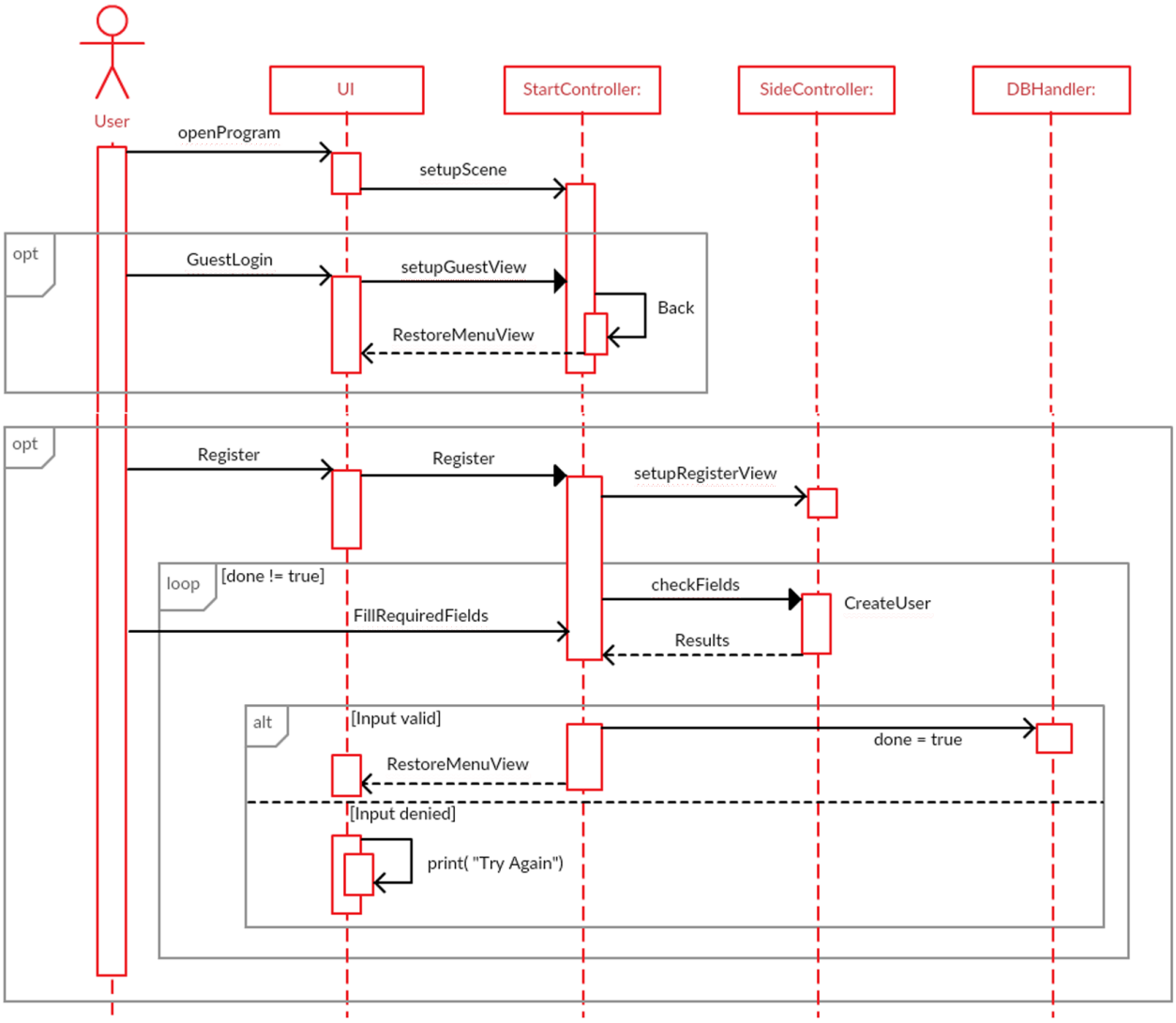


Figure 10 Initial Access Sequences

Figure 7 shows 2/3 of the three access sequences to reach the main builder screen. The standard 'Login' was omitted to limit redundancy due to its similarities to 'Register'. This is the first sequence initiated by default upon accessing the program. This is the only sequence that does not carry through into EasyFXLite which removes the database entirely.

III.4. Technical Design

4.1. Chosen 3rd Party Assets

- JPhoenix Library for smooth animations which cut designing time to half
- MS SQL database instead of files to accommodate a larger and more diverse dataset

4.2. Special Designs

- Using the Factory Pattern to set up animations coupled with the Builder Pattern to create each individual animation which provided flexibility in UI designs
- Adding “root Link”, a class that runs on the flyweight pattern to create and return FXML pages when requested
- Having the entire project on a giant MVC pattern that in itself holds up to 40 design patterns

4.3. Custom Assets

- Creating our own Markup Language (EZML) that runs similarly to FXML with flexibility to allow the use of Blueprints with a custom file extension and pseudo-language (BPE) that manage the user’s requests into files to be compiled into code or UI elements when needed.
- On the sidebar of functionalities, functions providing basic setter/getter functionalities have an input box next to them and event setters have a button that takes the user to the blueprint scene. In there, every function added through blueprint drag & drop will be linked as an event for that chosen event type and that chosen node. Blueprints themselves are predefined modules of code that can be written once and added forever in the blueprint explorer for use whenever the user wishes.
- The BP script can be customized or the user can opt for one of our available templates. We also aimed to make it easy for developers to create their own blueprint libraries to share with the community.

III.5. Maintenance and Delivery

Launch date: Monday, February 4th, 2019

After the initial launch, the program will run for a set time with no other releases except bi-weekly patches to fix reported bugs. This period is currently set as a month and is subject to extension if deemed necessary by the developers. After this period of debugging, subsequent updates will be done on a to-be-determined schedule, each update with its own period of debugging and patches of course.

Currently the plan is to implement the main update on EasyFXLite which will act as a Beta testing grounds. Per cycle the past update will be moved up to EasyFx and a new update will be pushed onto the beta version.

Chapter IV -**Application Test**

IV.1. Validation Test

- To determine that the requirements fairly represent what is needed, a condensed list of requirements was gone over with JavaFx developers to allow feedback and additional ideas
- Prototypes of the GUI were made nd gone over by a selected few testers
- EasyFXLite was made alongside EasyFx in an Agile fashion which served in itself as a constantly evolving prototype and test bench

IV.2. Unit Test

Login:

- Logging in or registering with some required fields empty
- Logging in with a false account/password combination
- Clicking buttons before the full scene animations have loaded it in
- Running multiple instances of the program
- Attempting SQL injections on the login scene

Builder:

- Attempting to place nodes outside the canvas
- Spamming all types of containers within each other
- Giving the wrong input type on data fields
- Attempting SQL injections on data fields

Blueprints:

- Making non logical connections that set the output into a node that can’t display the output properly
- Running a case without any BPs configured

IV.3. Benchmarks

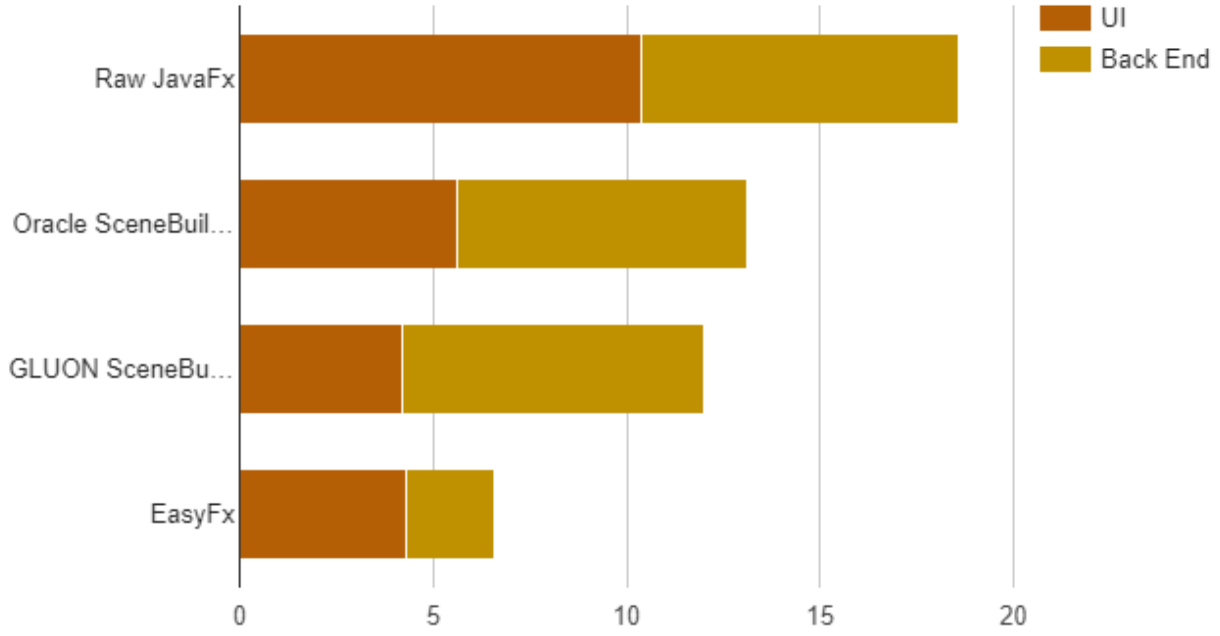


Figure 11 Benchmarks

Chapter V - Conclusion

Overall we would personally label our endeavor as a success as, though our goals and aspirations throughout the design process were evolving from making a library to an entirely new program, we managed to hit all our goals in record time.

V.1. Future Considerations

As we already implemented two versions of the program we are very much considering going forward with the maintenance schedule and releasing new versions. Our next approach will be an attempted extension onto GLUON's SceneBuilder that provides our advanced functionalities running on their already tested and proven base program. EasyFx began as a one-time project but is evolving into something special and we are proud to be the developers behind it.