<u>DO NOT TURN IN - THIS IS INCOMPLETE</u>

Design Document for MVP Project Planner

Minimum Viable Product

Cody Dukes, Nathan Grimsey, Maple Gunn

itsmobnt@gmail.com | ngrimsey@uw.edu | gunnra@uw.edu

https://github.com/TCSS360Group2Fall2023/Golden-Group-2-Repository

School of Engineering and Technology, University of Washington Tacoma

TCSS360: Software Development And Quality Assurance Techniques

Professor Jeffrey Weiss

November 19, 2023

Contents

Introduction

This document outlines the design plans we have for making our project planning app. It contains our design pattern rationale, class diagrams, and user story design diagrams. The design heuristics we will be adhering to are listed, as well as the way they are integrated or considered. There are also annotations for the different shown diagrams to give further insight into how our design is meant to function.

Rationale Summary

MPP is meant to keep track of tools, budgets, and individual projects. MPP will provide functionality to keep track of an inventory of tools that can be added to a project, as well as expenses associated with a project. There will be a simple User Interface (UI) to store all the materials needed for a project and the overall cost of buying those materials. Projects can be exported to other users which will include the tools used, expenses, schedules, logs, and any other relevant information or files stored within a project.

We tried to keep our different pages as different classes, each mostly acting as interfaces to access functions/objects. For example, aboutScreen accesses the profile and dev objects rather than having them be hardcoded, in cases where we'd want to access those objects elsewhere (such as profile being used potentially in exporting data) this aligns with heuristic 5 since the interface is reliant on the model rather than vice-versa. Objects like Materials and Tools have relationships with several different classes as they can be accessed from many different screens and have very distinct functions/responsibilities. This adheres to Heuristics 1 and 2, as it keeps data horizontal and avoids having a god class that stores all the project data in itself.

The Heuristics by which we are building our design around:

Heuristic 3.1: Distribute system intelligence horizontally as uniformly as possible, that is, the top-level classes in a design should share the work uniformly.

Heuristic 3.2: Do not create god classes/objects in your system. Be very suspicious of a class whose name contains: **Driver**, **Manager**, **System**, or **Subsystem**.

Heuristic 3.3: Beware of classes that have many accessor methods defined in their public interface. Having many implies that related data and behavior are not being kept in one place.

Heuristic 3.4: Beware of classes that have too much noncommunicating behavior, that is, methods that operate on a proper subset of the data members of a class. God classes often exhibit much noncommunicating behavior.

Heuristic 3.5: In an application that consists of an object-oriented model interacting with a user interface, the model should never be dependent on the interface. The interface should be dependent on the model.

Heuristic 3.6: Model the real world whenever possible. (This heuristic is often violated for reasons of system intelligence distribution, avoidance of god classes, and the keeping of related data and behavior in one place.)

Heuristic 3.7: Eliminate irrelevant classes from your design.

Heuristic 3.8: Eliminate classes that are outside the system.

Heuristic 3.9: Do not turn an operation into a class. Be suspicious of any class whose name is a verb or is derived from a verb.

Heuristic 3.10: Agent classes are often placed in the analysis model of an application. During design time, many agents are found to be irrelevant and should be removed.

Heuristic 1: We follow this by having a clear delineation of interfaces versus model.

Heuristic 2: We follow this by separating responsibilities between the classes.

Heuristic 3: We follow this by not having too many accessors in our public interface.

Heuristic 4: We follow this to the extent that we don't have a god class, but constructor classes are going to inherently not be very communicative.

Heuristic 5: Our models are not dependent on the interface.

Heuristic 6: Our models are intuitive.

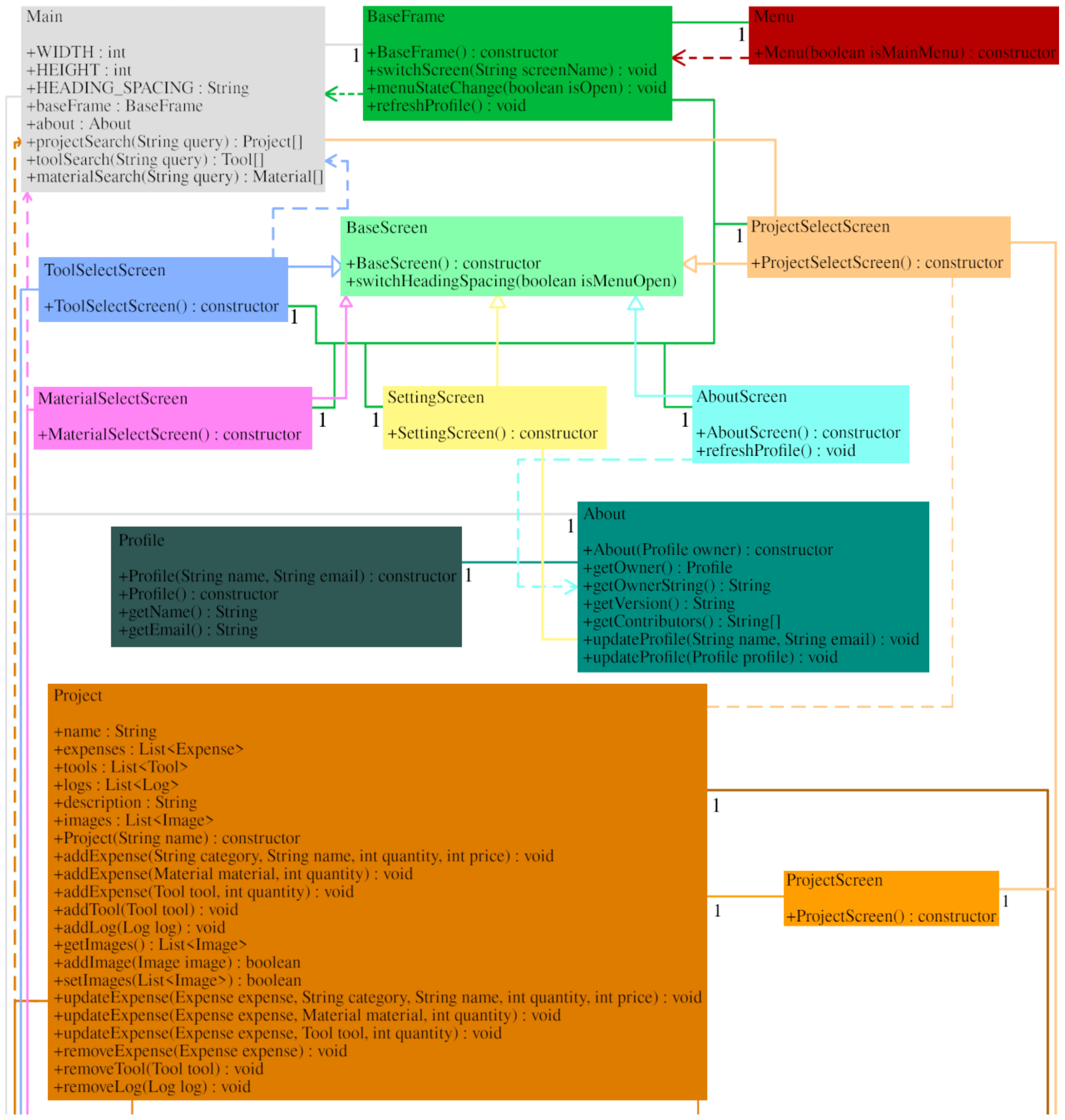
Heuristic 7: We don't have irrelevant classes.

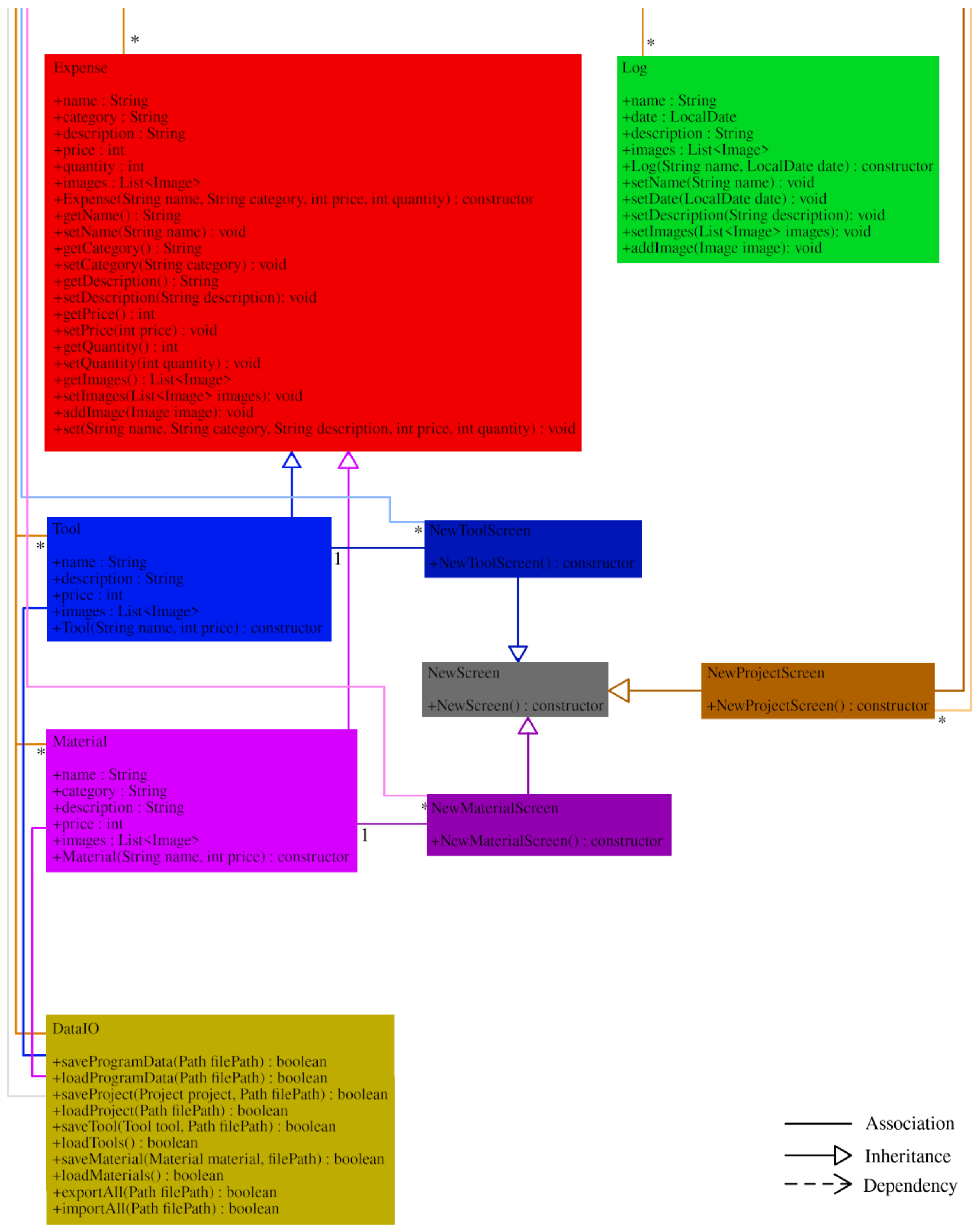
Heuristic 8: There are no classes outside the system.

Heuristic 9: We have an about object which sort of is an operation class, but if we didn't it'd be a part of our interface so it was necessary.

Heuristic 10: There are no agent classes.

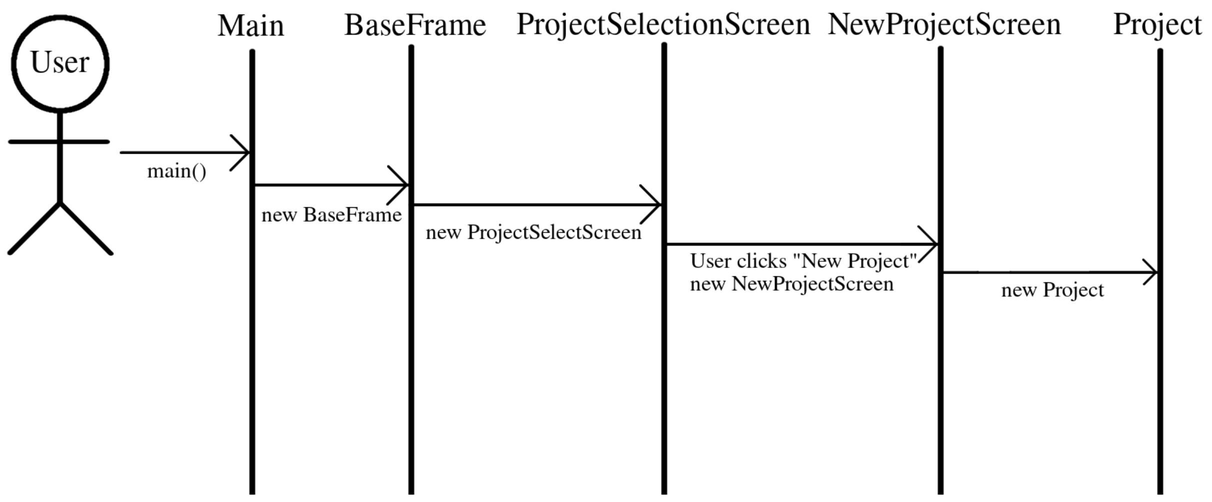
Class Diagrams


Main
+WIDTH : int
+HEIGHT : int
+HEADING_SPACING : String
+baseFrame : BaseFrame
+about : About
+projectSearch(String query) : Project[]
+toolSearch(String query) : Tool[]
+materialSearch(String query) : Material[]
BaseFrame
+BaseFrame() : constructor
+switchScreen(String screenName) : void
+menuStateChange(boolean isOpen) : void
+refreshProfile() : void
Menu
+Menu(boolean isMainMenu) : constructor
BaseScreen
+BaseScreen() : constructor
+switchHeadingSpacing(boolean isMenuOpen)
ToolSelectScreen
+ToolSelectScreen() : constructor
ProjectSelectScreen
+ProjectSelectScreen() : constructor
MaterialSelectScreen
+MaterialSelectScreen() : constructor
SettingScreen
+SettingScreen() : constructor
AboutScreen
+AboutScreen() : constructor
+refreshProfile() : void
Profile
+Profile(String name, String email) : constructor
+Profile() : constructor
+getName() : String
+getEmail() : String
About
+About(Profile owner) : constructor
+getOwner() : Profile
+getOwnerString() : String
+getVersion() : String
+getContributors() : String[]
+updateProfile(String name, String email) : void
+updateProfile(Profile profile) : void
Project
+name : String
+expenses : List<Expense>
+tools : List<Tool>
+logs : List<Log>
+description : String
+images : List<Image>
+Project(String name) : constructor
+addExpense(String category, String name, int quantity, int price) : void
+addExpense(Material material, int quantity) : void
+addExpense(Tool tool, int quantity) : void
+addTool(Tool tool) : void
+addLog(Log log) : void
+getImages() : List<Image>
+addImage(Image image) : boolean
+setImages(List<Image>) : boolean
+updateExpense(Expense expense, String category, String name, int quantity, int price) : void
+updateExpense(Expense expense, Material material, int quantity) : void
+updateExpense(Expense expense, Tool tool, int quantity) : void
+removeExpense(Expense expense) : void
+removeTool(Tool tool) : void
+removeLog(Log log) : void
ProjectScreen
+ProjectScreen() : constructor
1
1
1
1
1
1
1
1
1
1
1
1
1


*
Expense
+name : String
+category : String
+description : String
+price : int
+quantity : int
+images : List<Image>
+Expense(String name, String category, int price, int quantity) : constructor
+getName() : String
+setName(String name) : void
+getCategory() : String
+setCategory(String category) : void
+getDescription() : String
+setDescription(String description): void
+getPrice() : int
+setPrice(int price) : void
+getQuantity() : int
+setQuantity(int quantity) : void
+getImages() : List<Image>
+setImages(List<Image> images): void
+addImage(Image image): void
+set(String name, String category, String description, int price, int quantity) : void
*
Log
+name : String
+date : LocalDate
+description : String
+images : List<Image>
+Log(String name, LocalDate date) : constructor
+setName(String name) : void
+setDate(LocalDate date) : void
+setDescription(String description): void
+setImages(List<Image> images): void
+addImage(Image image): void
*
Tool
+name : String
+description : String
+price : int
+images : List<Image>
+Tool(String name, int price) : constructor
1
*
NewToolScreen
+NewToolScreen() : constructor
NewScreen
+NewScreen() : constructor
NewProjectScreen
+NewProjectScreen() : constructor
*
*
Material
+name : String
+category : String
+description : String
+price : int
+images : List<Image>
+Material(String name, int price) : constructor
1
*
NewMaterialScreen
+NewMaterialScreen() : constructor
DataIO
+saveProgramData(Path filePath) : boolean
+loadProgramData(Path filePath) : boolean
+saveProject(Project project, Path filePath) : boolean
+loadProject(Path filePath) : boolean
+saveTool(Tool tool, Path filePath) : boolean
+loadTools() : boolean
+saveMaterial(Material material, filePath) : boolean
+loadMaterials() : boolean
+exportAll(Path filePath) : boolean
+importAll(Path filePath) : boolean
Association
Inheritance
Dependency

User Story Sequence Diagrams

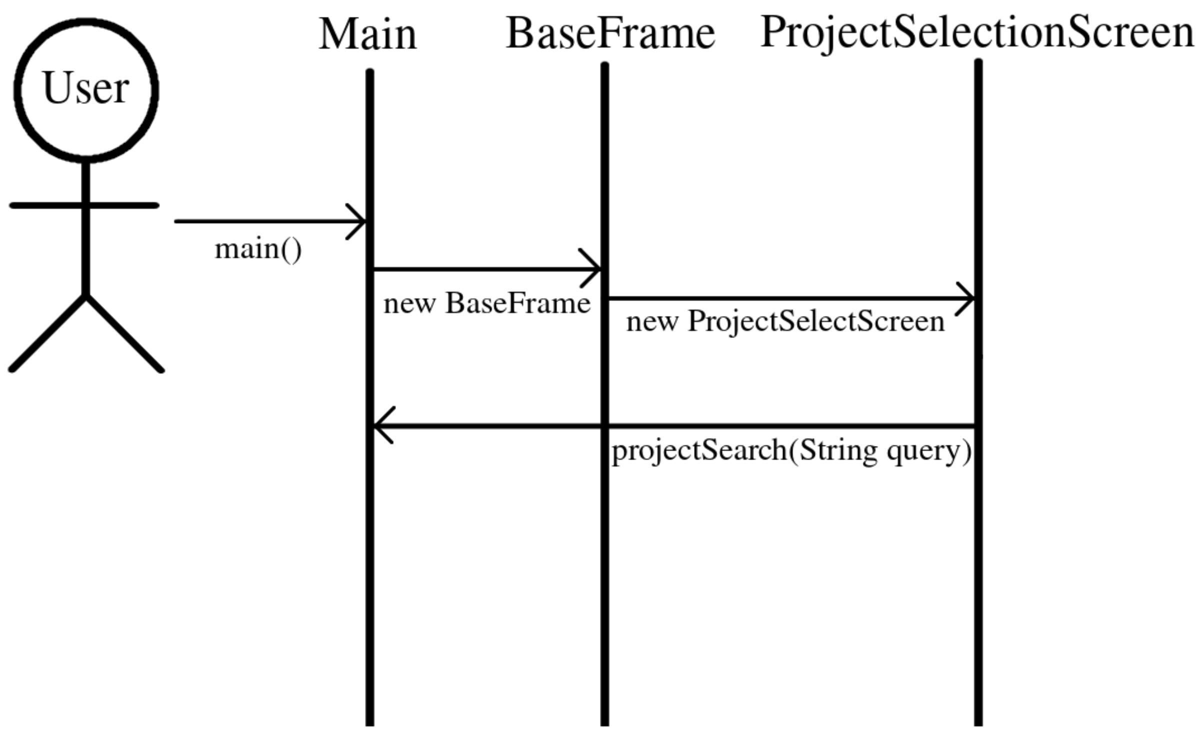
US01 - As a DIY enthusiast, I'd like to start a project that contains my budget, things to buy, and a way to log my progress.



Description of the computation:

The application is first opened by the user. Once open, the user can open the project screen, where they will have the option available to create a new project. Doing so will open a window where the user can enter details about the new project, such as a title and picture. Once satisfied, the user can confirm the creation of a new project, and the project will be saved within the application.

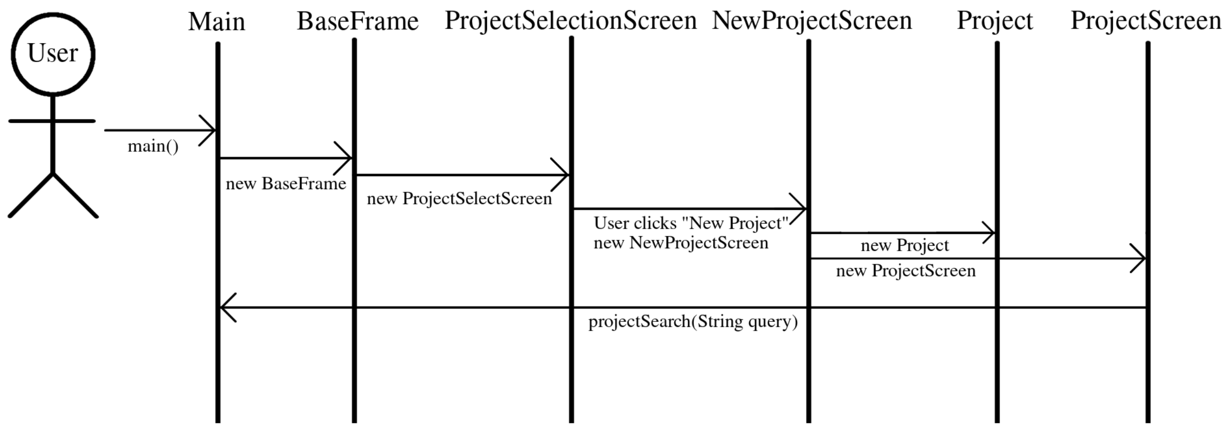
US02 - As a project manager with a lot of projects, I'd like the ability to use keywords in a search instead of having to memorize entire folder names.



Description of the computation:

The user opens the application and goes to the project screen. They then click on the search functions text field. They can then type a keyword into the field, and either press enter or the 'search' button, which will prompt the program to pass the text to main and perform a keyword search. The program will then display a list of projects that match the searched keywords to the user, of which they can directly open any project by clicking on it.

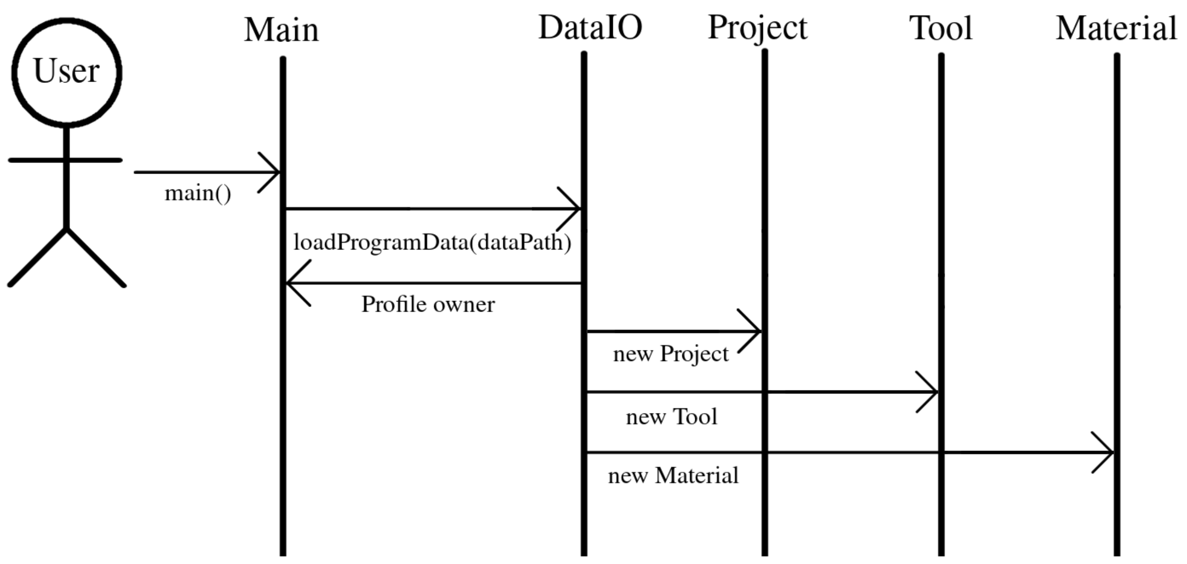
US05 - As a DIY enthusiast, I want to be able to reference a previous project in a new project so I can use floor plans and material estimates in my new projects.



Description of the computation:

User opens the application, opens the sidebar menu, selects the tool option which directs them to the tool screen, and then uses the addTool() function to add a tool to their directory.

System Startup Sequence Diagram



Our app has a simple startup that will retrieve data externally from the export class. It will take this data and then populate the projects, logs, materials, and tools.