Project Milestone 1.2

David Cooper

High-Level Design

The only architectural pattern which makes sense for this project is MVC. I will discuss each other pattern and why it isn't applicable.

The pipe and filter pattern makes the most sense when there are a lot of different things which can be done with the data. For instance, if you are working at Youtube, there is a large amount of user data coming in. This user data is then processed in tons of different ways to achieve different goals. This project involves very little data. The way it works is that the instructor alone adds data to the system and the students are users of the data without any ability to manipulate permanent data. If the system were to collect usage data, then a series of filters may make sense, but since that is out of scope the design pattern is not applicable.

For event based and microservice architectures, these are mostly applicable to real time applications. If a user makes a request, it is handled by a system using the architectural pattern. Because this project does not involve real time data processing, these patterns are also not applicable.

For layered patterns, MVC is a pretty sensible way to create a web application, which this project is. As mentioned before, the students do not manipulate data. Because of this, MVC doesn't make sense for the workflows involving students. The workflows used by instructors involve viewing and manipulating data, which lends itself to MVC.

Low-Level Design

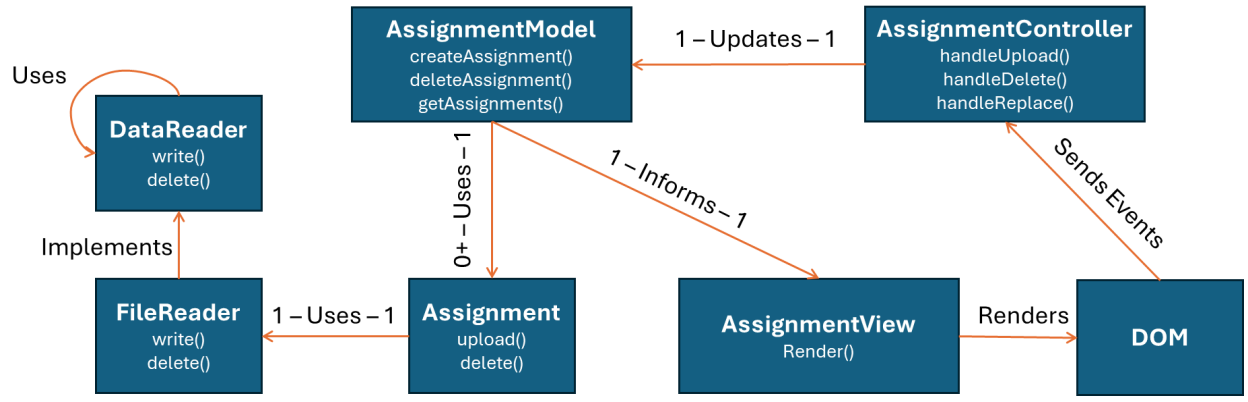
Singleton is a very useful design pattern for anything involving data access. A database connection, a file manipulator, and an event logger are all situations which are good uses of this pattern since usually only one of these needs to exist to manipulate physical data. For this project, this will make sense for a file editor.

As discussed with the last guest lecturer, it makes a lot of sense for your file access to be decoupled from a specific class. As such, an interface will be used for the file access class to implement. This way, it will be easy to switch to a database or other form of reading physical data.

Besides that, we will implement a really basic MVC. Because it is web based, the DOM will serve as an additional layer of the view. Therefore, the class definitions will be:

```
Interface DataReader:
      Private DataReader()
      Public static getDataReader()
      Public write()
      Public delete()

Class FileReader implements DataReader:
      Private FileReader dataReader
      Private FileReader()
      Public static getDataReader()
      Public write()
      Public delete()

Class Assignment:
      Private DataReader file
      Public Assignment(DataReader, sourceFile, destFile, assignmentName)
      Public delete()
      Public upload()

Class AssignmentModel:
      Private Collection<Assignment> assignments
      Private AssignmentView view
      Public AssignmentModel()
      Public createAssignment(sourceFile, assignmentName)
      Public deleteAssignment(assignmentName)
      Public Collection<Assignment> getAssignments()

Class AssignmentView:
      Private AssignmentController controller
      Public AssignmentView()
      Private render()

Class AssignmentController:
      Private AssignmentModel model
      Public AssignmentController()
      Public handleUpload(file, assignmentName)
      Public handleReplace(file, assignmentName)
      Public handleDelete(assignmentName)
```

The class diagram will look as follows:



The above handles the instructor facing tasks, which is basically just a file manager. The assignments themselves are webpages which are specific to the assignment, so they can't be designed in the same way.

Prototype

Here is a simple wireframe of the above instructor facing file manager. It is very boilerplate and I don't feel there is much useful feedback to be had from it. Instead, I chose to create an example assignment and do usability testing on that. That is attached as a zip file in Canvas.

Assignment Name		
Assignment 1	⚙	🗑
Assignment 2	⚙	🗑
Assignment 3	⚙	🗑

New Assignment

For the above prototype, I chose to do it as a very simple file manager that you might see anywhere. Choosing to use icons instead of text is something which may inhibit usability for users who use screen readers, but alt text should be able to help with that. However, the icons are a cleaner design and should be obvious to most users. Management would take place in an extended view to limit screen clutter.

For the example assignment, the html for this was built by exporting a google doc. A lot of the formatting is strange as a result, but could be improved. Besides that, there are simple inputs for testing the attackers and defenders with buttons to submit. Because it can be computationally intensive, having submit buttons is needed. For the code input, allowing for custom code is a cool idea, but may be challenging to use. Alternatively, text descriptions of the logic used could

replace the code examples. Because I think custom input is cool, text descriptions wouldn't work. For usability concerns, having an image output absolutely does not work with screen readers regardless of if alt text is used.

Feedback

Task description: The task you are given is to try to understand the assignment and try out the interactive elements (they have no functionality though).

The assignment was interesting to understand, although a little challenging at first. Below is my feedback.

1. The assignment is a little vague to understand as it is right now
2. I didn't like the fact that I had to go to a external website to just understand the problem
3. The interactive part - 'Try it Yourself', the assignment doesn't really mention what this does. This may not be a good design choice. A small prompt above the interactive parts should be helpful.
4. I would also recommend UI enhancements.
5. Since this is a interactive assignment, a small prompt about how the assignment works as a whole would be helpful too.

Hi David,

I went through the demo and tested the interactive elements. Here's concise feedback:

- The assignment is clearly structured and logically progresses from algorithm design to implementation and visualization.
- The Risk explanation and probability table make the problem understandable.
- The interactive inputs are intuitive, but the outputs are clearly placeholders. It may help to label them as mock results or add minimal validation (e.g., prevent negative inputs).
- The plot shown after submission appears static.

Overall, the demo communicates expectations well. With minor UI and structural refinements, it would feel more polished.