Software Engineering Applications
ESOF 423 - Spring 2022

Patrick O’Connor
Marnie Manning
Silas Almgren

Section 1: Program

All of the source code for this project can be found on our Github.

Specifications for our tech stack:

Framework: .NET 5

Languages: C#/JS/HTML & CSS

Hosting: Heroku Free Tier - 1 web dyno (container), 1 worker dyno, 512 MB per dyno

Libraries: XUnit (for testing), OpenXML

Section 2: Teamwork

Member 1

Over the course of the semester I have taken a good mix of both Front end and back end. With the assistance of Member 3, a solid representation of the validation results now exists and is automated in that a new warning or error could be added without any implementation changes in back end logic for displaying data. Furthermore as others have done I have tackled a couple of the actual validation of requirements in submitted dissertations.

Member 2

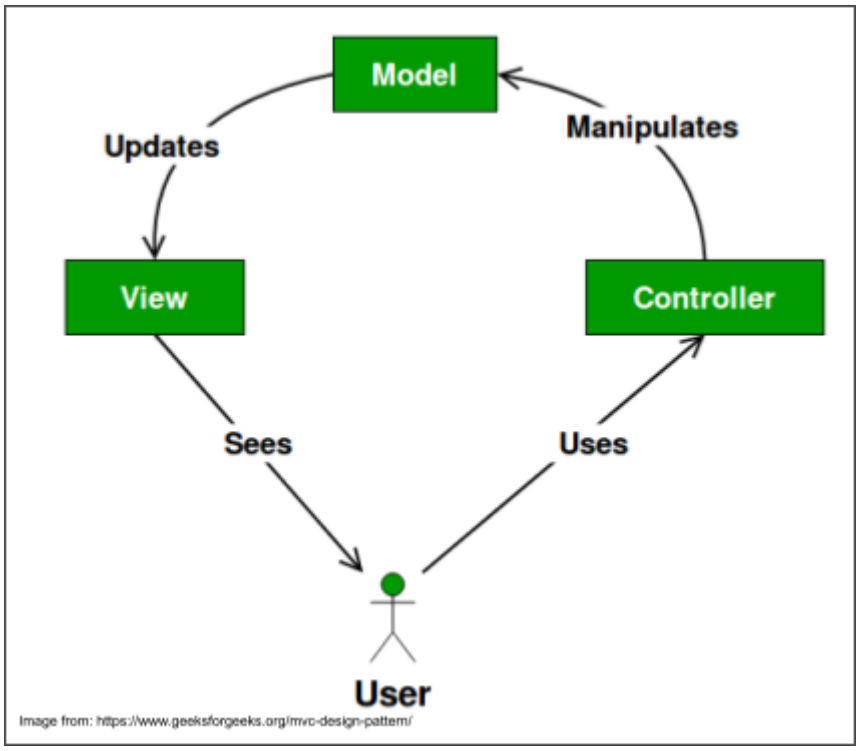
Most of what I worked on has been back end functionality for parsing and measuring the qualities of a given document submitted by users. Specifically I have worked on parsing table and figure information such as table/figure location, spacing, and other formatting requirements required by figures. I also worked to help put together and maintain some of the artifacts for this project like the trello board and burn down documents.

Member 3

For the most part, I tried to tackle the DevOps side of things. Initially, this involved getting an empty .NET Core project set up in the github, configuring the CI/CD to automate building and testing the app, and getting it up on Heroku. From there, the bulk of my work was setting up the overarching structure of the app. This entailed setting up the controllers, view models, creating the structure to get a word document into the backend, and building some generic structure for where the validation code can be written. Overall, the aim of this work was to set up a good structure to allow the other members to write good code without worrying about framework specific details. Of course, like all of us, I also built a couple of the validation components.

Section 3: Design pattern.

One design pattern that was used was the Model-View-Controller (MVC) pattern. This design pattern can implicitly be seen in the structure of our web application. By examining the source code in our Github repository, it can be noted that there are individual directories for each component of this design pattern (Models, Views, and Controllers). In this structure, the controllers dictate the routing to be used by the web application. The models are used to program the logic of the application, as well as pass data to the views. With the specific framework used, .NET Core, the views are rendered into static HTML which is loaded for the user on the client side. A diagram of this structure can be seen below.



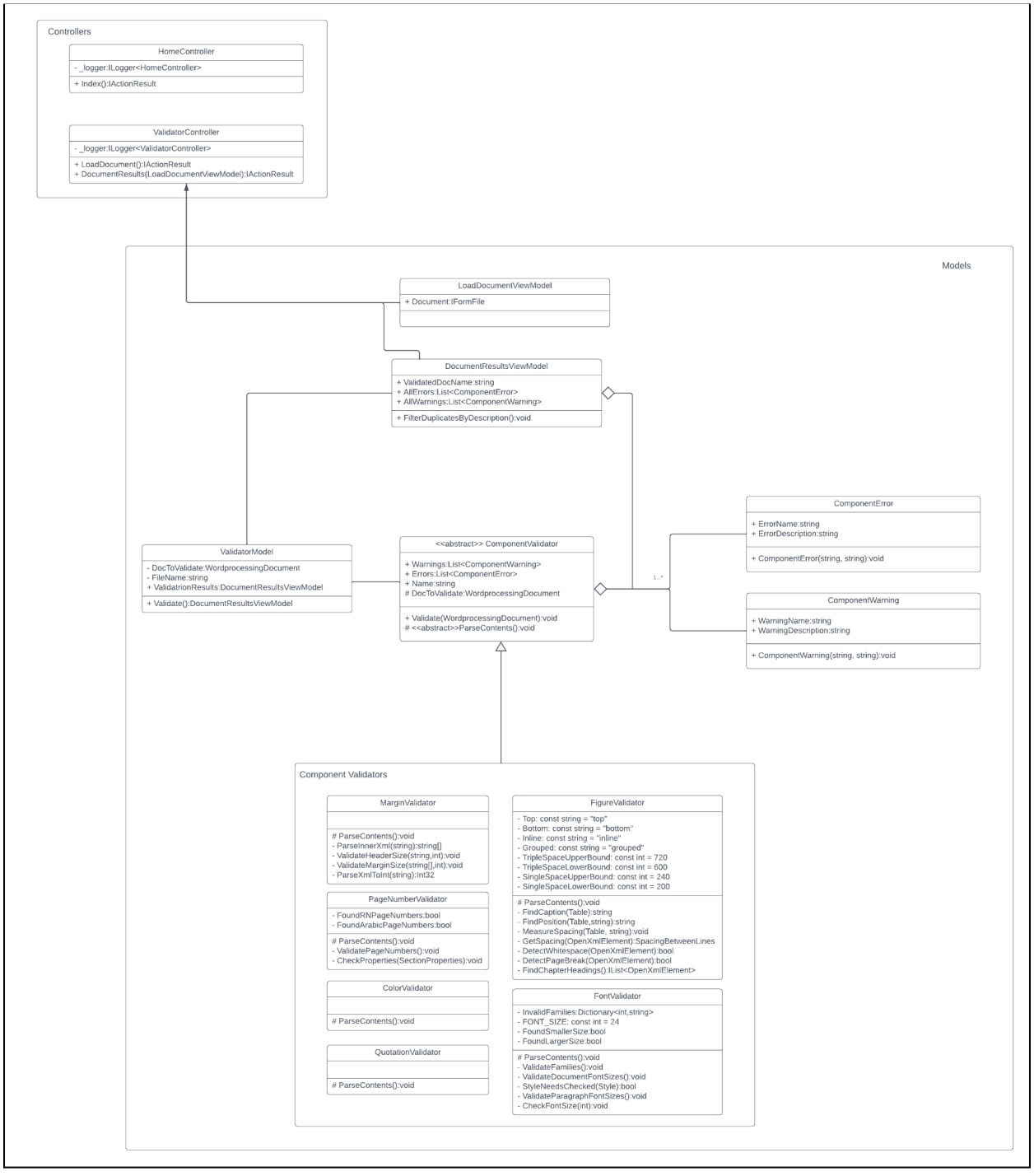
This design pattern was chosen for a couple of reasons. One main factor was simply that MVC is the default structure for a .NET Core web application. Since we chose early on to use the C#-based OpenXML library, .NET was our best choice for a web framework. Additionally, the MVC pattern allows for good separation between the different elements of the application. For example, new formatting rules can be easily added to the backend models without needing to edit any functionality on the frontend views. In addition to editing existing functionality, the concept of controllers makes it easy to add new features that are clearly separated from the ones already in place. Overall, this design pattern helps to make the application reusable and extendable.

Section 4: Technical writing

All of our documentation can be found through the README on our github repository.

Section 5: UML

A high resolution version of our UML class diagram can be found on the github repository.



Section 6: Design trade-offs.

One design tradeoff that had to be made in the development process was where to host our application. Cloud development is currently an extremely lucrative field, which means there are effectively endless options for hosting services. After initial deliberations, our choices were narrowed down to three options: DigitalOcean, Heroku, and simply hosting on a school server. We ended up hosting on Heroku, but it's worth discussing why we didn't go with the other options. DigitalOcean is an incredibly popular hosting service that allows you to configure web servers on "droplets", which are effectively web-facing Virtual Machines (VMs). Essentially, getting a droplet allows you to access a Linux server of your choice where you can configure your application. In turn, setting up our web application on DigitalOcean would have been pretty similar to using a school server. For a .NET app, this would involve configuring a web server like Apache and setting up the .NET app to be served over a reverse proxy on this web server. For the most part, this isn't a terrible process, but it would have taken a fair amount of time and would've been harder to maintain in the long run. Instead of taking this more involved approach, we used the "one-push" hosting service, Heroku. Hosting on this service simply involves creating an application within their website and pushing your application from a command line. When pushing, Heroku automatically configures the VM your application is served from and figures out all the details needed to host your chosen framework. One caveat with Heroku is that your VM is only live when the application is being used. If the site is idle for long enough, it will effectively power down. As a result, the first user to access the application after it's been idle can expect a 5-10 second wait time while the VM spins back up. Another consideration we had to take with this service is that it doesn't have out-of-the-box support for .NET applications. This required us to containerize our application with Docker. Fortunately, Docker is very simple to use, and setting it up took much less time than configuring a web server would have. Ultimately, Heroku proved to be the best hosting solution. While it provided less customizability and slower speeds at start time, it is much less complex to use, was efficient enough for our usage, and will offer much better maintainability.

Section 7: Software development life cycle model.

For our project, we used a modified Agile Scrum framework to develop the capstone project. Our Scrum framework consisted of two-week-long sprints with planning meetings at the beginning of each sprint. During a planning meeting, we would outline what was in our backlog for a given sprint and assign a story point value for each item in that backlog. Once we had story points assigned we would assign the different action items to different team members. The assigned action items were stored on a group Trello board we could refer back to throughout the sprint. During our two-week sprint, we would regularly have standup meetings on Wednesday

and Friday with occasional meetings on Monday. Throughout the sprint, we would record the story points completed and the hours spent on a burndown chart that tracked the progress of the entire project.

This method worked well for us as it promoted high communication and reflection. Choosing to use story points with agile helped keep our project in perspective as we worked. By the last two sprints, we had a good idea of how long a given action item might take given its story point value and what tasks might need to be broken down into two different action items. Additionally, choosing to use story points instead of hours in our planning helped keep morale high, there was less stress when an action item took longer than we expected. Our frequent standup meetings kept us in communication, and when we had problems we could easily help each other or brainstorm.

While this method generally worked well for this project there were a few aspects that could have been better suited to our project. Due to the life cycle model being predetermined by the class the time frame of the sprints did not line up with where we were actually at in development. The beta release ended up being too soon and by the time the release candidate was due we were already testing it for bugs. Being able to choose the length of our sprints, or choose which sprint would have which deliverable might have been able to help prevent the discrepancy between the deliverables and sprints.

Additionally, having our client more integrated into the review phase of the sprint could have helped us gain a better understanding of what our client wanted early on and how this project would fit into the future of the product. The structure of a sprint presented in this class did not involve clients in the review process. We met with our client towards the latter half of the planned six sprints and this meeting changed the focus of our development from a broader to a more focused scope for the end product. Perhaps if our clients were more involved we would have been able to spend more energy on the aspects of the product they valued the most.