**Evaluation of the finished product, regarding the Keller-Williams Family Check-In**

--Patrick

**i. Assessment of the completeness and correctness of your system**

**1. Is the software done (and how do you know what “done” means)?**

Yes, a full version 1 is complete. Our initial feature goal, what we call 'version 1 features', has been fully implemented. In addition, we have implemented many of the 'version 2 features' that we thought would need to be integrated after our 10 week schedule. We are ahead of where we thought we would be during our initial assessment of progress and would feel comfortable selling this product to our client in its current state.

**2. For web developers: Does it do what the client wants it to do?**

Yes, our clients are exceptionally pleased with our work. A detailed email feedback, provided by Thomas Dye our lead consultant at Keller-Williams, can accurately display how pleased they are with what we've provided.

--Aaron

**ii. Assessment of the quality of your design**

**1. Were there any features that were especially hard to implement because of the design?**

Entity Framework Core has some restrictions on handling one to one relationships so we had to come up with some creative solutions for handling this issue. Our solution was to utilize SQL commands, but this has implications if the model state ever drastically changes. This is explained by comments in the code in case it becomes an issue for any future developers.

**2. Is this design scalable (will it be easy to add features)?**

Yes, we specifically designed our program to be scalable at our clients request. Their hope is that this program is able to be utilized at other Keller-Williams office locations in the near future. From the beginning Keller-Williams has explained to us their vision of the system supporting 300 or more brokers and our system can currently accommodate their request.

**3. Is the design easy to understand and communicate to others?**

The architecture would be easy to understand and communicate to other programmers familiar with C#. Utilizing MVC has kept an easy separation of concerns between the various parts of our code. Specific parts of the controller methods and JavaScript may be less intuitive and require explanation, but we have added comments to provide clarity. Additional frameworks which have been added to our system have online documentation readily available.

**4. Does this design result in sufficient performance (will the web site or game be responsive in the more challenging use cases, such as when it is run an older/slower machine, when there are many simultaneous users, when it is performing computationally intense operations)?**

We have had positive feedback on the responsiveness and performance of the system by the Keller-Williams office staff. Their machines and ours have all ran the web app successfully, at an efficient speed, from our sample site hosted on Azure. When possible we've used efficient querying techniques to query the database only for relevant data, rather than retrieving all data every time. One example of this is in IInteractionsRepository.cs using an IQueryable for GetAllInteractions.

**5. In retrospect, would a different design have been better?**

Our design has been effective and easy to continue adding features to. Different approaches could have been taken, but we've found our current design to be a strong one that can accommodate changes and adapt for future maintenance of the web app. One change we could have made, which could have made the code more maintainable at the cost of client accessibility, is the use of separate pages for all editing of entities rather than modals.

--Josh

**iii. Assessment of the implementation of your system.**

**1. Will the code be easy to maintain?**

**a. Is it easy to fix bugs?**

With our bug reporting functionality we have an easy method of getting feedback from users that includes details about their browsers, what page they were on, their personal comment, and numerous other specifics. Using this, we've found it easy to detect and fix bugs that have come up thus far. The only possible bug fixing challenge we believe has the potential to cause problems is the heavy integration of JQuery into our razor views.

**b. Will it be easy to add new features to the system (how do you know)?**

We were able to complete all of our initial features a few weeks ahead of schedule. After that, when our client thought of adding the functionality of auto-adjusting the priority of tasks when they got closer to the due date, we were able to add that functionality to our next sprint and have it working by our next client meeting. We've had success in adding version 2 features. It should continue to be feasible for anyone maintaining the system to continue adding features with reasonable ease.

**c. To what degree did you follow accepted coding practices (separation of concerns, DRY code, no “dead” code, minimal dependencies)? Give examples.**

Our architecture uses the MVC framework which has effectively kept a separation of concerns. We briefly expanded on the file structure to add a section specifically for repositories and their interfaces, which continued to enhance keeping our code organized. DRY has been part of our design philosophy from the start and we've followed through on that by re-utilizing modals and partial views to prevent the need from repeating similar code. We've removed “dead” code when needed, as an example, we had some partial views that we found were no longer necessary and were removed. MVC has enabled us to keep a clear view of our dependencies and limit any unnecessary complications. The only exception to best coding practices is our use of in-line styling on some of our elements, for example, on BrokerInteractions.cshtml the add interaction button uses in-line styling to adjust the margin.

**2. Is the code well documented internally?**

**a. Descriptive names for classes, methods, and variables**

We've kept consistent in our naming conventions for our classes, methods, and variables. For example, some of our modal pages are named \_AddBrokerModal.cshtml, \_EditBrokerModal.cshtml, and \_AddMessageModal.cshtml. Each name clearly describes the purpose and scope the modal is in relation to. A similar convention is used for repositories, view models, and many other names in our code.

**b. Comments where needed to explain what your code is supposed to do**

Parts of our code where we utilize less intuitive processes, or where there may be future issues with maintenance, we provide comments to provide insight into our thought process. One example of this is in our HomeController.cs where we use SQL to directly query the database we have a comment noting that this method could be affected if changes are made in the future to the models. Another example is in our JavaScript, specifically the customScript.js file, where we clearly note what each section of code is addressing.

**c. Are there unit tests that make the purpose of methods explicit?**

We utilized unit tests during our initial implementation of our controller classes with success. They ensured that the basic functionality of our methods were working appropriately. As time became a factor, and our scope continued to grow, we were unable to find the time to update the tests as our program evolved.

**3. Will your code run fast (for example, will your database support a heavy load of users, or will large numbers of moving objects be rendered smoothly)?**

At no point in time during development was our web app running slowly. Our alpha testers, utilizing our web app hosted from Azure, had no issues with speed. When we had the Keller-Williams office staff use the web app they also had no problems with speed.

**4. Is your code efficient (did you accomplish your goal without writing a lot of extra code)?**

The addition of new features has expanded the amount of code required to accomplish what our clients wanted the web app to do. However, with that being said, we've frequently kept to a DRY philosophy and have been able to re-use modals and partial views to limit how many separate repeated sections of code we've had to write. We've done a few “nip/tuck” passes through the code to remove unnecessary sections and streamline what's being utilized. Considering our ten week time frame and the scope of features we were able to implement, we feel our code is in a reasonably efficient state. The only exception to this is the customScript.js file's earliest code, where we were using repeated logic for different cases while exploring possible solutions. As our code evolved our customScript.js code became more dynamic and efficient and future development would enable us to streamline our code.

**5. How bug-free is your code (are there known bugs, does it crash unexpectedly, or does it do the wrong thing at random times)?**

We've managed to tackle nearly every major bug. Only extreme corner cases remain at this point in time. For example, we found that when an admin would delete a staff member it would delete the StaffProfile but not the actual StaffUser. We were able to make the adjustments in our final week and get this problem handled. To date the only major issue we imagine might be a problem is handling concurrency errors if one staff member is attempting to modify a task that another staff member has deleted, but we have plans in place to begin development of a fix. The basic functionality of our web app is in a strong place for day to day use as a bug free tool, with the caveat that we are looking forward to the Keller-Williams office staff stress testing the web app by using it on a daily basis.

**6. How do you handle run-time exceptions?**

Our implementation of handling run-time exceptions is limited and not fully complete. Logic is in place for catching errors, for example in AdminController.cs we have an if/else structure in our BrokerEdit method to display an error if the broker object we're editing is null and then redirects back to the brokers page. There isn't functionality for displaying feedback about the error to the user at this time, so the user experience still feels clean, but we don't have any tools currently to log run-time exceptions for developer use.