Summary

Django proved to be a good framework for our project with its simplicity and emphasis on quick development. Still, its steep learning curve meant it took time for the team to become efficient using it. Github worked great for easily coordinating our team members' activities, and we had very few issues using it effectively, presumably because the team was already familiar with many of its features. As DevOps manager, I tried to automate many of the tools, including Heroku and continuous integration so that my team did not have to learn to use them. This saved development time, but also meant that they were unable to fix any bugs they had themselves.

Individual Platform/Tool Discussions

Django

Django makes development much quicker due to its abstractions, especially for the view and control parts of the MVC. We had a few more issues with the model part, as its abstractions tend to be more "leaky," and the models require more adjusting. A lot of the implementation details are hidden, which is typically helpful for simplifying development, but makes it more difficult to integrate external APIs, as it is harder to tell when things went wrong. For example, when a variable name in settings.py is misspelled, the program does not crash. Instead, it doesn't apply the change you want to. This is one of the consequences of using interpreted languages like Python. The upsides are that it is writing code and quick testing for web development, as the entire project does not need to be compiled beforehand. Django removed a lot of boilerplate code that typically comes with web development. The simple and succinct syntax is useful for writing lots of similar functions in views.py, and makes learning Django easier for our team, which did not have any previous Django experience. Still, Django has a relatively steep learning curve. At first, my teammates and I struggled to deliver code quickly, but code velocity increased by the time we reached Sprint 6.

Testing helps to mitigate some of these problems. My advice to students next semester is to write tests for successful loading of all of the pages early on into the project, so that even if you inadvertently change code that causes code somewhere else to crash, you can catch issues earlier and more easily.

GitHub

Github is very helpful in coordinating coding across multiple team members. It is also great for reviewing what teammates have already done. The only problem we had was that it can be difficult to tell exactly what the effects of a given merge will be. The workaround is to make the merge, test the results, and revert it if there are unintended side effects, which can be inconvenient. We found that GitHub issues are not very helpful because the format in which we have to talk about features is too structured. The system is small enough that a google doc is enough to keep track of all of the issues. My advice to students next semester is to put effort into writing accurate commit messages for yourself and your teammates.

Heroku

My team did not directly interact with Heroku at all. They interacted with the live website, but the continuous integration handled everything else. Initially, I was committing via the Heroku CLI, but continuous integration made everything much more smooth. There was one time in which the continuous integration tests passed, but the build failed on Heroku, which my teammates were unable to debug due because they lacked access to the crash logs. However, this was more of an issue with finicky migrations in Django than anything else. Otherwise, I found Heroku to be extremely straightforward to use, and my teammates found heroku and continuous integration to work well. My advice to students next semester is to fully learn to setup django

projects using the command line only, then run most of the same commands but prefixed with “heroku run” for simplest project setup.

Continuous Integration

The rest of my team was not familiar with how continuous integration worked, which was not a problem for most of the project. The default Django yml file in GitHub got me most of the way to a working pipeline, making it very simple to set up. I would advise next semester’s students to begin using Continuous Integration on the first sprint after first using Heroku to make development as simple as possible, and for your team to have a set production procedure from early on.

Instructor-Specific Comments

I integrated all of the tools during the sprint when they were required. I think continuous integration should be required much earlier, as automated testing is really helpful to have early on in development, and Amazon S3 doesn’t need to be implemented until much later in development.