Developer Behaviors and Team Processes

Release 4 Evaluation, February 8, 2019

|  |
| --- |
| **1. Testing with unit tests and other tools.**  We have written unit tests to ensure proper operation of parts of our code as it is written. Our unit tests for the back end are written in python using the “unittest” library. There are some parts, such as the gphoto2 commands, that cannot be mocked or easily tested, but we make sure to test as much as possible. We have manipulated previous tests, as the code that we were testing has changed due to adding another camera into the mix. We have also added a few unit tests for the communication code. These tests have to do with deleting files that are not selected by the user on the front end. In terms of the front end, we’re in the process of implementing usability testing and unit tests.  **Documentation:**  Gitlab CI |
| **2. Always have a working build of the product with tests running.**  On Spreetail’s GitLab repository, we have a branch named “master” which contains our working build of the product. When all the features in “develop” are thoroughly tested, reviewed, and approved by the Development Manager, the branch can be merged to “master”. The master branch is where we monitor our working build. Master will always contain working, tested code from the previous release. We use GitLab’s CI to build our code every time something is merged or pushed to master. We also perform these builds on the develop branch, so that we can be assured that the code being pushed to develop is tested and build as well. GitLab runs a script called “.gitlab-ci.yml” that allows us to customize how we build our code. We are using “Docker” to place our code into containers that can be run on almost any operating system. Our CI script runs docker build commands that build the camera control and front end docker containers. In this docker container build, our unit tests are run, and the build will fail on any test failure.  **Documentation:**  Gitlab CI |
| **3. Run static analysis tools to remove errors and warnings.**  Hardware and Communication:  These components are built within the PyCharm environment. We are using the default static analysis tools provided by JetBrains in this development environment, which follows the PEP (Python Enhancement Proposals) standards. This provides active syntax highlighting whenever a warning or error shows up.  Front End:  The Front End is being built in React, NodeJS, JavaScript, SASS, and HTML using Visual Studio Code Editor (VSCode). VSCode has a vast amount of plugins that can be installed to provide code analysis. The libraries we’re using are called ESLint and Prettier. ESLint extends the Prettier extension, as well as Airbnb’s open source config files for formatting (installed through node package manager). We’re also able to add our own configurations on how we want to format the code.  **Documentation:**    Front End: VSCode -- ESLint and Prettier |
| **4. Maintain a story map using the Senior Design story format.**  We use StoriesOnBoard to maintain an up-to-date story map for the project. This story map represents the high level tasks that we need to get done in order to complete a specific iteration. We use this story map to derive user stories and do sprint-planning later in Zenhub.  **Documentation:**  [**https://360sdspreetail.storiesonboard.com/m/360-spreetail**](https://360sdspreetail.storiesonboard.com/m/360-spreetail) |
| **5. Maintain up to date definition of done and backlogs for each milestones and sprint.**  Our definition of done can be found on our github wiki page. This page describes the criteria for code to be “done” in our project context. We were able to develop this set of criteria by meeting with our sponsor and discussing what review, testing, and building needs to happen before code can be considered done. We have also updated all of our Zenhub stories to contain “Acceptance Criteria”. These criteria are mainly the developer tasks for the story rewritten as completed events. This allows us to look back when completing the work for a story to check and see if all criteria for being done is met by the development progress made.  **Documentation:** Zenhub:<https://app.zenhub.com/workspaces/spreetail-5bbceb414b5806bc2bec3047/boards?repos=145894767>  Github wiki: <https://github.com/cseseniordesign/spreetail/wiki/Definition-of-Done> |
| **6. Estimate stories and assign points in the context of design.**  Our sprints are now 1 week in duration, and we do sprint planning every Thursday at the end of our team time. We estimate our stories when adding them to the backlog by applying a generalized tag that represents a small, medium, or large task. When we plan stories into the sprint from the backlog, we then use a form of planning poker. We look at a story, and we all write down how many points we believe the story would take. We then look at all of the point values and discuss the story if points do not correlate. This method of estimation has been very successful and easily adopted.  **Documentation:**  Zenhub: <https://app.zenhub.com/workspaces/spreetail-5bbceb414b5806bc2bec3047/boards?repos=145894767> |
| **7. Release milestones to the sponsor.**  At the beginning of release 4, we met with our sponsor to nail down what is expected for the end of the release. The two main tasks that we decided on were completing the hardware setup and as code to control 2 cameras as well as discover and implement a simple version of communication between the front end and the hardware. At the end of release 4, we presented a complete hardware rig along with a demonstration of both cameras working simultaneously from two angles. We also presented our new front end preview page as well as the functionality to request preview images and display them on the front end. This also demonstrated the new architecture that we absorbed by including the Azure Service Bus.  **Documentation:** |
| **8. Have a security plan for the highest security risk areas of project.**  Our two highest risk security issues for the project are access to authentication keys from the web application and complete access to our web app from anyone. If a user accesses our web page now, they are sent the javascript code that displays our front end components as well as connects to the Azure message service. Users can inspect the web page and get the authentication keys that allow access to the Azure message Bus. Also, anyone can access our page by entering the url for our web app.  We’ve looked into both of these issues and have come up with 2 solution that would solve both issues. First, we could limit access of the web app to only Spreetail’s network, which would ensure that only spreetail employees on the network have access to the web app and the authentication keys. Second, we could implement a login feature from Spreetail that requires users to log in with their Spreetail credentials before being allowed access to the web app, which would also keep the authentication keys in the javascript safe. These solutions will be discussed with Clayton during the next Sponsor meeting to determine the best solution  **Documentation:** |
| **9. Measure performance and use telemetry to inform decisions.**  We have taken consideration into the performance of our cameras, communication techniques, as well as computer hardware. Switching from a Raspberry Pi to a Linux tower has granted us with better overall performance within the CPU as well as transfer speeds from camera to hard disk.  Cameras are plug and play as long as they are compatible with the gphoto2 library. However, switching from the Nikon D3400 to the Canon EOS 6D Mark II allows us to snap pictures and auto focus faster and more consistently. The Canon EOS 6D Mark II has more options for auto focus, being able to determine the size of the focus. We’ve also updated the lens to a prime lens instead of a standard stock lens. Prime lenses (50mm f/1.8) can capture more detail, compared to a stock lens(18-55mm f/3.5-5.6), which focuses on the bigger picture. However, the lens are also interchangeable based on the situation.  Communicating to the tower and sending an image back took an extreme amount of time (15 seconds to take, transfer, downsize, and send image) when doing it via direct connection. This was reduced to around 5 seconds when we sent it through a new communication layer, Azure Service Bus. We tested sending 50 images that were already on disk and were able to send them all out within 20 seconds. Since it’s a message queue, the front end (or whoever wishes to receive it) is able to pull an image right when it arrives in the queue.  We began extensively logging actions within the backend for both the camera controller and the connection layer. This has made it extremely helpful in diagnosing issues that came up during development this release. We log to static log files stored on the hardware rig. These files are named by  **Documentation:**  Gitlab Wiki:<https://github.com/cseseniordesign/spreetail/wiki/Telemetry-and-Testing> |
| **10. Maintain cadence iterations.**  We do our sprint planning at the end of team time each Thursday, which is the end of the week for our team. This allows us to look back and see what we’ve completed as well as what may need to get pushed into the next iteration. We generally plan for the number of team time hours available that week, but we sometimes plan more if development speed needs increased. Overall, our burndown charts for each sprint were very vertical, as most stories were completed on Thursdays at the end of the week. We have labeled all of the R4 stories in Zenhub to be a part of a new milestone called R4. This allows us to see the burndown for the entire release, which can be seen below.  **Documentation:**    Zenhub: <https://app.zenhub.com/workspaces/spreetail-5bbceb414b5806bc2bec3047/boards?repos=145894767> |