FosterLove Transition Report Update

Tomas Viejobueno, Harriet Khang, Aashai Avadhani, Hikma Redi, Yanyu Lin

Deployment

The complete product will be deployed to the AWS server owned by our client - routing our codebase changes through an EC2 instance. This EC2 instance is propagated into three availability zones (us-east-2a, us-east-2b, us-east-2c) through an Autoscaling Group and then ran through a Load Balancer - which splits the traffic accordingly among the zones. Then, the DNS records are handled in Route 53 by having aliases to flpinventory.com. The software will be contained in a docker container that provides better consistency and automation. Current deliverable we presented to the client is in the form of prototypes and feature demonstration on a local server. This could be different from final deployment, but given advantages of containerization and previous instructions, the team doesn't expect deployment to be troublesome.

Currently, local development and demonstration is based on Python and Django installation, as well as a number of library requirements. The dependencies are marked in software root directory and within the ReadMe. The deployment guide given to the user will also include the necessary dependencies and how to install them. A faster and more suitable way to tackle possible issues would be using automated dependence installation when setting up the deployment container. This will also allow future improvements like ours to be implemented and agilely deployed. Local development differs from docker deployment in that all requirements will need to be downloaded manually or at least using Django commands to install all requirements in the “requirements.txt” in the root directory.

To test the stability and consistency of deployed services, the team can utilize the credentials to the server and look in the AWS to make sure the instance is running properly. The credentials for AWS have been provided through the Drive folder from the previous developers.

Another difference between local development and server deployment is login information. To make development easier, we typically use a superuser account with manually set admin passes. This information, however, may be easily revealed if the environment configuration files are not set to confidential on GitHub. The clients use each individual’s own data for accessing the services. We should also remove any relevant data when deploying the system to the server.

Documentation and Scope

Most of the features we have implemented and are planning to implement are simple improvements to the current FosterLove site, but the main functionality will be described in documentation in how it is used by users and implementation details for future developers. Our main improvement that we will be focusing on will be the creation of a KPI dashboard within the FosterLove site. Currently, FosterLove tracks the inventory flow through an exported CSV file that is manually turned into a Google Sheets file, which in turn is analyzed by a FosterLove employee and used to create an estimated price of goods sold and received. Our KPI dashboard will not only automate part of this process but provide data analytics on the inventory and cash flow. Ultimately the purpose of the KPI dashboard is to help FosterLove automatically create a financial estimate to show to future investors and stakeholders, making it imperative for our client to be able to easily use this feature. Since the KPI dashboard will be an entirely new feature that our team is deploying, we plan to create more detailed documentation on how to interact with and use the KPI dashboard - compared to the already somewhat established and intuitive features.

The scope of the KPI documentation that we will provide to our client will cover all of the features of the KPI dashboard: the ability to track the number of items donated and sold overall and within each category, the ability to automatically generate the estimated total price of items donated and sold, the ability to automatically set the price of each category item, and the ability to keep track of the current inventory left of each item category in real time. The KPI dashboard will not only auto-generate these statistics but will also create a simple graphic over time that visualizes the aforementioned statistics. The core domain of the KPI dashboard will be data analytics.

Additional Software

There is no additional software that we are currently using. This might change in future sprint, but our documentation will reflect this if so. In any case, we will provide additional documentation within a Drive folder as part of the transition plan - which will be split into “For Future Developers” and “For FosterLove” sections.

The “For Future Developers” section will contain extra documentation compared to last year’s team explaining, in detail, how to use the AWS infrastructure and how it is currently being set up. This will include architecture and sequence diagrams - as well as a basic rundown of the architecture itself. Furthermore, we plan to add comments, as well as a basic description of how the main features are being implemented such that the future developers can easily understand and modify these features. Furthermore, we will make note of future changes that would improve the system that we did not get

to. There will also be a file containing sensitive information that the future developers will need such as email, AWS, and other login information.

The “For FosterLove” section will contain information regarding workflow and a user guide explaining how the main features can be used and the overall functionality of the system. If any other information is relevant to the client, then that will also be contained within this section.

Differences Between your Final Delivery and Sprint Delivery

The final project delivery for our iteration of FosterLove will be enhancements on the current FosterLove system. The FosterLove website that the last group built has the foundations of the current system. The overall system currently has no clear fundamental issues (in terms of the back-end functionality or front-end displays), but the desires of our clients are mainly focused on UI changes such as quantity change dropdowns and automating/simplifying processes for FosterLove on the back-end such as merging old, unused items into larger, broad items. The demands of the FosterLove clients were largely independent feature changes such as category changes, quantity change editing, google drive integration, search bar improvement, and recommended item changes etc. Prior to the first sprint, we had a meeting where we prioritized the most important tasks we have to get done for the client that we can finish in 2 weeks for the upcoming sprint. This led us to have 2-3 tasks overall that we could reasonably accomplish for the entire project. To accomplish these tasks on a sprint-by-sprint basis, we would have a meeting at the beginning of each sprint where we assigned task owners for each task based on which member(s) wanted to do said task the most.

This allowed freedom and flexibility for each team member to implement the solution in their one particular way. After tasks were assigned, we had stand-up meetings later in the week (Friday or Saturday) to check in on team member’s progress as well as have technical problems/questions we had when solving each task. The slack was used for any issues that a team member had when going through the sprint and we would schedule spontaneous pair-coded meetings to help solve urgent issues. Per sprint, we would all do code reviews for every PR by locally downloading the branch and user testing from our own perspective as well. Once a team member approved the merge, we would merge it into the master system. Per sprint QA we wanted to ensure that continuous integration such as django flow tests, unit tests, and integration tests were used throughout each task. Moreover, we believe that these CI flows would help us ensure our sprint QA is protected. For the final QA, our main value we are upholding is to “increase precision, and automation for FosterLove”.

We believe that each task accomplishes our goal(s), but to ensure this within each sprint we have individual QA for each feature such as CI and Unit Tests. The CI

tests and integration process we have (and Agile development process of bi weekly meetings, standups, pair programming) have been working well to develop features that help FosterLove. Hence, the main difference between internal planning for the final project delivery mainly focuses on larger, broader tasks that need to be accomplished, while sprint internal planning mainly focuses on smaller and specific features/tasks. In terms of QA, the main difference lies in that we use CI and Unit Tests for our sprint features, while we use more in-depth manual testing, CI, and more detailed Unit Tests in conjunction with the larger, broader tasks to make sure everything has no errors or unexpected functionality.

Transition-related Risks

For transitioning the project, we see 2 major clients for the transition plan. For the technical clients (future developers) and for the non-technical clients (FosterLove), we treat the transition differently. The non-technical clients don't need any pure implementation details (like a data structure implementation) or information regarding infrastructure, but they need to know the general functionality details such as how the change(s) impact their workflow (new features, automation, etc). We rely on the client to identify parts of the system that need improvement or changes (not from a technical perspective), but from a user perspective (i.e. change color, add a button, need a tab for a new inventory item, etc). However, we, as a team, also suggest possible ways to improve their workflow. The chance that we don't receive guidance on what we need to improve within their workflow is minimal (since we meet with them weekly and future developers should as well), but the consequence would result in us personally testing to see what aspects of the workflow could be improved. Since we already have a baseline understanding of their workflow, understanding what can be improved wouldn't be too difficult. Thus, the consequences of this risk event aren't too impactful, overall.

For the technical clients, a possible risk event is their understanding/knowledge on the codebase such that they can implement new changes effectively. However, this is not entirely out of our control. We will write extensive documentation on the interaction of the database, back-end and front-end. A majority of the project changes are dependent on the database maintenance for future iterations of the project. For example, a change in a database items list will force the search bar display to display these changes accordingly. The interactions between components in an architecture diagram and sequence diagrams are something that is needed in order for components to be added or changed throughout the lifecycle of this product - simplifying the process for future developers. Furthermore, code comments are also something that we need to add as we change the system - because they are slightly sparse except for the larger components. Given poor documentation for the codebase, then it is unclear how to

change that codebase (for example a category in the database). This is a problem we are facing currently, as a team, right now. The consequence of a poorly documented component architecture leads to a higher learning curve of the codebase and leads to complications in understanding how to change the database. For example we see that, for this particular client, the database is live and consistently being used 24/7 to track inventory. Hence, having good documentation on how to change the database is critical. Thus for the technical side, we need a concrete technical architecture diagram of how the components work/interchangeable on a system level basis.

Licenses

Our project will use a GNU GPL v3 License. We chose this license because we are running our project with open source dependencies and libraries such as Heroku that use the GNU GPL v3 License. Furthermore, there is no need for this code to be proprietary to FosterLove as there is no intellectual property that needs to be protected for this use case. The GNU GPL v3 License allows for the copy, use, modification, distribution, and sale of any of the code (open-source) - only requiring any derivatives of the work to also use the GNU GPL v3 License. It is one of the most popular, if not the most popular, open sources license(s). Due to our client only needing to utilize the project as a database system with check-in and check-out, there is no desire or need to protect any intellectual property. Hence, the GNU GPL v3 license is the most suitable for the needs of our project and client, as well as being required due to our libraries and dependencies.