Code Quality Review Report

Brandon Hillbom and Dillan Zurowski

200426340 and 200431334

Capstone 2024

Table of Contents

Introduction

The following report covers multiple aspects of the quality of code for the Intelligrain project. This project was built to be passed on to Ground Truth Agriculture and thus the importance of quality code is of the highest standards.

Analysis

Establishing Coding Standards

During project initialization, we established standards for both our code and our GitHub processes. This includes naming conventions for variables, classes, files, and naming conventions for branches and commits. This ensured that we were on the same page and provided descriptive messages for our GitHub activities. These standards contribute to the overall readability and maintainability of the project so that it can be scaled with minimal issues. It also allows developers to easily find what they are looking for.

Architecture & Separation of Concern

We changed our initial architecture plans to ensure we are separating aspects of our application to follow a Model-View-Controller architecture. Instead of having all of the data processing on the frontend application including database queries, we created a FastAPI server that utilized API calls to a python server which is where the heavy database calls are made. This also cleaned up the code as the larger query functions take place on the server in Python

rather than on the application in Dart. Python is also more heavily supported by available plugins which allow better data manipulation.

Within our Dart files, we implemented a Model-View-Controller (MVC) architecture for each feature. Separating what the user sees and interacts with, the logic, and API calls was important for scalability. This allowed for quicker debugging and cleaner files as each file in the folder structure had a specific purpose. We also further divided our folder structure for user interface elements into screens and widgets so that it would be easier to know which widgets can be integrated with others and which widgets are stand alone pages.

Proper state management was also crucial for our application as we utilized Flutter's global provider for the user state and the cached Grain Predictions, which uses the context of the widget tree to read and write. This meant that if the provider was being used by a widget that no longer existed, it would cause dependency issues and unexpected errors. By utilizing callback functions to ensure context was not used across asynchronous functions and by ensuring all widgets were properly disposed of, we maintained quality code architecture in our Flutter framework. However, it is our intention in future iterations to move away from the standard provider and implement an alternative that is more scalable and doesn't rely on context. This would decrease the possibility of using context asynchronously or from a deactivated widget.

Coding Best Practices

Our code avoids the use of hardcoding data where applicable. However, since we don't have access to the camera model and can't run this application in the field, we have a preset

JSON file where we read our samples from. This simulates how the Orange Pi computer on the combine would store it's grain predictions in the JSON file. With this in mind, we built the application so only slight modifications are needed by Ground Truth Agriculture in order to have the Orange Pi write to the file and the mobile app read from said file. Another place where hard coding was necessary was assigning field identifiers to grain predictions. We did not implement the ability to create and manage fields but we wrote the code to only show grain predictions from the defined field. This means that implementing this feature would cause minimal impact to the data we already have. That being said, everything else in our application uses data from the database which can be manipulated by the user on demand. Our data from the JSON file also gets added to the database for the admin to use to simulate getting new real data.

An important part of ensuring best practices was type checking which kept us from casting variables to other types without implementing the proper error handling. We found this made it easier when developing and debugging on the front end side. However, our backend was built with python which does not enforce type checking. This made it difficult to develop at times and opened our backend to possible type errors. We mitigate this risk by enforcing type checking where we could. We also utilized environment variables for safe storage of our private API keys so that we could minimize security concerns. We also improved type checking and possible String values by implementing enumeration which enforces that the value be one of the available options. For example, our team member's status and role type each have 3 enumeration values.

Inheritance was also included where possible to avoid passing unused data around our app. One class we made, "DisplayGrainPrediction", is a class that passes/stores data for filtering the samples on the map screen. This class inherits from the class "GrainPredictions"

without using any of the unused fields which also improves performance due to the vast data that may exist in a GrainPrediction.

Comments are used above each widget and function to describe what the function does and why it is needed where it is not obvious. For example, a widget that simply displays a button is less important to comment on than a more complex widget. We also added any mentions of workarounds and how they will potentially be changed when released to Ground Truth Agriculture.

Functionality

Since our application is user-based, ensuring quality functional code was a top priority. We performed multiple user tests with real farmers as demonstrated in the User Testing Report. We made sure that all feedback was taken into account and we refactored our code frequently to ensure our solutions were meeting the expectations of our user. We tested our application to identify bugs and ensure they were fixed promptly so our application would provide the most user-friendly experience possible. Our interfaces are intuitive by utilizing signifiers and affordances for interactable elements. For example, we found that our user didn't know they could click on the label of our graph so we added an info icon signifier so that it affords to be clicked. We also focused on providing feedback to users in the form of confirmation popups, information popups, toast messages, and state changes. This ensures the user doesn't get caught wondering if they did something wrong or the application is broken in some way.

Non-Functional

At the end of this project, we will be handing the code to Ground Truth Agriculture. This means that we spent a considerable focus on creating quality code and focusing on factors such as ensuring maintainability, readability, reusability, and extensibility as we want the handoff to be as clean as possible. One of the first requirements was that the application needed to be coded in a way that they could easily add, remove, or change features when needed and that was something we considered while developing the project. We did this by first ensuring that the file structure was clean and easy to understand. We used an MVC pattern and structured our files accordingly. Structuring the code this way allows Ground Truth Agriculture to extend and scale more easily.

Performance

We increased performance by moving the heavy data processing to python's FastAPI server to decrease the load on the front-end application. However, this creates a new bottleneck as the APIs rely on a secure and fast network connection between the two devices. As well as the API server device must be on the same network, with the server's IP address added to an environment variable on the flutter application before being run. Algorithms and data structures are as efficient as possible and the plugins we used were analyzed to ensure performance is maintained.

There are other optimization opportunities regarding how we cache the samples collected. Currently, we are using the Flutter Provider plugin. This works for our purposes however, we may need a different plugin when scaled up for a larger field. Older devices also

have some issues as load times are significantly worse and they may not meet the requirements of some APIs such as the mapbox_gl API.

Security

Security is something that we took well into consideration. We implemented secure environment files and made sure important API links and passwords were included in there. However, we failed to implement the API server over HTTPS which is something that we were hoping to implement. HTTPS was working on the server side, however, we had issues with the DIO plugin we were using on the front end as it did not seem to accept requests on HTTPS, only HTTP. We attempted to mitigate this problem but found we were spending too much time on it so we decided to move it to the backlog for now. This is not an issue at the moment since GTA will use their web-based server to communicate with the application in the future.

Testing

As mentioned in our User Testing report, we performed a variety of user tests with Ground Truth Agriculture team members, real farmers, and random users with a variety of technical backgrounds. The tests we performed allowed us to ensure we were meeting the needs of the users as it is a user based application. Our user tests showed us the changes we should make as well as what designs they liked such as the modified dot design over our original path design. We also had the chance to listen to the users' stories to learn more about the problem and why this app is important to them.

We also performed a number of code tests as seen in our Code Testing Report. These tests allowed us to find bugs in our code that needed to be fixed as well as ensure that our code meets a quality standard.

Dependencies

Dependencies are properly handled on the front end through the pubspec file and running “flutter pub get” in the terminal. The server dependencies are handled through the Python package Pip where required packages are listed in the requirements.txt file and can be installed on any device using the command “pip install -r requirements.txt”. This allows our dependencies to be consistent with all servers on startup.

Version Control

As mentioned above, commit message standards were set during project initialization and were maintained throughout the project lifecycle. For example, if we were implementing a feature, we would write the commit message as “Feat: a descriptive message.” We would also name the branch using the type of change we were doing first, followed by a couple words describing the change. From the previous example, we might have written the branch name as “feat-descriptive-message”. We were careful to always include a descriptive message for each commit and branch name and followed the steps in our GitHub wiki to ensure each time we merged there were no issues. This allows us to refer to our changes in version history more easily.

Conclusion

We believe our code is up to quality standards and with minor adjustments, can be used in the field by Ground Truth Agriculture. We followed basic coding standards such as consistent naming convention, separation of concerns for the front-end and back-end logic, ensured the code was readable and maintainable, and made sure our application followed all functional requirements determined from our user tests and Ground Truth Agriculture.