Team Apollo

Packagebird

Elisha Aguilera, Setenay Guner, Denish Oleke

Client and Mentor - Brandon Busby

Instructor - Dr. Bolong Zeng

CPT_S 423 – Software Design Project ll

Submitted 03/04/2022

Description Updates - 1

As the month has progressed, the project's internal architecture, while sound from a high-level of abstraction, has some presented issues with the implementation, principally the high degree of coupling with certain functions revolving around loading and unloading from the MongoDB data persistence component. Some of the decisions made earlier in development are now slowing down our present development speed. However, as it stands, the project will meet its requirements by the conclusion of the Beta development period. The architecture has been settled and will not change for the remainder of the semester. However, internal components may be altered and extended, notably the logging and diagnostic tools for server management.

Technical contributions made by Elisha Aguilera were the design and implementation of the package building process, which is triggered from an authenticated client issuing a build command, which relays the package name and version to the server over a gRPC method stub. Once received, the client finds and extracts the package along with its dependencies, then calls the build shell command or commands attached to the package on that directory. The testing process is virtually the same, except in the place of build commands, a test shell script is triggered. The results are collected and relayed back to the client for display on their terminal.

Elisha Aguilera and Setenay Guner designed and implemented portions of the configuration process; the server can reference a config YAML file located in the same directory as the executable or generate a hardcoded default config if none are present. This stores the location of project and package source files, the MongoDB connection address and other configuration parameters. This is needed since the server must tolerate shutdown and other interruptions, and since there is only a single configuration a simple file is sufficient for this functionality.

While designing and implementing package handlers, a portion of the implementation was contributed by Denish Oleke along with the server response time following the fundamental layout set my Elisha Aguilera besides the documentation that was worked upon by the team in general.

Considering future iterations of this project, it may be helpful for teams to compare the state of this package manager to theoretical package managers, such as MPM developed by Pietro Abate and company at the University of Paris Diderot (Abate et al., A Modular Package Manager architecture), which uses a modular architecture to solve common issues with a focus on flexibility. Our project has opted for a constrained non-modular approach, although the client can be written in several languages so long as it remains compliant with the gRPC defined interfaces provided.

Beta Prototype Test Results – 2

Unit Tests – 2-1

Unit and integration test were primarily conducted by Setenay Guner with assistance from Elisha Aguilera due to varying levels of familiarity with components of the application. Due to some components containing auto-generated code, namely the gRPC server stubs, not every line was covered in a conventional sense. In addition, some of the components rely on side-

effects involving the filesystem and MongoDB database documents, and as such those tests may not be considered “pure” in a traditional sense. However, for the future the adjusted coverage of the code will be documented prior to the final beta demonstration and release since we believe by then any other issues including any bugs will have been fixed as required.

	Line Coverage	Branch Coverage	Covered Lines
>DatabaseInterface	42%		203
>PackageDBInterface.go	47%	87%	146
>ProjectDBInterface.go	45%	71%	57

Table 2-1-1. Line and branch coverage of the unit tests.

Network Operations Tests – 2-2

Network operations testing was primarily conducted by Elisha Aguilera. As the application consists of two networked components it is prudent to test the responsiveness of a battery of operations in various conditions. As stated in the *Beta Prototype Update* document, to precisely control network conditions, we utilize *Clumsy version 0.2* to simulate increased latency and jitter in increments. The final case is testing operations against a server hosted on a Virtual Private Server located in Los Angeles, approximately 884 miles away from the client’s location in Sultan, Washington with an average round-trip ping of 34 milliseconds as measured from PowerShell.

Detailed below in Table 2-2-1 is the various parameters altered such as latency and jitter, the operations conducted are synchronizing files, listing server content, and the time taken for the operation to complete. All results were collected from a client running on Windows 10 x64 bit with a server over localhost connection along with a locally hosted MongoDB or hosted on a remote Ubuntu 18.04 Docker container running on a remote VPS in Los Angeles with an average ping time of 37 milliseconds from 64 samples, and approximately 844 miles from the client.

Operation Conducted	Condition	Time (varied units)
Syncing Full Gigabyte Source	VPS LA Docker Server	~214.7 seconds
Syncing Half Gigabyte Source	VPS LA Docker Server	~103.48 seconds
Syncing Quarter Gigabyte Source	VPS LA Docker Server	~51.73 seconds
Syncing Full Gigabyte Source	Localhost 250ms Delay	~5.35 seconds
Syncing Half Gigabyte Source	Localhost 250ms Delay	~2.56 seconds
Syncing Quarter Gigabyte Source	Localhost 250ms Delay	~1.36 seconds
Syncing Full Gigabyte Source	Localhost, 250ms Lag, 25% Change Drop, 30 Millisecond Throttle	~5.28 seconds

Syncing Half Gigabyte Source	Localhost, 250ms Lag, 25% Change Drop, 30 Millisecond Throttle	~4.19 seconds
Syncing Quarter Gigabyte Source	Localhost, 250ms Lag, 25% Change Drop, 30 Millisecond Throttle	~1.34 seconds
Listing Server Contents	VPS LA Docker Server	~2.52 seconds
Listing Server Contents	Localhost 250ms Delay	~1.98 seconds
Listing Server Contents	Localhost, 250ms Lag, 25% Change Drop, 30 Millisecond Throttle	~3.17 seconds

Table 2-2-1: Network operations tests and results in battery operations.

User Interface Tests – 2-3

A volunteer was solicitated by Elisha Aguilera for the purpose of evaluating the usability of the software from the standard command line client. The subject was instructions to create a project, make and synchronize modifications to the project, create a package and run a build command on the package. These tasks were completed by the subject with minimal instruction from the test conductor. Table 2-3-1 lists the operation asked of the user, the general rating of the usability, and any notes on the operation as shown below.

Operation Performed	Rating (Poor, Moderate, Good)	Notes
Creating a project	Good	Manual instructions should be better. Operation still straightforward.
Listing server contents	Moderate	Package names are confusing and should be specified where the name and version are. Should list all contents if no flags provided.
Syncing a project	Poor	Should be specified in the help feedback and manual. Perhaps included in the feedback from creating a project.
Creating a package from project	Moderate	The manual should have more details and feedback should be clearer.
Building a package	Moderate	Feedback on terminal should be clear and give option to transfer results back to client instead of separate operation.

Table 2-3-1: User interface operations performed and test results.

The next section was the setup of a new Packagebird-server from an Ubuntu Linux server within a Docker container, again with minimal feedback. This was to gauge how simple the install and run shell scripts are from a new user's perspective. Like the prior usage test, the instructions are conveyed to the subject with minimal input from the conductor beyond clarification. Table 2-3-2 lists the setup operations asked of the user, the general rating of the usability, and any notes on the setup operation.

Operation Performed	Rating (Poor, Moderate, Good)	Notes
Setting up a server from a shell script	Good	Operation is straightforward and instructions are clear.
Starting the server from a shell script	Moderate	Run shell script should be bundled with install script to minimize number of downloads.

Table 2-3-2: Server setup operations performed and test results.

Beta Prototype Validation Results – 3

Unit Tests Validation – 3-1

The unit test results were generated with Cover and Golang's testing package, revealing 35% of lines in Package and Project covered by all our test. These indicate the general state of the beta prototype is in decent code health; however, it should be noted that these tests should not be considered "pure" due to the side-effects from external systems that are difficult to control-for, and that only line-coverage was measured. Were there more time and ability, conditional and mutation testing could also be applied for this project and potentially reveal cases not originally caught.

These tests were written to measure the functionality of Project and Package database interaction. We tested adding new packages and projects, deleting a single package as well as deleting all packages. Execution of tests compare actual and expected values and result fail if they are not a match.

Network Operations Tests Validation – 3-2

The network operations tests results were generated by the methods outlined in the Network Operations Tests section; referencing Table 2-2-1, file transfer operations were as expected most affected by latency increases. In the “real-world” network conditions operations performed as expected with most operations. A bar-graph of the results is shown below in Chart 3-2-1.

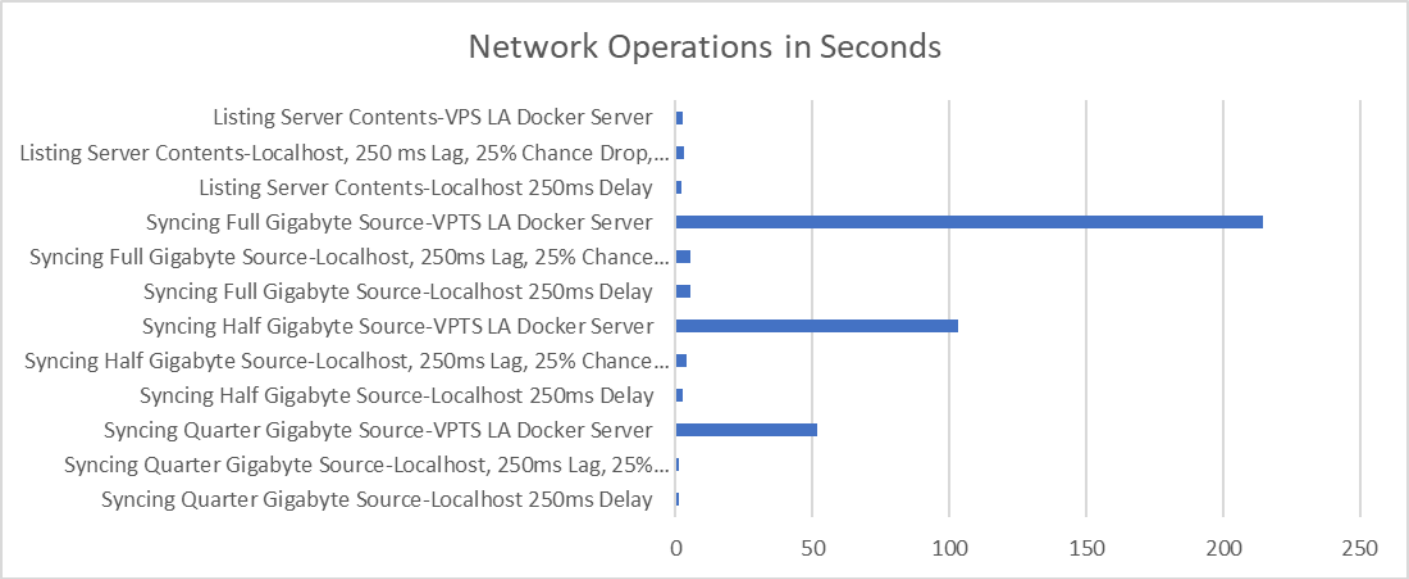

Chart 3-2-1: Horizontal bar chart of network operations in seconds.

This fits within our previous expectations, gRPC is a well-defined protocol and RPC standard, and has been used in situations requiring much more precise and deliberate streaming of information, such as transmitting in near-real-time failure notifications for optical networks (Paolucci et al. Demonstration of grpc telemetry for soft failure detection in Elastic Optical Networks). In our case, most operations are not considered time-sensitive so long as they are completed within a reasonable time. Moreover, file transfers are much more dependent on throughput of information as compared to the rest of the operations, requiring only signals and simple information relayed to-and-from the server.

User Interface Tests Validation – 3-3

Most tasks were performed with assistance from the help command and the manual; however, many of the commands were long and ‘complicated’ as compared to traditional POSIX-style commands consisting of contractions and acronyms, the leading ‘Packagebird’ command. Replacing the existing commands with shorter commands would benefit memorability and ease of input. The ‘list’ subcommand with three-letter flags seemed memorable to the subject.

A particular trend observed is that operations should have better feedback to the user when called as opposed to the constrained and simplified responses currently used. This is good interface design in a graphical context, buttons will highlight when the cursor is placed atop and alternate color or shade when clicked. This feedback can be translated into a terminal context by having more informative responses and possibly catching unintended syntactical errors; while our project must work in environments with limited graphical output and interaction, having richer feedback to the user assist will likely encourage engagement and improve proficiency with Packagebird over time; if possible, we could simulate the very detailed responses as shown in the prototype command line interface Gracoli, developed by Pramod Verma (Verma Gracoli). This interface is far too complicated for the scope of the project but organizing system responses into tables is an easy improvement. All tasks can be referenced in Tables 2-3-1 and 2-3-2.

Summary of Work Remaining – 4

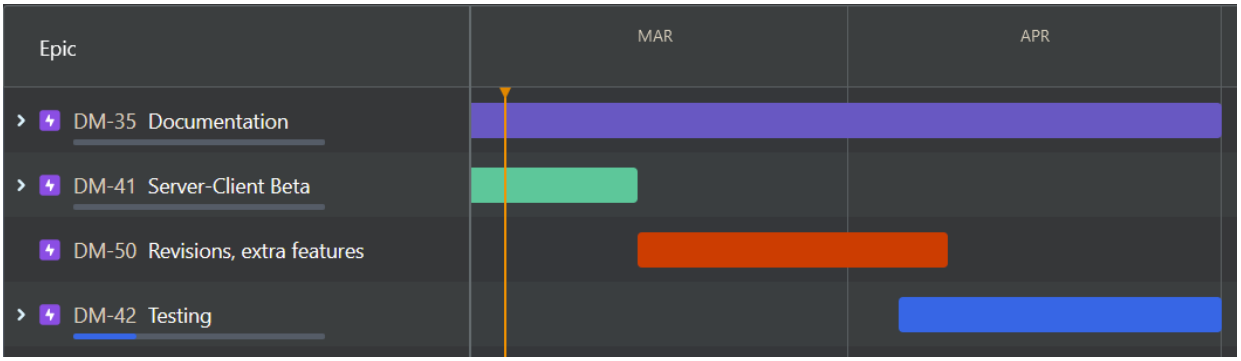

Chart 4-1-1: Gantt Chart summary of work remaining.

Despite the state of the prototype, many features need further implementation, namely team member control and privilege levels, building and testing of packages needs to be further tested, and the entire project source code needs better documentation for the inheriting team. In addition, to establish a continuation of between our projects, our client has asked that the previous team's work be incorporated as a package and built for demonstration purposes if possible. This is not as simple as assembling terminal constrained C programs, but we feel it is a worthy object to cap off the project.

Elisha Aguilera will continue making contributions on tightly linked client and server units of functionality, primarily exercising and testing features on the server related to building and testing packages. In addition, he will continue to work on the diagnostic tools for testing and validating the operation of the solution. He has already created installation and a "Shell" run scripts for the server components for Linux platforms, and per the client's request will create installer executables for the Windows platform.

Appendix

Team Photos

Elisha Aguilera – Team Liaison

Setenay Guner

Denish Oleke

References

Abate, Pietro, et al. “A Modular Package Manager Architecture.” Information and Software Technology, vol. 55, no. 2, Feb. 2013, pp. 459–474., https://doi.org/10.1016/j.infsof.2012.09.002.

Paolucci, F., et al. “Demonstration of Grpc Telemetry for Soft Failure Detection in Elastic Optical Networks.” 2017 European Conference on Optical Communication (ECOC), 2017, https://doi.org/10.1109/ecoc.2017.8346066.

Verma, Pramod. “Gracoli.” CHI '13 Extended Abstracts on Human Factors in Computing Systems on - CHI EA '13, Apr. 2013, https://doi.org/10.1145/2468356.2479631.