ECE 46100 Team 3

Parker Bushey, Kaitlyn Roberts, Ashwin Senthilkumar

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do.
Accountable together - We are Purdue.

Signed: Parker Bushey, Kaitlyn Roberts, and Ashwin Senthilkumar
Date: October 25, 2021

Table of Contents

1 - Tool Selection and Preparation

1.1 - Programming language, toolset, component selection

We will use Python for this project's code, which will be stored on Github. We will use Python to interface with Github REST API and Google Cloud Platform. Npm-audit will be used to perform security audits on packages. RESTler will be used to perform automated security probing of the registry. The Web interface will be HTML, and automated tests will be run using Swagger OpenAPI.

Resources:

- https://docs.github.com/en/rest/reference
- https://docs.github.com/en/actions/learn-github-actions/
- https://cloud.google.com/docs/overview, https://cloud.google.com/sdk/docs/scripting-gcloud, https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app
- https://docs.npmjs.com/cli/v7/commands/npm-audit
- https://github.com/microsoft/restler-fuzzer, https://github.com/microsoft/restler-fuzzer/blob/main/docs/user-guide/QuickStart.md
- https://swagger.io/specification/

1.2 - Communication mechanism

For setting up meetings and general communication, we will use GroupMe. If meetings do not take place in person, Webex will be used to allow for distanced synchronous work. All changes to our code are tracked through GitHub.

2 - Starting Project Analysis

The given project 1 implementation is written in Python. Even though we wrote our project 1 in Java, we are deciding to switch to Python for project 2 so that we can easily interface with this implementation. In assessing this implementation of project 1, we found that a given test file of URLs provided reasonable scoring for each of the sub scores and overall score. The program runs quickly and the log file is readable and contains all necessary scoring information. Additionally, all the dependencies are common modules that we are familiar with and trust. Therefore, this implementation is trustworthy.

3 - Team Contract

3.1 - Commitments

- Each team member will complete the work that is expected.
- Team members will complete the work given to them to the best of their ability.
- Everyone is expected to keep each other informed of their progress and struggles.

3.2 - Meetings

- Team members are expected to attend all meetings unless other members are alerted ahead of time or in the case of an emergency.
    - If someone cannot make it to a meeting, they will alert other members at least a day in advance.
    - In the event a meeting is missed, work shall be made up on one's own time.
- Each member will come prepared to meetings.
- Team members will be respectful to other members during meetings.
- Weekend meetings will be dedicated to completing the milestone documents and recording progress made / roadblocks that week

3.3 - Formatting

- Format code in an organized manner.
    - Class and variable names should be written in camel-case.
    - Use descriptive variable names everywhere that is appropriate.
- Include comments on important parts/changes.
- Avoid putting too much on a single line.

4 - Team Synchronous Meeting Times

Meetings will take place over Webex and we will plan on at least twice weekly. Additional meetings will be scheduled as needed. The meeting times are as follows:

- Weekly Meeting 1: Saturday, 1:30 PM
- Weekly Meeting 2: Wednesday, 6:00 PM

5 - Requirements

* indicates requirement is a baseline requirement

5.1 - Internal

- The system will be tested with the openAPI Swagger*
- You will provide a URI (hostname+port) where the grader person can interact with the registry web server*
- The system will start in the reset state, with a default user*. If admin privileges are implemented, this user is an admin, and has the username/password given in the project spec
- The system must use at least one GCP component and use Github Actions to perform Automated tests on pulling (CI) and Automated service deployment to GCP on merging (CD)*
    - Every pull request must receive a code review from three independent sources

5.2 - Client

- System will be accessible with REST API*
- System will be able to:
    - Support interaction with 10,000 clients at once
    - take zipped files as input, representing a npm package*
        - Files may contain multiple packages in a group together
        - Packages will be given ratings by the project 1 implementation given
        - **Packages will also be given a version dependency score (the fraction of its dependencies that are pinned to the minor version specified compared to other versions)**
        - Version notation will be:
            - Major.Minor.Patch
            - Exact: “1.2.3”
            - Bounded range: “1.2.3-2.1.0”
            - Tilde and Carat ranges: “~1.2.0” or “^1.2.0”.

    - Input (ingest) a new npm package to the project 1 code, and upload it to the registry only if it scores above 0.5 on each metric*
    - Update a npm package on the registry to a specified version if that version is ingestible*
    - Debloat a package before uploading it to optimize storage
    - Reset the registry to an empty state with only the default admin user*
- Web server/interface will have Purdue Branding
- Web server/interface will have automated tests using Selenium
- Web server/interface will be able to:
    - Be ADA WCAG 2.1 AA compliant
    - Fetch local history of packages or package groups
    - Have a paginated list of all packages in the registry (10 packages per page)*
    - Request an ingestion or update of a package*, with the option to debloat
    - Request a security audit of a package, group, or the full registry
    - Allow the user to download a package*
    - Register, authenticate, and remove users with distinct “upload”, “search”, and “download” permissions.
    - Create accounts to be authenticated with a username and password
    - Create admin accounts that can register users
    - Allow a user to delete their account
    - Implement user groups that have restricted access to packages labeled as “secret”. When a user of a group uploads a secret package, only their group has access to it.
    - Provide tokens to the user to communicate with other APIs, which are valid for 1000 transactions or 10 hours
    - Execute a provided arbitrary Javascript on a package labeled as “sensitive” before uploading it
        - A sensitive package can have zero or one associated Javascript program
        - Any user can delete or upload sensitive packages, and add or run associated Javascript
        - These Javascript programs expect five inputs: “MODULE_NAME MODULE_VERSION UPLOADER_USERNAME DOWNLOADER_USERNAME ZIP_FILE”
        - If the program exits with a non-zero exit code (an error) the package upload should be rejected and the program’s error output should be propagated to the user

6 - Preliminary Design

* indicates requirement is a baseline requirement

6.1 - Server / Web

6.1.1 - ADA compliance

- This optional requirement WILL be implemented
- Website will be checked for ADA compliance regularly throughout development utilizing both the list below and thorough tests utilizing the Google Chrome Extension Accessibility Insights for Web
- Relevant WCAG 2.1 AA requirements (summarized):
    - 1.1.1 all non-text content, such as pictures or graphics, shall have alt text associated
    - 1.3.1-1.3.3 Website can be read in the correct order and with correct hierarchies programmatically without any visual cosmetics (i.e screen reader reads in correct order and can find headers and groups in the page)
    - 1.3.4 Website can be viewed on any reasonably sized screen and regardless of orientation
    - 1.3.5-1.3.6 The purpose of interfacing elements, such as buttons or input forms, is programmatically defined.
    - 1.4 Content is visually readable and not jarring to look at
    - 2.1 Content must be transversable using keyboard inputs and keyboard navigation software only
    - 2.4 Content is quickly traversable using keyboard navigation utilizing a table of contents, page titles, programmatic grouping of content with labels, and skippable blocks of common content
    - 2.5 Content can be interacted with using single pointer controls, and interactable content is labeled correctly
    - 3.1 Text content can be parsed by a language translator and be understandable in the non-default language (i.e text and words are simple enough to likely have defined translations, or if not the word/phrase is context specific)
    - 3.2 Webpage format and content format is consistent and predictable, even when users change UI settings of their browser or are using alternative navigation methods
    - 3.3 Human input is guided by instructions and then error checked when complete, and the user is notified if their input was incorrect
    - 4.1 Content on the page can be programmatically parsed by assistive technologies, utilizing common HTML elements and attributes in correct places

6.1.2 - Package History

Ideas/notes:

- This optional requirement WILL be implemented
- For each package, upload time/date is recorded, the version stored, and which user/account uploaded it
- When a package is updated, record the time/date of the update, the new version being stored, and which user/account updated it. History of the old version will be ported to the new version.

6.1.3 - Ingest/Upload/Update/Download Requests*

Ideas/notes:

- Buttons on HTML will be linked to methods in our code
- Cloud run Functions? https://cloud.google.com/run/docs/triggering/https-request
- Use google firestore: https://cloud.google.com/workflows/docs/tutorial-callbacks-firestore

6.1.3 - Paginated Listing of all Packages*

Ideas/notes:

- From gcloud documentation: “For list commands, you can further refine your output by using the --limit flag to set a maximum number of resources to list. You can also use the --page-size flag to define the number of resources per page if the service lists output in pages. To sort, use the --sort-by flag with the relevant field to sort.”

6.1.4 - Accounts

Ideas/notes:

- Likely WILL NOT implement, maybe if time allows

6.1.5 - Account permissions

Ideas/notes:

- Likely WILL NOT implement, maybe if time allows

6.1.6 - Reset*

Ideas/notes:

- Make call to cloud to delete all buckets

6.1.7 - Github Actions*

- Continuous integration and continuous deployment will be implemented using GitHub actions.
- Will run automated tests whenever the system is pulled from the registry.
    - Every pull must receive a code review from three independent evaluators.
- Will deploy the system to Google Cloud Platform upon a successful merge.

Ideas/notes:

- Employ a third party package tester instead of writing our own testing software.
- Independent code reviewers might be our team, meaning the program should alert each reviewer that a pull request has been made.

6.2 - Computational code

6.2.1 - Zipping/Unzipping*

- take zipped files as input, representing a npm package*
    - Files may contain multiple packages in a group together
    - Packages will be given ratings by the project 1 implementation given

6.2.2 - Version parsing*

- **Packages will also be given a version dependency score (the fraction of its dependencies that are pinned to the minor version specified compared to other versions)**
- Version notation will be:
    - Major.Minor.Patch
    - Exact: “1.2.3”
    - Bounded range: “1.2.3-2.1.0”
    - Tilde and Carat ranges: “~1.2.0” or “^1.2.0”.

Ideas/notes:

- Implement something similar to node-semvar for only the needed parsing and then specify the uploaded version of the package on the registry programmatically, so that later the update command can compare and distinguish between versions.

6.2.3 - Ingesting*

Ideas/notes:

- Will check the score of the repo. If it is greater than 0.5 then it can be accepted into the database. The database only contains “ingestible” repositories
- If the repository is not ingestible, the program will print an error message

6.2.4 - Updating*

Ideas/notes:

- An update will consist of first ingesting the new version of the package, and if ingestible, copying the previous version’s history and uploading, then concluding by removing the previous version
- Should updates allow both downgrading and upgrading in versions if they are ingestible?

6.2.5 - Debloating

Ideas/notes:

- MAYBE implemented depending on how input is given to the system

6.2.6 - Security

Ideas/notes:

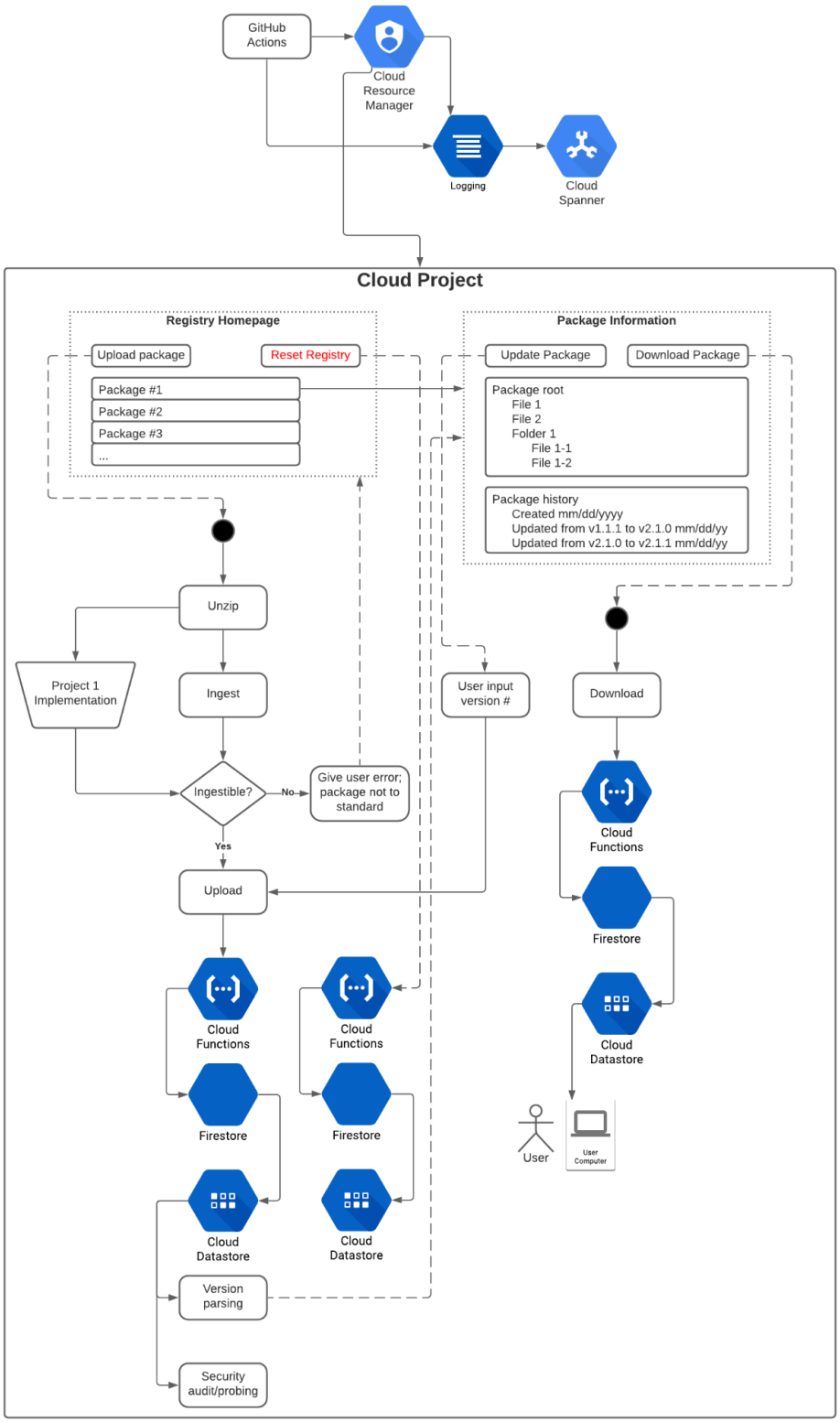
- This optional requirement WILL be implemented
- A security audit can be performed on a package utilising npm-audit
- Security probing will be performed using the RESTler tool
- A STRIDE analysis security case will be prepared to demonstrate minimization of security risk*

6.2.7 - Logging/Analytics

Ideas/notes:

- This optional requirement WILL be implemented
- Run a test with 10,000 clients and 50,000 distinct packages, where all clients are trying to download the lodash package at the same time. Record the mean, median, and 99th percentile latency of this test.
    - Use cloud spanner: https://cloud.google.com/spanner/docs/latency
- Run tests for each command or step a package can go through, both good and bad packages
    - Use restler fuzzing?
- The system will be tested with the openAPI Swagger*

6.3 - Diagrams


GitHub Actions
Cloud Resource Manager
Logging
Cloud Spanner
Cloud Project
Registry Homepage
Upload package
Reset Registry
Package #1
Package #2
Package #3
...
Package Information
Update Package
Download Package
Package root
File 1
File 2
Folder 1
File 1-1
File 1-2
Package history
Created mm/dd/yyyy
Updated from v1.1.1 to v2.1.0 mm/dd/yy
Updated from v2.1.0 to v2.1.1 mm/dd/yy
Unzip
Project 1 Implementation
Ingest
Ingestible?
No
Give user error; package not to standard
Yes
User input version #
Download
Upload
Cloud Functions
Firestore
Cloud Datastore
Cloud Functions
Firestore
Cloud Datastore
Cloud Functions
Firestore
Cloud Datastore
User
User Computer
Version parsing
Security audit/probing

7 - Planned Milestones

7.1 - Week 3 milestone

- Ingestion (Ashwin: 6 hours)
- Zipping/unzipping (Parker: 15 hours)
- Version parsing (Katie: 10 hours)
- Continue researching Cloud documentation

7.2 - Week 4 milestone

- HTML base and requests (Katie: 20 hours)
- Github Actions (Parker: 15 hours)
- Updating (Ashwin: 15 hours)

7.3 - Week 5 milestone

- Paginated listing (Katie: 10 hours)
- Reset (Ashwin: 10 hours)
- Delivery 1: Demo some working functionality

7.4 - Week 6 milestone

- Package History (Katie: 10 hours)
- Security (Parker: 20 hours)

7.5 - Week 7 milestone

- Logging (Ashwin: 20 hours)
- Final ADA checks (Katie: 10 hours)

7.6 - Week 8 milestone

- Delivery 2: final delivery

8 - Validation and Assessment Plan

We will use the requirements described earlier in this document along with running the auto-grader tests to validate the final product. Part of this process will be using Swagger to run tests on the program. Furthermore, the RESTler API will be used to perform automated security probing and test fuzzing.