Project Part #2: A Trustworthy Module Registry

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

(On group submissions, have each team member type their name).

Type or sign your names: _____

Write today's date: _____

Assignment Goal

This assignment provides an opportunity for you to work as a team on a still-small-but-more-challenging software engineering project. It will introduce you to several new challenges, including: planning for a bigger project; refactoring and extending an existing system; deploying and monitoring on a modern Cloud platform ("DevOps skills"); and performance and cybersecurity considerations.

WARNING:

1. This project spec is designed to be far more work than I think you can do in the time allowed. In your project plan, your team **must** identify and justify a reduced scope.
    a. All teams should complete the requirements marked "*baseline*". This set of requirements is relatively short; proposals to complete *only* this set will need to be carefully justified, preferably well in advance, to persuade me that you are not trying to inflate your contract. If I had to choose, I would rather have you under-promise and over-deliver, than over-promise and under-deliver.
    b. Among the other requirements, choose ones of interest to your group.

In this project you have slightly more constraints.

- You must use multiple components from Google Cloud Platform (GCP). Details below.
- You must start with an existing implementation of Project 1 – albeit not your own. You are allowed to maintain, refactor, and evolve this implementation.
- You must continue to keep at least 30% of the implementation in the language selected by the previous team.
- This is a longer project. One modest mid-project milestone (Delivery 1) has been defined for you to ensure that your team has begun work. Although some elements in the mid-project delivery have been defined, achieving those elements alone should not be too challenging. Your plan should include the delivery of some other aspects of the design by then as well.

This project has a longer timeline than Project 1. It will run for a total of 9 weeks of class, 10 weeks if you count Spring break.

Relevant Course Outcomes

A student who successfully completes this assignment will have demonstrated skill in all learning outcomes of the course.

Resources

The following resources and links will help you understand and complete this assignment.

- **REST APIs**

  - Fielding's 2000 dissertation: You can start at Chapter 5 ("REST") but the whole thing is eminently readable and edifiying.
  - 20 years later, brief commentary on what Fielding meant vs. what REST means in practice (and conjectures about why).
  - REST vs CRUD
  - Specifying REST APIs through OpenAPI (formerly known as Swagger): https://swagger.io/specification/
  - Testing REST APIs: https://github.com/microsoft/restler-fuzzer
- **Google Cloud Platform**
  - Here's the starting point: https://cloud.google.com/docs/overview
- **OWASP Top 10**
  - Some common ways that engineers introduce cybersecurity vulnerabilities into the systems they develop: https://owasp.org/www-project-top-ten/

Assignment

Introduction

The bad news: ACME Corporation rejected your company's deliverable. They decided to proceed with another company's offering.

The worse news: Your company was gambling on a lucrative contract with ACME Corporation. Your company is now out of business, and you are out of a job.

Good news: Your competitor, Beta Software Solutions (BSS), won the longer-term ACME Corporation contract. They are awash in cash and hiring software engineers to work on the next phase of the contract. Thanks to your domain expertise, you and your team have been hired for this phase. You will need to extend BSS's winning implementation to ACME Corporation's new requirements.[1] As needed, your extension may (1) integrate entirely missing components using your previous company's implementation; or (2) fix defects, refactor, or overhaul existing components as necessary, but *without* copying code from your previous company's implementation.

ACME Corporation is continuing to expand its Node.js footprint. They are really benefiting from (your erstwhile competitor's, now employer's) tool for identifying trustworthy npm modules. However, they find themselves dissatisfied with aspects of the npm registry and their process for interacting with it:

- npm contains many untrustworthy or irrelevant modules, which pollute the search results.

[1] It is possible that their "winning" implementation did not win on *technical* grounds, but rather on business considerations. Those factors might include pricing or trust in their company due to a long-term relationship. You should review their implementation carefully before you proceed, and you may flag weaknesses and opportunities for improvement (e.g. if their code crashes, it will negatively affect the service you build around it).

- ACME Corporation would like to put its own modules – which are trusted, of course – into the same place that third-party modules are stored, but it does not want to share those modules publicly. It seems reputationally hazardous to do so without careful consideration.
- Some of ACME Corporation's component re-use review processes involve a time-consuming manual inspection, and ACME Corporation doesn't have a good system in place for tracking which modules have been vetted. Npm does not support a good way to distinguish vetted from unvetted modules, except via some kind of honor system.
- ACME Corporation's engineers download many modules, and npm can take a while to serve the desired content.

For these reasons, ACME Corporation would like your team at BSS to develop a custom registry for their npm modules.[2]

Sarah is still your contact person at ACME Corporation. Here are ACME Corporation's desired requirements, in her words.

Sarah's desired requirements

Functional requirements

Your system must support the following general behaviors. Some behaviors are detailed later on, and others are introduced under an appropriate heading.

- Upload, update, rate, and download packages represented as zipped files.
    - Should work for individual packages.
    - Should have support for doing so on package groups, e.g., so that multiple related packages can be updated atomically. Support a three-stage transaction – (a) initiate an empty request; (b) append upload commands to the request, one at a time; and (c) execute the request.
    - The "rate" option should provide the net score and sub-scores from Part 1, to inform prospective users of the quality of the module. It should also include **two new metrics:**
        - The fraction of dependencies that are pinned to at least a specific major+minor version, *e.g.*, version 2.3.X of a package. (If there are zero dependencies, they should receive a 1.0 rating. If there are two

[2] It might be cheaper for your company to subcontract some of the work (e.g. buy an existing system and configure it for your customer). In this vein, you looked for any companies that currently offer registry-as-a-server. As it turns out, npm itself formerly sold this product ("npm enterprise") before npm was purchased by GitHub in 2020. Now this product is sold under the name "GitHub Packages" and includes other module registries as well. You recall, of course, that GitHub is itself owned by Microsoft. Since Microsoft is a competitor of ACME Corporation, ACME Corporation does not want to leave its private software in their clutches. It would be possible to construct the registry that your customer desires by interfacing with GitHub Enterprise/Packages, but your contract forbids this solution.

dependencies, one pinned to this degree, then they should receive a ½ = 0.5 rating).
        - The fraction of project code that was introduced through pull requests with a code review.
    - Although ACME Corporation may upload its own internal modules to this registry, any such modules will be organized in the style of an npm package.
- Request the ingestion of a public npm package. To be ingestible, the package must score at least 0.5 on each of the metrics indicated in Part 1. If ingestible, the command should proceed to a package upload. You should add any metrics that are missing in BSS's solution.
- Fetch history for an individual package, including on a particular set of its versions. Support:
    - Exact: "1.2.3"
    - Bounded range: "1.2.3-2.1.0"
    - Tilde and Carat ranges following the definition here. "~1.2.0" or "^1.2.0".
- Fetch a directory of all packages. Consider that this query may involve an enormous amount of data, e.g., if the registry has millions of packages in it. Choose a design that won't become a denial-of-service vector.
- Search for a package using a regular expression over package names and READMEs. The results of a package search should resemble the "directory" view, but a subset thereof.
- Track the popularity of packages by downloads and stars. This feature should contain design elements to reduce attempts to artificially inflate a package's popularity. Popularity information should be included in any search results or directory information for packages.
- The size cost of introducing a package – both directly and via its transitive dependencies, measured in terms of the size of zip files – should be communicated to potential users. Users may wish to query the size cost of multiple packages at once, because those packages might rely on a set of shared dependencies such that the transitive cost of their dependencies is not much higher than the cost of just one of them.
- To reduce storage costs, the "upload" and "update" features should support a "debloat" option that removes unnecessary "bloat" from the package.
- Reset to the default system state (an empty registry with the default user)

Your system should be accessible via a REST-ful API. (HTTP verbs with conventional meanings.)

Your system should be accessible via a pleasant and web browser interface that is compliant with the Americans with Disabilities Act (ADA)[3]. Use a front-end framework of your choice. Automate your tests with Selenium.

[3] According to the US Department of Justice, Civil Rights division, a website that is compliant with the Web Content Accessibility Guidelines (WCAG) will meet the requirements of the ADA (ADA guidance from the DoJ). Your website should be compliant with WCAG 2.1 at level AA. See https://www.w3.org/TR/WCAG21/ for details. These free tools from Microsoft may be helpful.

Baseline functional requirement

You must support, via a REST-ful API:

- Upload, update, rate, and download individual packages.
- Support for the new metrics requested by the customer.
- Ingestion of a public npm package as described
- Package search
- Directory of all packages
- Reset to default system state

You must provide an ADA-compliant web browser interface. It need not be pleasant.

Non-functional requirements

Performance requirements

Your system should support concurrent interactions with up to 500 clients.

ACME Corporation wants to know the mean, median, and 99th percentile latency when 500 clients are all trying to download the lodash package, from a registry containing 1,000 distinct packages. Provide measurements.

Traceability requirements

ACME Corporation is concerned about untrustworthy modules entering the registry. They would like to be able to trace the history of a module within the registry – who uploaded each version when?

ACME Corporation is concerned about the downloading of sensitive modules.

- They would like to execute an arbitrary JavaScript program prior to the download of any sensitive modules. Each such sensitive module can have zero or one JavaScript program associated with it. Any user can upload and delete sensitive modules and provide or query the associated JavaScript monitoring program. This program may need to communicate with ACME Corporation's audit servers.
- This JavaScript program expects to run under Node.js v18, and accepts five command line arguments: "MODULE_NAME MODULE_VERSION UPLOADER_USERNAME DOWNLOADER_USERNAME ZIP_FILE_PATH". If the program exits with a non-zero code, the download of the module should be rejected with an appropriate error message that includes the stdout from the program.
- They would also like your system to be able to return the download history of the sensitive modules – which accounts downloaded them at what times?

Security requirements

ACME Corporation would like to restrict access. Your system should support:

- Register, authenticate, and remove users with distinct “upload”, “search”, and “download” permissions.
- The system should permit authentication using a combination of username + secure password, and yield a token to be supplied to the other APIs as a payload parameter.
- This token should remain valid for 1000 API interactions or 10 hours.
- Some users should have an “admin” permission. Only administrators can register users. Users can delete their own accounts.
- Some teams at ACME Corporation have experimental code that should not be accessible by other members of the company. There should be a “user group” mechanism that can be specified at user creation time. During upload, a package can be marked as “secret”. Such a package can only be queried by a member of the group that uploaded it.

You must persuade ACME Corporation that your system does not expose them to substantial security risks. To this end, you should:

- *Justify:* Prepare a security case based on a STRIDE analysis of your system.
- *Demonstrate:* Carry out automated security probing of your system using the RESTler tool. Document any detected vulnerabilities. Indicate how they came to be in your system (perhaps with a “Five Whys” analysis). Repair them.

ACME Corporation would like to be able to conduct security audits of packages in the registry. They should be able to request the audit of one package, a group of packages, or the entire registry. As a default, they would like to use the “audit” mechanism provided by npm. They would also like to be able to run audits using an arbitrary JavaScript program. This JavaScript program expects to run under Node.js v18, and accepts one command line argument: “ZIP_FILE_PATH AUDIT_FILE_DIR”. Audit results will be written to a file in AUDIT_FILE_DIR. The result for all audited packages should be displayed. For any package on which the program exits with a non-zero code, the results of the audit program should be available.

Baseline requirement

All teams must prepare a security case.

All teams must employ RESTler to scan their system, and document the outcome. If RESTler misbehaves, open issue(s) and indicate how Microsoft resolves them. (But please do due diligence first. Lazy issues will reflect poorly on Purdue).

System deployment and costs

ACME Corporation is a big company, so for scalability reasons your system should be deployed on Google Cloud Platform (GCP). You must expose a single uniform API, but under the hood you may use multiple GCP components. These should be indicated in your project design.

Here are some suggestions depending on your need:

Need	**Some relevant GCP products**
Compute	Compute Engine (VMs); Kubernetes Engine (containers)
Storage	Storage (blobs in buckets); Bigtable (key-value); Firestore (NoSQL); MySQL (relational)
Auto-scaling	"Managed group instance" within compute engine; Pods in Kubernetes

ACME Corporation has provided GCP credits for you to develop and test your system. Work out your expected costs in advance using a spreadsheet or a GCP tool. Evaluate on small resources where possible. If you exhaust the credits, please contact Professor Davis.

Your system source code must reside in a single GitHub repository.

ACME Corporation places a high value on CI/CD. By Deliverable 1, your system should be using GitHub Actions to perform:

- Automated tests (CI/Continuous Integration) on any pull request
    - You might conduct some tests (e.g. end-to-end performance tests with many clients) outside of your automated test pipeline.
- Automated service deployment to GCP on successful merge (CD/Continuous Deployment)

For consistent quality, ACME Corporation requests that every pull request receive a code review from at least three independent evaluators.

Baseline requirement

- All teams must use at least one GCP component – e.g. a VM with storage attached, hosted on Google Compute Engine.[4]
- In their mid-project milestone (Delivery 1), all teams should include screenshots demonstrating the use of CI/CD via GitHub Actions.
- Introduce new behaviors using pull requests, with code review by at least one teammate. Provide evidence of this in Delivery 1 and the final report.

[4] A "one fat VM" configuration would not be scalable and thus could not sustain a high load. However, it would be relatively straightforward to configure, so it might be useful for an initial prototype.

Internal (Baseline) Requirements

Auto-grader API

To facilitate automatic grading, the project material includes an OpenAPI ("Swagger") specification for the "baseline" components. You will also use this specification to test your system; see the *RESTler* link under the Resources heading.

The project requirements contain changes that are not yet reflected in the OpenAPI spec. The course staff will provide an update by March 15. This should not be a blocker for you.

We will not attempt to deploy your system ourselves. Instead, you will provide us with a URI (hostname + port) that we can use to interact with a clean version of your system.

In its initial and "Reset" state, your system must have a default user. If your team chooses to support authentication, the username and password are given next. If you support user creation/deletion, groups, etc., then this user should have admin privileges.

- Username: *ece30861defaultadminuser*
- Password: *correcthorsebatterystaple123(!__+@**(A'"`*;DROP TABLE packages;
    - Please consider why this password looks so strange, and make sure you handle user input carefully.

Source code hosting

Per the academic honesty requirements of this course, you are not permitted to compare software implementations with other teams. I encourage you, however, to discuss "Operations" issues such as GCP configuration with one another and on Piazza.

Project management

I recommend but do not require that you use GitHub Project Boards for progress tracking. You **must**, however, use some project management software (e.g. Trello, Monday, GitHub Projects, etc.). Use reports from that software in your weekly milestones.

Software re-use

You are allowed to re-use existing software to support your implementation, either as tools (e.g. VSCode; git; GitHub; TravisCI) or as components in your implementation (e.g. a module to help you parse command-line arguments). You should include a justification of any components you choose to re-use – how will you decide whether they are trustworthy? (discuss in the Project Plan) and how did that assessment work out in practice? (discuss in the Project Postmortem)

You are allowed to re-use code snippets from software engineering resources such as Stack Overflow. You must provide a citation (web URL is fine) to the relevant post. Also, please review Prof. Davis's general perspective on Stack Overflow. Technically, Stack Overflow snippets are themselves governed by a software license, but you are not trying to distribute your project code and I do not think anyone would seriously pursue litigation along these lines.

You are **not** allowed to copy-paste code snippets out of an open-source project – this is a great way to expose ACME Corporation (and your future employer in the real-world) to lawsuits. Any re-use of this nature must use existing module APIs2and/or extend those APIs so that you can access the logic you want.

Timeline and Deliverables

The project will be completed over a 10-week period. At the end of…

- Weeks 1-2: Assessment of BSS's codebase; learning, planning, estimating, negotiation, and initial design. An initial plan is due on Sunday Feb 26 and an update might be appropriate on 5 March.
- Weeks 2-5: Internal milestones (Spring break occurs during this period).
- Week 6: *Delivery 1: Demo of some functionality* (some baseline requirements like CI/CD. Other aspects are up to you).
- Weeks 7-9: Internal milestones.
- Week 10: *Delivery 2: Deliver the rest of the functionality*
- Week 11: Postmortem

Weeks 1-2: Planning and getting started

Submit a Project Plan Document (Word Doc or PDF) including the following. Some of this can be copied from Part 1, but please review and edit as appropriate.

- Tool selection and preparation
    - Programming language, toolset, component selection [linter? Git-hooks? CI? Testing framework? Logging library?]
    - Communication mechanism [Slack? Teams? Email?]
- Team contract
    - For example, your team might agree: do the work they take on, to document their code, testing rules, style guide, timeliness expectations
- Team synchronous meeting times
    - I recommend at least one (short) mid-week sync to discuss issues, and one end-of-week sync to put together your weekly reports.
- Requirements
    - A refined and organized list of requirements, based on Sarah's description and specification.
- Preliminary design
    - Diagrams to support planning

        - Activity diagrams to depict the various activities performed by your system.
        - Data flow diagrams to depict the flow of information (including who can access which information, which you will want for the STRIDE analysis)
        - (Photos of a whiteboard are fine, or online UML tools like those provided by LucidChart)
- Timeline and planned milestones for weeks 2-10
    - Each milestone should list the features, sub-tasks, the owners of those tasks, the estimated time to complete it, and how success will be measured.
    - Any communication requirements between tasks should be noted, e.g. "Jason and Tahani need to discuss the interface involved between task A and task B."
- Validation and Assessment plan
    - What is your plan to assess whether the delivered software satisfies Sarah's requirements? What behaviors will you check? What performance metrics (if any) will you apply?
    - Many teams remarked on Part 1 that they left validation until the end and ran into trouble. Do you want to try something different on Part 2?
- Starting project analysis
    - You **must** use another team's Project 1 implementation. This implementation may (a) not work correctly; and/or (b) may use languages and tools with which your team is unfamiliar. You should review this implementation and answer questions such as: the cost (if any) of integrating it into your design; the interface you plan to use to talk to it (*e.g.*, continuing in the same language vs. wrapping it with an interface in your preferred language); the changes you plan to make to it; and so on.
    - You should conduct an independent assessment of how trustworthy the project is (*e.g.*, Does it meet requirements? Does it have a test suite?), and list this in your document.

Weeks 3-9: Internal milestones

Each week, submit a report with your updated list of milestones, tasks, etc. representing completion and the actual time spent by each team member on the project.

This report should be self-contained, *e.g.*, including the relevant information from the original plan. For example, you should use a table for each feature and track the status of the relevant tasks.

If you *deviate substantially* from your timeline, consider attending one of the course staff office hours to discuss the deviation.

Week 10: Deliver

Submit the software itself, along with brief report describing the status of the software in relation to Sarah's requirements and specification.

Week 11: Postmortem

Deliver a project postmortem report. This report should reflect on each aspect of your Plan (from week 1) compared to your Execution. What went well? What went poorly? Where did your time estimates fail? When and why did you deviate from your Plan? For all of these questions, try to answer the question "Why?"

Grading rubric

Points breakdown:

- 35% Design & Planning document + Milestone documents.
- 50% Working delivery, broken down as:
    - 25% "Our auto-grader can interact with it in the "baseline" features"
    - 15% "The additional promised features are delivered with evidence that they are working correctly"
    - 10% "Per our manual inspection, the software follows reasonable-looking engineering practices, e.g. good file/class/variable names, consistent style, choice of data structures, use of patterns to isolate what is changing, appropriate pipeline for automated deployment, etc."
- 15% Post-mortem.

Provided that the teammates complete the tasks, they were assigned as part of the project plan, all team members will receive the same grades. If there is an issue with teamwork, please raise it with Prof. Davis as early as possible.

- Your team's milestones should allow you to observe problems with forward progress.
- For personality clashes etc., use your judgment to determine if you want to speak with Prof. Davis.

ACME Corporation's Budget is not Bottomless

Sarah reminds you that your team members are from an independent contracting firm. She says the company is **willing to pay your team for up to 90 hours per person for this project**[5], and would rather see ***something that works – at least partially! – by the deadline.***

- Your project plan and your weekly progress updates should reflect an appropriate amount of time for the project, e.g. at least 5 hours per team member per week. If you wait until the last minute, Sarah will be nervous, pull the plug on the project, and might break off future contracts with your company.

[5] "90 hours per person for this project" – Since the project will run for 10 class weeks, that will average out to 9 hours per teammate per week. The postmortem week should be lighter and the earlier weeks a little heavier.

- If you begin to deviate from your planned timeline, you should submit a **revised** plan as part of a weekly update. That way Sarah can keep management abreast of progress and aware of any changes in the functionality that will be delivered.
- You should plan your project in such a way that you can deliver incremental value to Sarah even if you cannot complete all of her requirements.
    - Recall the aircraft requirements document from the Requirements Engineering unit – one of the final chapters designated useful subcomponents that the vendor could deliver.
    - One of the slides from class has an excerpt.
- I will be amazed if any team were able to complete the full set of functionality by the deadline. **In your design document, you should have enumerated and organized the requirements, estimated the cost of each feature, and identified the subset of the functionality that you think your team can reasonably deliver. The total desired cost for your team should be roughly 90 hours per person. Estimate and negotiate accordingly.**

Guidance

You can complete this project with a "Cloud-First" approach – all code is always deployed on the Cloud. You might prefer, however, to have one teammate study the GCP options while others start the implementation locally. Likewise, there are tools to test GitHub Actions locally to reduce debugging time on those.

When offering a web service such as the Trustworthy Module Registry, *some* component of your system must handle incoming requests. After this point, however, these requests are handled through a sequence of function calls and interactions with internal components. If you prefer, you can focus on the entities *behind* the interface first, and add the "web interface" part later.

Initially, scaffolding and mocking can be used in place of components. This helps you focus on inter-component interfaces first, and facilitates testing each component in isolation. At deployment time you can replace a mock with the real component.