**Project 2 rubric: Delivery**

For this grading, the following definitions hold

Ingestible package – Package with scores above 0.5 for each of the trustworthiness metrics. I will select 1 from popular NPM packages (such as Lodash, react, etc) that fits this criterion.

**Baseline Requirements: 25 points**

|  |  |  |
| --- | --- | --- |
| Component | Points | Rubric |
| Upload Package | 3 | * 1 point: Given a valid request with a valid base64 string in the content field, package is created successfully, and valid response (200) returned. * 1 point: Given a request with empty content, package creation fails, and a valid response (400) is returned * 1 point: Attempt to create a package that already exists fails and a valid response (403) is returned. |
| Ingest Package | 2 | * 1 point: Given a request with the URL field pointing to an ingestible package, package is successfully uploaded. * 1 point: Given a request with the URL field pointing to an uningestible package (invalid license, no unit tests and only one maintainer), package upload fails and a valid response (400) is returned. |
| Update Package | 3 | * 1 point: Given valid request for a package that already exists, and content field containing valid base64 string, package update is successful, and a valid response is returned. * 1 point: Given valid request for a package that already exists, and the URL field pointing to an ingestible package, package update is successful, and a valid response is returned. * 1 point: Given request to update a package that doesn’t exist, package update fails, and a valid response (400) is returned. |
| Rate Package | 3 | * 2 points: Given a valid package ID, endpoint returns the complete metric scores for the package. * 1 point: Given a package ID that doesn’t exist, endpoint returns a valid error response code (400). |
| Dependency Metric | 2 | * 1 point: There is the presence of a dependency metric in the call to rate package. * 1 point: Gives a correct dependency score to a package that reflects the ratio of its dependencies pinned to a specific major+minor version. |
| Download Package | 2 | * 1 point: Given a valid package ID, returns the meta data and content for the package * 1 point: Given an invalid package ID, returns a valid error response |
| Get List of Packages | 2 | * 1 point: Given a package name as query parameter, returns the package with that name * 1 point: Given a package name that doesn’t exist as query parameter, returns an empty list |
| Reset to Default State | 1 | * 0.5 point: Call to endpoint returns a successful response * 0.5 point: After the endpoint is called, subsequent call to get list of packages should return an empty list |
| Test Evidence | 3 | * 3 points: Provided evidence that they performed manual or automated tests on all their reported features or provided justification while some features were untested.   + Partial points if the evidence provided is not exhaustive. |
|  |  |  |

# **Additional Features: 15 points**

|  |  |  |
| --- | --- | --- |
| STRIDE Analysis | 5 | * 3 points: Provided evidence that they conducted STRIDE analysis on their project. This includes providing an analysis of their system against each of the STRIDE metrics. * 2 points: Identifies at least, 2 threats to their systems and discusses ways they mitigate against such threats. |
| Updated Open API Specification | 4 | * 2 points: Provides an updated Open API specification that reflects the final state of their submitted project. * 2 points: Discusses changes made to the Open API specifications and provides justifications for their decisions |
| Additional features | 6 | * 3 points: Provides a documentation of additional features (at least one additional feature) completed and screenshots that showed working behavior. * 3 points: Provides a comparison of features they planned to complete and the features they completed. Provides adequate justifications why they couldn’t complete any specific feature. |

# **Manual Inspection for Engineering Practice: 10 points**

|  |  |  |
| --- | --- | --- |
| Documentation | 3 | * 2 point: Has a README that explains the purpose of the project, including, e.g., how to configure it, deploy it, and interact with it. * 1 point: Three randomly chosen files all have introductory remarks at the top and some semblance of JavaDoc/PyDoc/etc. throughout |
| Effective communication | 2 | * Look at three randomly chosen files   + 1 point: Are the names of variables and functions “pretty good”   + 1 point: Is the style consistent across these files? |
| Appropriate CI/CD | 3 | * 1 point: Presence of YAML/etc. files encoding the CI/CD pipeline * 1 point: Successful implementation of CI (linting, testing, etc.) * 1 point: Successful implementation of CD (Automatic deployment of application to GCP. Full credit if much of this process is automated, and the rest is documented. Some manual steps, e.g. distributing tokens, are OK as long as they are documented) |
| Teamwork | 2 | * Each teammate contributed at least 100 lines of the code according to a simple “sum the lines/authors via git blame” check. |

# **Bonus Points: 10 points**

|  |  |  |
| --- | --- | --- |
| Restler | 10 | * 3 points: Provided a script that enabled us run Restler against their service. * 2 points: Provided a line coverage report showing % lines covered by their automated tests. * 2 points: Provided a line coverage report showing % lines covered by Restler. * 3 points: Described up to 5 bugs found by Restler but not their automated tests. |

# **Advanced**

|  |  |  |
| --- | --- | --- |
| Scalability |  | * Run load testing scenario. 1000 clients interacting with 1000 different packages. No crash. |
| Pathological |  | * Concurrent create/update to the same package from 10 clients completed successfully. No crash. |