**Project 2 final deliverable -- Report**

*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**

In this assignment, you will deliver your Project 2 implementation (*software*) and communicate the final status of your Project 2 (*report*).

This document provides a template for the report. This is a form of documentation that your customer can use to decide (a) whether you’ve met the contract, and (b) re-negotiate the contract based on deviations therefrom.

# **Relevant Course Outcomes**

A student who successfully completes this assignment will have demonstrated the ability to

* *Outcome ii*: the ability to conduct key elements of the software engineering process, including…deployment
* *Outcome iii*: Develop an understanding of the social aspects of software engineering… including…communication [and] teamwork.

# **Assignment**

Fill out each of the following sections.

## Location of project

Provide a URL that we can use to interact with your team’s deployed service:

|  |
| --- |
|  |

Provide a link to your team’s code repository on GitHub:

|  |
| --- |
|  |

## Succinct description

In a 5-7 sentence paragraph, describe the system that you have implemented.

|  |
| --- |
|  |

## Functional requirements

### Baseline metric

You were required to implement a new metric to compute the degree of version pinning in a package. This implementation needed to build on another team’s Project 1 implementation.

Describe any changes you made to the existing Project 1 design or implementation,[[1]](#footnote-1) divided into two kinds:

1. Changes to allow you to implement the new metric

|  |
| --- |
| **Change 1**: XXX  **Justification**: XXX  **Change 2**: … |

1. Changes to improve the reliability of the component so that your Project 2 implementation would satisfy the customer’s requirements.

|  |
| --- |
| **Change 1**: XXX  **Justification**: XXX  **Change 2**: … |

### Baseline API

In your *Project 2 Plan* document, you described the system features and requirements you planned to implement. Here you will describe how things went.

Fill in the following table for each of the *baseline behavioral features* (e.g. “ingest a package”) and the degree to which you’ve met each of them. Make one copy of the table per feature.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature:** …  **Relevant endpoint(s):** …  **How completely is it implemented?** *(If partial, explain limitations)*  **How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*   |  |  |  | | --- | --- | --- | | *Endpoint | Verb(s) | Payload option(s)* | *Validation approach(es)\** | *Test records\*\** | | X/Y | GET | Valid input |  |  | | X/Y | POST | Valid input |  |  | | … |  |  |   \*Validation approaches: For example, (None ; Manual ; Automated unit tests ; Automated end-to-end tests)  \*\*Test records: For example, link to the relevant tests in your repository; link to the most recent relevant run of your CI; write the date of the most recent manual test. |

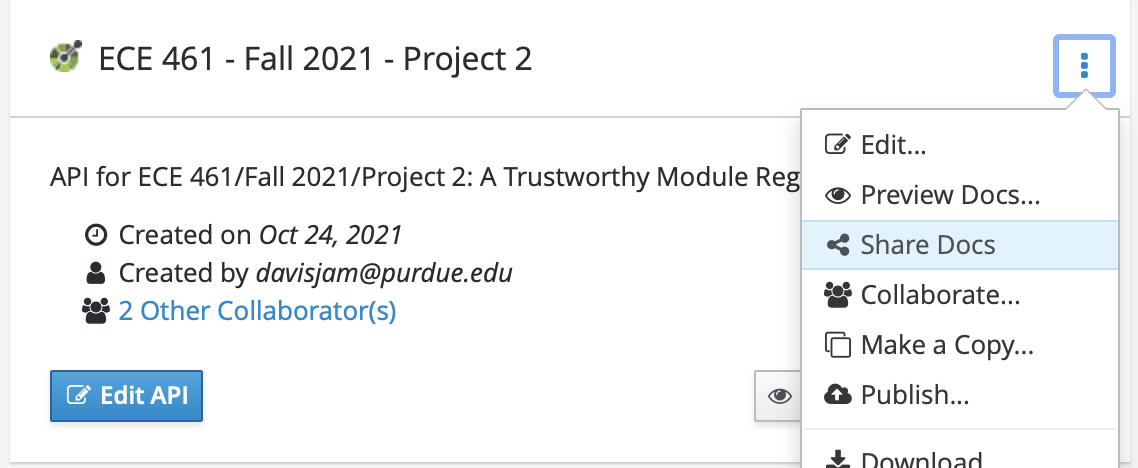
### Non-baseline: API

If you implemented additional behaviors in your team’s web API, (e.g. package groups; traceability features; security features) they should be reflected in your team’s OpenAPI specification on GitHub.

Provide a link to your team’s OpenAPI specification:

|  |
| --- |
|  |

(If possible, please also include a link to an instance on apicur.io page to save us some time in grading! Here’s how to do that: )



Fill out the following template for each additional feature:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature:** …  **Relevant endpoint(s) and input/output type(s):** …  **Summary of design:**  *Summarize your design of this feature. I suggest 5-15 sentences and 1-2 diagrams. At least one diagram is required – pictures of a whiteboard are fine.*  **How completely is it implemented?** *(If partial, explain limitations)*  **How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*   |  |  |  | | --- | --- | --- | | *Endpoint | Verb(s) | Payload option(s)* | *Validation approach(es)\** | *Test records\*\** | | X/Y | GET | Valid input |  |  | | X/Y | POST | Valid input |  |  | | … |  |  |   \*Validation approaches: For example, (None ; Manual ; Automated unit tests ; Automated end-to-end tests)  \*\*Test records: For example, link to the relevant tests in your repository; link to the most recent relevant run of your CI; write the date of the most recent manual test. |

### Non-baseline: Browser-based interface

If you implemented a browser-based web interface, it should be ADA-compliant (WCAG 2.1 at level AA).

1. Show the result of the automated tests implemented by <https://github.com/microsoft/accessibility-insights-web>, as applied to each of your web page(s). Screenshots are fine.

|  |
| --- |
| **Page 1:** *URL*  **Outcome of tests: *…***  … |

1. Describe any additional steps you took to consider accessibility (e.g. educational resources you consulted; design choices you made; implementation decisions that are not covered by Microsoft’s automated tests).

|  |
| --- |
|  |

## Non-functional requirements

### Baseline

#### Security: STRIDE analysis

The terms used in this section are defined [here](https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach).

**System model.** Present one or more data-flow diagrams of your deployed system. (A whiteboard picture is fine, but use the correct symbols please).

* You may provide multiple DFDs to capture different aspects of the system.
* You may indicate multiple trust boundaries, e.g. for different classes of users.
* Each diagram should indicate at least the following entities: data flow; data store; process; trust boundary. You may include interactors and multi-process if needed.
* Each diagram should number the entities for reference later on.

|  |
| --- |
|  |

For each trust boundary indicated, describe the nature of the untrusted party involved (e.g. “outsider threat [e.g. external hacker]” or “insider threat [e.g. ACME employee with valid credentials]” or “infrastructure provider threat [GCP]”).

|  |
| --- |
| **Trust boundary #**: 1  **Untrusted party**: …  **Trust boundary #**: 2  **Untrusted party**: …  … |

**Security requirements.** The project document defines many requirements. Identify the *security* requirements of your system, aligned with the six security properties defined by the STRIDE article. These requirements may vary by system, depending on which features you implemented. It is possible that you will not have a requirement associated with every security property.

|  |
| --- |
| **Confidentiality**   * [All systems]: Observers on the network cannot directly observe client-server interactions. * [If you implemented authentication]: Only authenticated users can retrieve packages. * …   **Integrity**   * …   **Availability**   * …   **Authentication**   * …   **Authorization**   * …   **Nonrepudiation**   * … |

Fill out this table for each [STRIDE property](https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach) (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege):

|  |
| --- |
| **Stride property**: Spoofing  **Affected security properties**: …  **Analysis of components:**   * Diagram+Component (e.g. “Diagram 1, component 3 – Database”): XXX   + Risk 1:     - Mitigations applied: …     - Degree of risk resolution: …     - Suggestions for additional mitigations, if needed:   + Risk 2:     - … * Diagram+Component: YYY   + Risk 1: …   **Stride property:** Tampering  …  **Stride property:** Repudiation  …  **Stride property:** Information disclosure   * (e.g. looking at confidentiality property #1, one threat is that there is an observer on the client-server network path. A mitigation is to use HTTPS for encryption. HTTPS would not protect against e.g. keyloggers on the client’s machine, but that threat would be out of scope for your system – a relevant additional mitigation might be “ACME Corp. employees should have anti-virus installed to check for keyloggers”).   **Stride property:** Denial of service  …  **Stride property:** Elevation of privilege   * *NB: If you implemented the “JSProgram” feature, tread carefully.* |

#### Deployment: GCP

Complete the following table. It should include all GCP components that you used.

|  |  |  |  |
| --- | --- | --- | --- |
| **Purpose** | **Selected GCP component(s)** | **Other GCP components considered** | **Justification for selected component** |
| *(e.g. “Compute” or “Storage” or “Auto-scaling” or …)* |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Deployment: CI/CD

CI/CD: Provide prose and screenshot(s) demonstrating that your team is using GitHub actions to facilitate continuous integration (e.g. by running a *linter* and a *test suite* on every pull request) and continuous deployment.

You may re-use answers from Deliverable #1 as appropriate.

|  |
| --- |
| **What steps is your team following prior to accepting a code change? (e.g. git-hooks, code review, linting, test suite, etc.)**  **Provide a link to an example in your GitHub repo where your team followed this CI process (e.g. a pull request):**  **How consistent have you been with this process? What is keeping you from full consistency?**  **What aspects of your system are being tested automatically by your CI scheme?**  **What kinds of defects might go uncaught, and how are you mitigating this risk?**  **Provide screenshots of the GitHub action file (e.g. YAML) that defines the CI stages**  **Provide screenshot(s) of the test suite in action, e.g. the reports from the various tools you have configured, as run on one of your team’s code changes.**  **Describe the extent to which you are able to “continuously deploy”. What is your team’s process to get your current prototype into a deployment on GCP?** |

### Non-baseline

#### Performance

If you considered the performance requirements:

* Provide latency details for mean, median, and 99%ile clients for the “many clients download lodash” scenario described in the project spec (or as close as you could get to that scenario).

|  |
| --- |
| XXX |

* Describe any design choices you made specific to performance (e.g. component selection; optimized paths such as caches; etc.)

|  |
| --- |
| XXX |

#### Traceability

If you implemented the traceability requirements: I think these are all functional behaviors and should appear under “Functional requirements – Non-baseline:API”.

#### Security: RESTler evaluation

**This section is optional.**

You may earn up to 5 bonus points on the whole project for using RESTler to support your validation effort. For full credit, you must provide:

* A script to run RESTler against your service and achieve at least 80% endpoint coverage and at least 40% of backend code coverage [this will require some configuration effort on your part];
* A line coverage report showing % lines covered by your handwritten automated tests;
* A line coverage report showing % lines covered by RESTler;
* A description of up to 3 bugs (if any...) found by RESTler and not your automated tests, with notes about why you think RESTler found them and how this may influence your testing practices in future; and
* A written statement that the whole team understands how RESTler works, knows what the configuration files do, and contributed to the effort (I do not want to have just one team member solo this part!)

To earn full bonus points, fill out the following table.

|  |
| --- |
| *(Demonstration: +3 points)*  **Link to the RESTler script + config directory in your repository:** …  **Link to line coverage reports (store them in your repository):**   * Coverage from handwritten automated tests: *LINK* * Coverage from a RESTler run: *LINK*   *(Bugs: up to +2 points)*  **Bug #1 discovered (or N/A):**  *A description of up to 3 bugs (if any...) found by RESTler and not your automated tests, with notes about why you think RESTler found them and how this may influence your testing practices in future*  **Bug #2 discovered (or N/A):**  *…*  **Bug #3 discovered (or N/A):**  *…*  *(Must include for any points)* **The following team members attest that they understand RESTler and its configuration, and that they contributed to the RESTler effort**:  *NAME 1*  *NAME 2*  *…* |

## Notes for the auto-grader

If your submission cannot be automatically parsed by the auto-grader described in the project specification, provide explanatory notes that the course staff can consider while scoring your submission. Be specific. Since this spec was provided well in advance, accommodating any deviations is at the discretion of the staff.

|  |  |
| --- | --- |
| **Deviation** | **Details** |
| … |  |
|  |  |
|  |  |

## Notes for the teamwork checks

In a typical team, each team member contributes a substantial amount of code. If on YOUR team, some team member contributed fewer than 100 lines of code, you may provide an explanation for us to consider.

|  |
| --- |
| **Team member**: XXX  **Important non-code contributions**: XXX  … |

1. Hint: It might be helpful to examine your team’s PRs or git logs to recall these changes. I assume, of course, that you followed an appropriate engineering process so that you can answer these questions. [↑](#footnote-ref-1)