Part 2 – Plan document

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: Erik Hays, Seth Karner, Sean Chang, Connor Barry

Write today's date: February 26, 2023

Assignment

In this document, you will describe your plan for Project 2. This document provides a template. You are welcome to add more content than is listed in the template, but hopefully this template helps you avoid missing anything crucial.

Tool selection and justification

What tools? Why those tools? (Use a table)

Technology	Justification for use
TypeScript	We are deciding to use typescript due to its very prominent use in Web Development and ability to coincide with NodeJS.
NodeJS	This is necessary as we will need to build and run our service, node js gives us the capabilities to do so.
NPM	NPM is the package manager that will allow us to install any packages we deem fit for use
Group 11's Part 1 Trustworthy Modules CLI	We will use this as it is what we will build our registry off of, it can also aid us in determining the right packages to use within this new service.
RESTler	This will scan our system and document the outcome for any security vulnerabilities.
GCP Compute Engine	This will manage the deployment of the service, using virtual machines to load balance.
GCP Storage	We have not yet decided whether we want to use cloud storage, firestore, bigtable, etc. But this will act as our database, to store modules and details about them.
Github Actions	This is how we will develop in a continuous integration/continuous deployment manner, this is necessary for automated testing as well as automated deployment.

ECE 30861 – Software Engineering

Team contract

Describe the general behavior expected of each team member.

Each team member should maintain a steady and fair workload, with constant updates within our bi-weekly meetings. There is a certain level of expected communication within the team too, such as when bugs arise in implementation, or new ideas /proposals for implementation.

Did you make changes between P1 and P2?

We did not make any changes to the structure of our team contract between P1 and P2.

Meeting plan

Meeting times?
Due to the increased workload for this project, we shall enforce a bi-weekly (twice per week) meeting requirement. One on Monday, so we understand what should be done for the week, and a second at some point later in the week to allow group members to check-in on progress for the week. Additional meetings can be scheduled as necessary.

How will you prepare for and conduct the meetings?
The first meeting of the week will have the previous week's milestone report to consult when discussing objectives and expectations for the week.
The second meeting will require each member to understand the progress that has been made, and communicate any concerns and questions that may better the ongoing project development.
Additional meetings will be called if needed. These meetings should be called for any unexpected concerns that may occur outside of set meeting times. The reason for the meeting will be specific, so the person(s) that calls the meeting should address that concern explicitly.

Did you make changes between P1 and P2?

Significant changes were made throughout P1 to our meeting plan, this will carry over to P2.

Requirements

A refined and organized list of the requirements *that you will support*, perhaps in an annotated table.

#	Requirements	Additional Comments
1	Integrate entirely missing components using	Extend BSS's winning implementation

#	Requirements	Additional Comments
	the received program implementation	to ACME Corporation's new requirements
2	System should be accessible via REST-ful API	HTTP verbs with conventional meanings
3	Upload, update, rate, and download packages represented as zipped files	The program should work for individual packages. Must support package groups using the specific three-stage-transaction mentioned. Include two new metrics relating to the number of pinned dependencies, and the fraction of the project code introduced through pull requests.
4	Verify if the inputted public npm package is ingestible, scoring a 0.5 or higher on every metric	If ingestible, the command should proceed to a package upload
5	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
6	Fetch history for an individual package	On a particular set of its versions. Exact: "1.2.3", Bounded range: "1.2.3-2.1.0"
7	Fetch a directory of all packages	Choose a design that won't become a denial-of-service vector
8	Register, authenticate, and remove users with distinct "upload", "search", and "download" permissions.	Registration: when a user wants to register, they must provide their username and password, and select which permissions are needed, from the list of: upload, search, and download. Authentication: Credentials are verified by the system, and their permission level is verified by the system and permission status is checked against the action they are trying to perform. Removal: A user's access should be

#	Requirements	Additional Comments
		revoked when removed from the system. This ensures a reasonable level of security.
9	Reset to the default system state (an empty registry with the default user)	1. Backup critical data 2. Login as admin 3. Delete all data 4. Reset user permissions 5. Verify the system is empty 6. Perform any necessary updates 7. Restart system
10	Web browser interface for system must be ADA-compliant	See recommendations: https://archive.ada.gov/pcatoolkit/chap5toolkit.htm
11	All teams must prepare a security case	1. Identify assets 2. Risk assessment 3. Security requirements 4. Develop security plan 5. Implement security controls 6. Test security measures 7. Document the case
12	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
13	All teams must use at least one GCP component	e.g. a VM with storage attached, hosted on Google Compute Engine
14	All teams should include screenshots demonstrating the use of CI/CD via GitHub Actions	In their mid-project milestone (Delivery 1)
15	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
16	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

#	Requirements	Additional Comments
17	The stakeholders will not attempt to deploy our system themselves	Instead, we will provide them with a URI (hostname + port) that they can use to interact with a clean version of our system
18	In its initial and “Reset” state, our system must have a default user	If our team chooses to support authentication, the username and password are given next. If we support user creation/deletion, groups, etc., then this user should have admin privileges.

Starting project analysis

How trustworthy is the project you are building from?

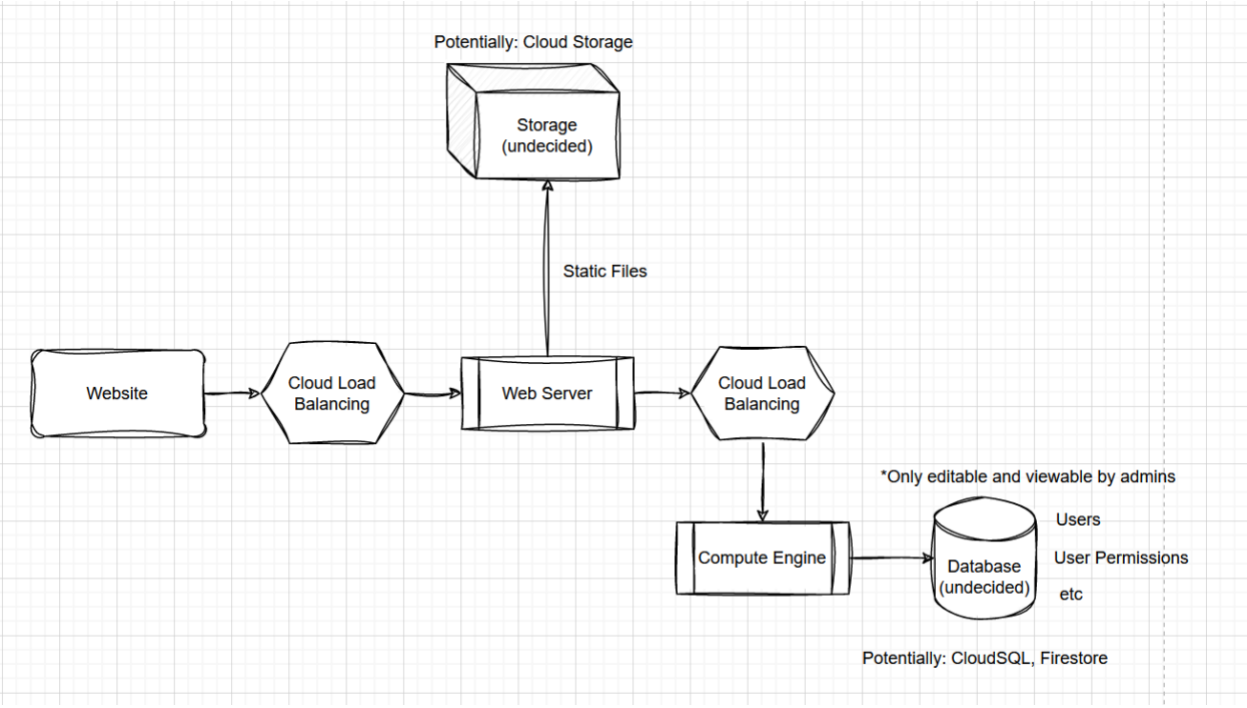
- How did you determine this?
- How did your findings affect your milestones and timeline?

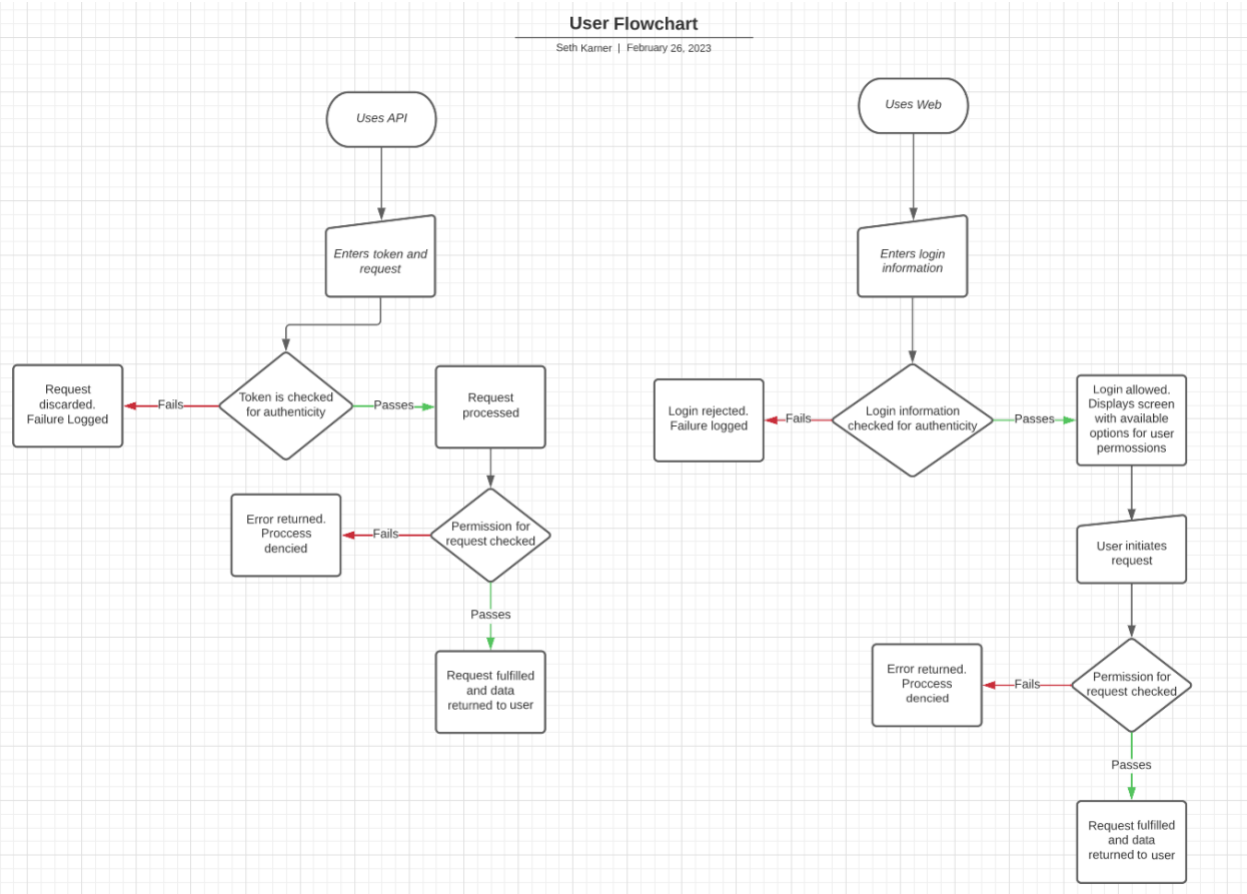
After our starting project analysis we deemed the existing service sufficient to build upon. There are a few minor flaws, such as the output to the command line was not aligned with the required output. Through communication with the team that developed the existing software, we were able to discuss flaws within the project. Another thing we need to take into consideration is the calculation of bus factor on a given module, this may require some refactoring in the calculation and could push us back a day or two on our implementation of the registry. Another issue was the output to the user, while it may not affect our implementation significantly, it is something that will require some refactoring as well. Overall, the existing software is of high quality with minimal amounts of refactoring needing to be done, resulting in an estimated week or so delay to understand the code base and initiate the refactoring stage.

Preliminary design

Prose and diagrams that describe your design. At least one diagram should label the GCP components you’ll use.

Diagrams


Potentially: Cloud Storage
Storage (undecided)
Static Files
Website
Cloud Load Balancing
Web Server
Cloud Load Balancing
*Only editable and viewable by admins
Compute Engine
Database (undecided)
Users
User Permissions
etc
Potentially: CloudSQL, Firestore


User Flowchart
Seth Kamer | February 26, 2023
Uses API
Enters token and request
Token is checked for authenticity
Fails
Request discarded. Failure Logged
Passes
Request processed
Permission for request checked
Fails
Error returned. Proccess dencied
Passes
Request fulfilled and data returned to user
Uses Web
Enters login information
Login information checked for authenticity
Fails
Login rejected. Failure logged
Passes
Login allowed. Displays screen with available options for user permossions
User initiates request
Permission for request checked
Fails
Error returned. Proccess dencied
Passes
Request fulfilled and data returned to user

Potential requests depending on user permission level: Adding a user, deleting an account, searching registry, adding to registry, etc.

Validation plan

You must have functional tests. How will you measure correctness? (Test plan or sketch thereof; measures of the test suite itself such as code coverage; etc.)
We will measure correctness by ensuring features are testing throughout development with a series of unit tests, and end to end tests of the entire component. We also plan to measure code coverage of our tests, to ensure we are not just testing portions of code that are guaranteed to work in all scenarios.

Will testing be done as a separate phase, or following a "test-as-you-go/TDD" approach?
We plan to approach testing in a test driven development fashion, this is so we can ensure that individual components are thoroughly tested before being sent through our automated testing after a pull request. This allows us to be sure that if any automated tests fail, we can, for the most part, rule out the most recent feature.

Will you incorporate logging to facilitate debugging? What is your strategy here?
We will not be incorporating logging.

If you will run performance tests, what metrics will you use?
We are not yet certain if we are going to run performance tests, but they would be extraordinarily valuable given the service we are providing. In the case we do run performance tests on our system we would measure metrics such as average wait time and load time, cpu utilization, and throughput.

Milestones and timeline

Use a table, referencing your team's requirements (or finer-grained expansions thereof). Each milestone should include the feature(s), the owner(s) of each feature, and the estimated time to complete it, and how success will be measured. (How will you know whether you are "on track"?)

- Include the total cost estimate in hours per person (it should be ~90 hours per person).

Pay special attention to the "Deliverable #1" which should be a demo of some functionality including CI/CD.

Make sure your estimates include time for learning relevant technologies, e.g. RESTler and GCP components. I also suggest the work to understand those technologies has some tangible reflection prior to incorporating into your system, e.g. "To show that person X has learned technology T, they will …".

Since this table is your justification for the features you will and will not support, it should be detailed enough that we find your estimates plausible.

Does your schedule contain enough slack for schedule overruns?

Key: *Req. #* is a reference to our requirements table.

Milestone #	Feature	Owner	Time Estimate	Success Measure
1	Req. #13, Req. 14	Connor	8-10	Stand Up VM's with Autoscaling, Have CI/CD template set up
1	Req. #1	Sean	6-8	Flaws in code are updated and system is up to date with ACMEs new requests
1	Req. #1	Seth	6-8	Flaws in code are updated and system is up to date with ACMEs new requests
1	Req. #1	Erik	6-8	Flaws in code are updated and system is up to date with ACMEs new requests
2	Req. #13, Req. #14	Connor	8-10	Stand up databases, and finish architecture including load balancing, multizone Vms, Finish CI/CD architecture (will likely be

				updated)
2	Req. #2, Req. #3, Req. #4	Sean	6-8	Created an API that can be used to access the functionality of our service
2	Req. #2, Req. #3, Req. #4	Seth	6-8	Created an API that can be used to access the functionality of our service
2	Req. #2, Req. #3, Req. #4	Erik	6-8	Created an API that can be used to access the functionality of our service
3	Req. #5, Req. #6, Req. #15	Connor	5-7	An API endpoint that can fetch history and an API endpoint for searching packages.
3	Req. #5, Req. #6, Req. #15	Sean	5-7	An API endpoint that can fetch history and an API endpoint for searching packages.
3	Req. #5, Req. #6, Req. #15	Seth	5-7	An API endpoint that can fetch history and an API endpoint for searching packages.
3	Req. #5, Req. #6, Req. #15	Erik	5-7	An API endpoint that can fetch history and an

				API endpoint for searching packages.
4	Req. #7, Req. #10	Connor	7-9	Implement a base Web interface that displays all the packages in the registry
4	Req. #7, Req. #10	Sean	7-9	Implement a base Web interface that displays all the packages in the registry
4	Req. #7, Req. #10	Seth	7-9	Implement a base Web interface that displays all the packages in the registry
4	Req. #7, Req. #10	Erik	7-9	Implement a base Web interface that displays all the packages in the registry
5	Req. #8, Req. #9	Connor	5-8	Authentication system should be implemented. Enable the system to reset to default state.
5	Req. #8, Req. #9	Sean	5-8	Authentication system should be implemented. Enable the system to reset

				to default state.
5	Req. #8, Req. #9	Seth	5-8	Authentication system should be implemented. Enable the system to reset to default state.
5	Req. #8, Req. #9	Erik	5-8	Authentication system should be implemented. Enable the system to reset to default state.
6	Req. # 11, Req. #12	Connor	6-7	RESTler outputs documentation proving our service to be secure and not at risk of attack
6	Req. # 11, Req. #12	Sean	6-7	RESTler outputs documentation proving our service to be secure and not at risk of attack
6	Req. # 11, Req. #12	Seth	6-7	RESTler outputs documentation proving our service to be secure and not at risk of attack
6	Req. # 11, Req. #12	Erik	6-7	RESTler outputs documentation proving our service to be secure and not at risk of attack
7	Req. #16, Req.	Connor	4-7	URI handling for

	#17			stakeholders fully integrated. Auto grader API completed.
7	Req. #16, Req. #17	Sean	4-7	URI handling for stakeholders fully integrated. Auto grader API completed
7	Req. #16, Req. #17	Seth	4-7	URI handling for stakeholders fully integrated. Auto grader API completed
7	Req. #16, Req. #17	Erik	4-7	URI handling for stakeholders fully integrated. Auto grader API completed
8	Req. #18, final check	Connor	8-10	End to End tests all pass, ensure that the service has a quality user flow and experience
8	Req. #18, final check	Sean	8-10	End to End tests all pass, ensure that the service has a quality user flow and experience
8	Req. #18, final check	Seth	8-10	End to End tests all pass, ensure that the service has a quality user flow and experience
8	Req. #18, final	Erik	8-10	End to End tests

	check			all pass, ensure that the service has a quality user flow and experience