Project Phase 2

A Trustworthy Model Registry

Note: This document has Word headings in it. Enable "Sidebar > Navigation" to facilitate browsing.

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; and performance and cybersecurity considerations.

WARNING:

> If attempted in its entirety, this project spec contains far more work than I think you can do in the time allowed. I am always happy to be surprised, but in your project plan, your team may identify and justify a reduced scope. **This reduced scope should be grounded in development cost estimates based on your team's velocity in Phase 1**.
>
> Unless there are special circumstances (e.g. a team member drops the course), all teams should complete the **baseline** system and may propose to complete some subset among the **extended features**.

In this phase you have slightly more constraints.

- You must use multiple components from the free tier of Amazon Web Services.
- You must start with an existing implementation of Project 1 – not your own. You will maintain, refactor, and evolve this implementation.
- 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.

Phase 2 has a longer timeline than Phase 1 (about twice as long). It will run until the end of the semester.

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 edifying.
  - 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/
- **Amazon Web Services**
  - Here's the starting point: https://aws.amazon.com
- **OWASP Top 10**
  - Here are some common ways that engineers introduce cybersecurity vulnerabilities into the systems they develop. Your team should be familiar with these and work to mitigate them in your design and implementation. However, note that a rote check-box approach to security will result in an insecure system. Anyway, here is the OWASP Top 10: https://owasp.org/www-project-top-ten/
- **Twelve-Factor Apps**
  - Some wisdom about building cloud-first applications: https://12factor.net

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.

The 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. *Typical case:* Fix defects, refactor, add, or revise existing components as necessary.
2. *Unusual case*: Burn it down and start from scratch.

Note that, for intellectual property reasons, you cannot reference nor copy code from your previous company's implementation (*i.e. you cannot use your own Phase 1 code*). As a heuristic, if what you inherited looks like it was made in a weekend, then you would probably be well served by doing it again yourself. Otherwise, building on it is probably more sensible.

ACME Corporation is continuing to expand its use of machine learning models. They are really benefiting from BSS's tool for scoring HuggingFace models. However, they find themselves dissatisfied with aspects of the HuggingFace registry and their process for interacting with it:

- HuggingFace contains many untrustworthy or irrelevant models, which pollute the search results.
- ACME Corporation's engineers download many models, and HuggingFace can take a while to serve the desired content.
- ACME Corporation would like to put its own models – which are trusted, of course – into the same place where they keep their third-party models, but it does not want to share those models publicly. It seems hazardous both for their reputation and their Intellectual Property 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 models have been vetted. HuggingFace does not support a good way to distinguish vetted from unvetted models except popularity measures, which are not ideal.

For these reasons, ACME Corporation would like your team at BSS to develop a custom registry for storing machine learning models.

Sarah is still your contact person at ACME Corporation. Here are ACME Corporation's desired requirements, in her words. As usual, Sarah could have done a better job of organizing her thoughts, but she's a busy person and so you'll have to identify and disambiguate some of her requirements for yourself. Read carefully and get analyzing.

Sarah wants to see a demonstration of all their baseline requirements, and then a deeper treatment to see what BSS can do on some of the extended requirements.

[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 therefore 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). Your team will ultimately be responsible for delivering a quality product, which may require refinements to the BSS prototype. *(In less coded language, what I'm saying is that you may have inherited a dumpster fire.* ***Treat new codebases like snakes – cool but maybe poisonous****).*

Sarah's Baseline Requirements

Functional requirements

Your system must support the following general behaviors.

- **CR[U]D:** (1) Upload registry artifacts, (2) rate them, and (3) download artifacts.
    - The "rate" option should return the net score and sub-scores from Part 1, to inform prospective users of the quality of the model. It should also include new metrics:
        - **Reproducibility**: Whether the model can be run using only the demonstration code included in the model card. Distinguish 0 (no code/doesn't run), 0.5 (runs with [an agent] debugging it), and 1 (runs with no changes/debugging)
        - **Reviewedness**: The fraction of *all code (not weights)* in the associated GitHub repository (if any) that was introduced through pull requests with a code review. If there is no linked GitHub repository, use -1 for the value.
        - **Treescore**: Average of the total model scores of all parents of the model according to the lineage graph (described below).
    - The "download" option will include the full model package, or sub-aspects (eg just the weights, or just the associated datasets) in case the customer does not want to pay the cost of the full download.
- **Model ingest:** Request the ingestion of a public HuggingFace model. To be ingestible, the package must score at least 0.5 on each of the non-latency metrics from the "rate" behavior. If ingestible, the command should proceed to a package upload. For this to work right, you should add (or repair) any Phase 1 metrics that are missing or erroneous in BSS's solution.
- **Enumerate:** Fetch a directory of all models. Consider that this query may involve an enormous amount of data, e.g., if the registry has millions of models in it, so you should choose a design that won't become a denial-of-service vector.
    - You should be able to search for a model using a regular expression over model names and model cards. The results of a package search should be a subset of the "directory" results.
- **Lineage graph:** Report the lineage graph of a model, obtained by analysis of the available model structured metadata (config.json). Lineage only includes data available from the models currently uploaded to the system. See the comparable HuggingFace feature.
- **Size cost:** Check the size cost of using a model, measured in terms of the size of the associated download.
- **License check:** Given a GitHub URL and a Model ID, assess whether the GitHub project's license is compatible with the model's license, for "fine-tune + inference/generation"

    - You might look at the ModelGo paper for starters. Title: "*ModelGo: A Practical Tool for Machine Learning License Analysis*"
- **Reset:** Reset to the default system state (an empty registry with the default user)

Interface requirements

Your system should be accessible via both:

1. *Programmatic:* A REST-ful API. Specifically, you should comply with the OpenAPI schema that accompanies this document.
2. *For humans*: A pleasant web browser interface.
    a. Your GUI must implement at least a subset of functionality. While we leave the exact subset up to you, it must be more than a single "query" text box.
    b. On making it *pleasant*, the interface must have *at least* basic styling. While the specific design is up to your creative abilities, a plain, unstyled white HTML page is not acceptable. You are welcome to use libraries like Bootstrap or Material-UI to make this easier.

Non-functional requirements

General engineering standards

Your system implementation must reside in a single GitHub repository, including both source and test.[2]

You should have automated tests at the unit (component) level, feature level, and end-to-end/system level. These tests should together achieve at least 60% line coverage.

You **must** use some project management software (e.g. Trello, Monday, GitHub Projects, Asana, Jira, etc.). Since you are building in GitHub, I recommend but do not require that you use GitHub Project Boards. You **must** use reports from that project management software in your weekly milestones, via screenshots etc.

You may re-use existing software to support your implementation, either as tools (*e.g.*, VSCode; git; GitHub; TravisCI; ChatGPT) 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 reliable and trustworthy? (*discuss in the Project Plan*) and how did that assessment work out in practice? (*may need to discuss in the Project Postmortem*)

As in Phase 1, you are required to incorporate LLMs into your engineering process, and your Project Plan should include a description of your approach.

[2] On this note, you may wish to look up the concept of a "monorepo" – it's discussed in the Software Engineering at Google textbook.

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.[3] Any re-use of this nature must use existing module APIs and/or extend those APIs so that you can access the logic you want.

You must make use of:

- Dependabot
- GitHub CoPilot Auto-Review
    - https://docs.github.com/en/copilot/how-tos/use-copilot-agents/request-a-code-review/configure-automatic-review

Front-end engineering standards

1. The front-end user interface should be accompanied by automated tests, e.g., using the GUI testing framework Selenium.
2. The front-end user interface should be compliant with the Americans with Disabilities Act (ADA)[1].
    a. 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 US Department of Justice).
    b. Your website should be compliant with WCAG 2.1 at level AA.
    c. See https://www.w3.org/TR/WCAG21/ for details. These free tools from Microsoft may be helpful.

Cybersecurity

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

1. *Design for security:* Develop a system design in the ThreatModeler platform.[4]
2. *Analyze risks*:
    a. *Principled*: Conduct a security analysis of your system based on the STRIDE approach. Include a dataflow diagram with the trust boundaries clearly indicated.
    b. *Prioritized*: Analyze your system with reference to the OWASP Top 10 and other lists of security best practices and threats.
    c. *Automated*: Assess all the "best practices" recommendations provided by ThreatModeler and indicate which ones are relevant.
3. *Mitigate:* Enumerate and rank the security risks you identify through these methods. Modify your design and implementation as appropriate, until you exhaust the ones you deem most critical. Justify any excluded risks based on your threat model.

[3] Q: "Doesn't open-source mean we can copy the source?" A: Here are some example lawsuits: https://en.wikipedia.org/wiki/Open_source_license_litigation.

[4] I am discussing support with ThreatModeler. They have not yet confirmed access. I will update this by November 1. If they cannot give us access, we will use a standard drawing platform, but this will be less awesome.

4. *Document*: You presumably identified and mitigated at least four vulnerabilities as a result of this analysis. In your security case, analyze how these vulnerabilities came to enter your system (perhaps with a "*Five Whys*" analysis). To do this, you will need strong record-keeping in your version control system – traceability from code to author and associated processes and decisions. This analysis will increase the customer's confidence that you are learning from these errors and will have fewer in the future. **Make sure that your project plan has enough detail about what records you will keep to facilitate meeting this requirement**.

System deployment

ACME Corporation is a big company, so for scalability reasons your system should be deployed on Amazon Web Services (AWS). You must expose a single uniform API, but under the hood you must use at least two AWS components. These should be indicated in your project design.

AWS offers a Free Tier. Use it. Do not exceed it. Work out your expected costs in advance using a spreadsheet or an AWS tool. AWS provides usage monitoring – set alerts to prevent sadness. Evaluate on small resources where possible. The course staff are not responsible if you accidentally owe AWS money. If you owe AWS **lots** of money please let the course staff know and we can try to intercede on your behalf.

- For LLM use, I believe the AWS SageMaker platform will work, though you may need to switch to another account in early December based on their current free tier model. **Again, set alerts**.

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 AWS on successful merge (CD/Continuous Deployment)

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

Observability

You must construct a system health dashboard that will allow ACME Corp.'s operations staff to monitor the health of your system. At a minimum, report semi-real-time data about activities in the last hour, and make it possible to inspect logs.

You must provide both a data feed (through an API endpoint, "/health") and a visualization through your web UI.

Auto-grading

Comply with the OpenAPI specification we provided. The autograder assumes you did so.

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 pursue the Access Control Track, 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.

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. Our autograder will run as a client of the system.

Statement on academic honesty

Per the academic honesty requirements of this course, you are not permitted to compare software implementations with other teams.

- The source you are building on from Phase 1 is visible; please copy the repo and make a private version for your team's use.
- One specific aspect of the implementation that you **can** discuss with other teams is "Operations" issues such as AWS configuration.
- You would be wise to incorporate

Sarah's Extended Requirements

Beyond these general requirements, Sarah would like to see your team do a "deep dive" on some aspect of the registry system. This will allow her to see your team's abilities and will also reassure her that you will be able to provide refined versions of the other aspects as needed.

Your team must select at least one of these extended requirements. Your project plan should indicate what you are doing and why. You need not attempt to satisfy every aspect of the extended requirements, and you have some creative freedom in definitions and approaches.

To grade these, we will emphasize manual review – studying your design and your report to see how well you tackled them. For the Security Track, we have an extended version of the OpenAPI spec and the autograder for this purpose. Thus, although you may think that the Security track is the "easy" one, we will also hold your results to a higher standard of correctness.

See also the "Warning" at the top of this document.

Security track

User-based access control

ACME wants to make sure only authorized employees can interact with packages.

- 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.
- A single user may have multiple concurrent tokens.
- Some users should have an “admin” permission. Only administrators can register users. Users can delete their own accounts. Administrators can delete any account.
- Password storage, authentication, etc. should all be designed and implemented using secure means. Include this aspect in your cybersecurity analysis.

A notion of sensitive models

ACME Corporation is concerned about the downloading of sensitive models.

- They would like to execute an arbitrary JavaScript program prior to the **download** of any sensitive models. Each such sensitive model can have zero or one JavaScript programs associated with it. Any user can upload and delete sensitive artifacts 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 v24 and accepts four command line arguments: “MODEL_NAME UPLOADER_USERNAME DOWNLOADER_USERNAME ZIP_FILE_PATH”. If the program exits with a non-zero code, the download of the model 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 models – which accounts downloaded them at what times?

Package Confusion Attack detection

Read:

- Beyond typosquatting: an in-depth look at **package confusion**.
- https://arxiv.org/pdf/2502.20528

Support an endpoint called “PackageConfusionAudit” that returns a list of packages that you suspect are malicious uploads that are trying to leverage package confusion.

As part of this, you should make use of package metadata information, e.g., popularity as measured by (1) presence in searches, and (2) usage via downloads. Confusion attacks often resemble real packages but are less popular. Attackers also make use of bot farms to inflate their popularity, so you might make some attempt to detect packages with anomalous download traffic.

Performance track

Measurements

ACME Corporation wants to know the throughput as well as the mean, median, and 99th percentile latency when 100 clients are all trying to download a copy of the Tiny-LLM model (ingested from https://huggingface.co/arnir0/Tiny-LLM), from a registry containing 500 distinct models. Describe your experimental design. Provide measurements of performance (black-box) and explanation (white-box).[5]

This workload should be triggerable from your team's system health dashboard, completed as part of the baseline requirements.

Identify at least two performance bottlenecks from these measurements. Describe how you found them. Describe how you optimized them. Describe the effect.

Here are two useful papers on this topic:

- https://dl.acm.org/doi/abs/10.1145/3213770
- https://gernot-heiser.org/benchmarking-crimes.html#sign

Experimenting with the effect of components

What is the effect of different AWS components on the performance (e.g., latency, throughput) of your system? For example, how much of a difference do you get from Lambdas vs. EC2, or object store vs. a relational database? Modify your system so that it is possible to select the underlying component as part of the configuration. Provide measurements of the effect.

High-assurance track

Test test test

Demonstrate at least 90% line coverage via component-level testing alone, and 95% coverage when including feature and end-to-end tests.

Demonstrate that at least 80% of your error messages are produced during the execution of your test suite.

Your test plan and implementation should carefully discuss the goal of hermetic testing (as described in the SWEng@Google textbook) and the extent to which you realized it, and your rationale.

[5] You've taken courses that involve writing lab reports. Please apply that training here – we will expect to see a clear and sound description of your measurement design.

Disaster proofing

Show the extent to which your system is robust[6] to the loss of a single AWS component, e.g., "S3 stops working". Use your system models to indicate the scenarios you should test. Perform the tests. Improve robustness (e.g., make sure reasonable error messages are provided).

Atomic updates to prevent race conditions

ACME Corporation is concerned about race conditions, e.g. when a pair of models depends on each other and an update to one of them might be visible before the other one is updated. Therefore, you should support the upload, update, and download of **groups** of packages, so that multiple related packages can be accessed atomically. Structure this by supporting a three-stage transaction accomplished through multiple API interactions – (a) initiate an empty request; (b) append upload/ingest commands to the request; and (c) execute the request.

Timeline and Deliverables

The project will be completed over a ~9-week period. Dates are on Brightspace/Gradescope. Roughly:

- Initial plan
- Internal milestone
- Delivery 1: Demo of some functionality (CI/CD plus the following baseline functionality: CR[U]D, Ingest, and at least some Enumerate).
- Internal milestones
- Two weeks before final deadline: Autograder becomes available.
- Delivery 2: Deliver the rest of the functionality
- Postmortem

Details follow on each component.

Project plan

Submit a Project Plan Document (Word Doc or PDF) including the following. Some of this can be copied from Phase 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?]
    - Role of LLMs
- Team contract
    - For example, your team might agree to do the work they take on, to document their code, testing rules, style guide, timeliness expectations
- Team synchronous meeting times

[6] The definition of "robust" is up to you. You will need to define it and show you've reached it.

    - 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
        - At least one activity diagram to depict the various activities performed by your system. You will likely need more, though you can say "activities X and Y are similar" as needed.
        - At least one dataflow diagram to depict the flow of information (including who can access which information, which you will want for the security case)
        - Diagrams should be drawn with a tool such as LucidChart or draw.io.
- Timeline and planned internal milestones
    - 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 Phase 1 that they left integration or validation until the end and ran into trouble. Do you want to try something different on Phase 2?
- Starting project analysis
    - You **must** use another team's Phase 1 implementation. This implementation may not work correctly. As part of your plan, you should review this implementation and answer questions such as: the cost (if any) of integrating it into your design; 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.
- Lessons learned from Phase 1
    - You completed a postmortem as part of Phase 1. You must indicate how those lessons learned are being integrated into your Phase 2 plan.

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.

Deliveries

Submit the software itself, along with documentation.

For Delivery #1, this is mostly screenshots and a summary of your progress relative to schedule.

For Delivery #2, a substantial report will be needed to describe the status of the software in relation to Sarah's requirements and specification.

Postmortem

Deliver a project postmortem report. This report should reflect on each aspect of your Plan 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?"

A template will be provided.

Grading rubric

Points breakdown:

- 30% Design & Planning document + Milestone documents.
- 60% Working delivery, broken down as:
    - 35% "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."
- 10% 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 the course staff 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 the course staff.

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**, 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 ongoing 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.
- 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.
    - (Note: It is possible for us to autograde a subset of features, so if you need to, then make it easy for us to give you partial credit by doing a reasonable subset well).
- In your planning document, you should enumerate and organize the requirements, estimate the cost of each feature, and identify the subset of the functionality that you think your team can reasonably deliver. The total desired cost for your team should be 80-90 hours per person. Estimate and negotiate accordingly. Revise estimates as needed to support any renegotiation of the delivery later on.

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 AWS options while others start the implementation locally. You might similarly prefer to start your AWS deployment with something simple like a "big fat EC2 VM" and only break it into multiple components later. In a similar vein, there are tools to test GitHub Actions locally to reduce debugging time on those.

When offering a web service such as the Trustworthy Model 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.

You might wish to have individual team members specialize in AWS components and handle all elements of each feature that interacts with that component. Or you could have team members specialize in features and learn enough about each AWS component that you can add the relevant feature code to it. There are many ways to divide and conquer, but please be thoughtful about which one seems appropriate. Also consider bus factor and your experiences in Phase 1.