Assignment T5: Second Iteration

Published | Assign To | Edit | ⋮

This is a team assignment. See submission instructions at the end.

Implement the final version of your server and make sure all the implementation code has been committed to the main branch of your github repository (its ok to have some other code in branches but it will not be considered part of your submission). In your github repository, **tag** (https://stackoverflow.com/questions/18216991/create-a-tag-in-a-github-repository) the set of file revisions considered part of the the second iteration. **The most significant differences between the first iteration assignment and this second iteration assignment are written in bold.**

Develop a "client" for your service. Your client could be an app, with a UI, or another service, with its own API. Your client should be demoable, but please do not spend a lot of time making a fancy UI! This is a software engineering class, not a user interfaces class. Explain in your README how to build and run your client (ideally building the client would be included in CI - see below - but that is not required). It's ok for the client to run on "localhost". Ideally your client code would be in the same repository as your service (Google stores billions of lines of code in a single repository (https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext)), but it's ok to put the client code in a separate repository as long as the README for your main repository provides a link to this second repository.

Make sure your server can interface to *multiple instances* of the client and tell them apart, e.g., by using an existing package or service supporting authentication and authorization. Note it is the responsibility of your server to verify authentication and authorization, do not trust the client!

Add continuous integration (CI) to your main github repository, where your server's codebase resides, so that all your build, analysis and testing tools are automatically run for every commit. Integrate CI with your repository, e.g., using github actions (https://github.com/marketplace?category=continuous-integration&type=actions) or github apps (https://github.com/marketplace?category=continuous-integration&type=apps). Include the CI reports in your repository and explain where to find them in your README (or include a clearly marked link in the README to where your CI reports reside). If the CI reports include any or all of the other reports mentioned below, you do not need to also include them separately, but please explain what is and is not included with the CI reports in your README.

Implement and run unit tests for all the major "units" (methods, functions, procedures, subroutines, etc.) in your server codebase. Unit tests should be grouped so related code (e.g., classes, modules, files, etc.) share setup, teardown and mocks when appropriate. Your entire collection of unit tests should run automatically during CI.

Implement integration tests between units that work together in some way, either by invoking each other or by sharing data. All your integration tests should run automatically during CI.

Implement system tests that exercise your server's final API, e.g., using some API testing tool. Your tests should exercise the multiple clients and persistent data aspects of your server. All your API tests should run automatically during CI.

You should use a branch coverage tool together with your unit, integration and system testing during CI. Try to achieve at least 85% branch coverage. Include your testing and coverage reports in your server repository. Try to fix most of the bugs found by the tests.

You should also use a static analysis bug finder tool on your entire server codebase. The static analyzer should run automatically during CI. Include the static analysis reports in your server repository. Try to fix most of the bugs found by the analyzer.

Implement end-to-end tests where your client exercises as much functionality of your server as possible. Although ideally automated, it's ok to run end-to-end tests manually. Document any manual tests with a checklist or some other mechanism to make sure you can re-run the exact same set of tests as needed, e.g., after fixing a bug.

Make sure all your test code and scripts have been committed to the main branch of your server repository (its ok to have some other test code in branches but it will not be considered part of your submission). All commit messages for your server repository, except for the initial repo-setup commit(s), should say something meaningful about what was added/removed/changed. All non-trivial code in your server codebase, including test cases, should be self-documenting with mnemonic identifiers and header comments and, where appropriate, in-line comments. (Trivial means things like getters, setters, small helper functions, and other self-evident code.)

Your entire server codebase, including test cases and even trivial code, should be compliant with a style checker appropriate for your language/platform. Include some reports from your style checker in your repository. If it does not generate files, use redirection of output to a file, screenshots, or some other means to produce files.

Include in your server repository any configuration files needed to build, run and test your server. Write a README.md that describes in simple terms how to build, run and test your server. The README.md should also describe all the operational entry points to the final version of your server or link to some external documentation, e.g., **Swagger (https://swagger.io/)**. This is your "API documentation". If certain entry points must be called in certain orders or never called in certain orders, make sure to say so.

Also include in your README.md a link to your client repository (if separate) and instructions explaining how to run your sample client with your server. Describe how some third-party could develop and run their own client that uses your server.

If any third-party code is included in either your server or client codebase, also document exactly which code this is, where it resides in your repository, and where you got it from (e.g., download url).

Submission instructions: One member of your team should submit a url pointing to your codebase repository. We encourage you to make your repository public, but if you prefer to keep it private then make sure the instructor and your Mentor have collaborator access! This should normally be the same repository from your revised proposal, but we ask you to submit again for this assignment to make it easier to use Coursework's "SpeedGrader" tool.

Points	50
Submitting	a website url

Due	For	Available from	Until
Nov 28, 2022	Everyone	-	Nov 29, 2022 at 11:59pm

Second Iteration Rubric	You've already rated students with this rubric. Any major changes could affect their assessment results.

Criteria	Ratings	Pts
Client Did they develop a demoable client that exercises a reasonable subset of their API, is the client code in the main repo or in an auxiliary repo linked from the main repo, and is there a README that documents how to build the client and run the client together with their service? They were not required to run testing or checking tools on the client except for end-to-end system tests (separate criterion below).	This area will be used by the assessor to leave comments related to this criterion.	5 pts
Multiple Client Instances Does their service support multiple simultaneous instances of the sample client? Can the service tell them apart? Can the service handle multiple requests/responses from different client instances in parallel and track which requests involving which data came from which client (and send corresponding responses only to the appropriate clients)? Does the test suite for the service include tests that check that the service indeed operates properly when it has multiple concurrent clients?	This area will be used by the assessor to leave comments related to this criterion.	5 pts
Third-Party Clients Did they document how some third-party could develop and run their own client that uses the team's service? Is the documentation sufficient to tell the third-party developer how to use an arbitrary subset of the service's API, not necessarily the same subset as the team's sample client? Does the documentation address what a third-party developer would need to do for authentication/authorization and for multiple instances of their client to use the service simultaneously with each other as well as concurrently with multiple instances of the team's own client and other third-party clients?	This area will be used by the assessor to leave comments related to this criterion.	6 pts
Authentication/Authorization Does the service enforce some reasonable form of authentication and authorization on all client instances? Is this authentication/authorization independent from end-users? (End-users should not be logging into the service.) If any current or future clients send private data to the service, does the service properly restrict access to that private data? Does the test suite for the service include tests that check whether the authentication/authorization protocol works? Are there tests that check for cases like a client tries to use API functions without authenticating and a client tries to perform unauthorized actions such as access another client's data? (It's ok if some API entry points are intended to be accessed without prior authentication, as long as there are some other API entry points that do require prior authentication.)	This area will be used by the assessor to leave comments related to this criterion.	5 pts
Continuous Integration Does the main repo perform CI such that all build, analysis and testing tools are automatically run for every commit to the main branch of the codebase (or on a schedule such as nightly)? Are the CI reports in the repo or linked from the repo?	This area will be used by the assessor to leave comments related to this criterion.	5 pts
Integration Tests Does the codebase include automated integration tests that run automatically during CI and include tests to integrate classes with each other? Are there also integration tests for testing the interfaces to all third-party libraries and services used? Are there integration tests for checking the interfaces to all environmental or external resources such as files and databases?	This area will be used by the assessor to leave comments related to this criterion.	6 pts

Criteria	Ratings	Pts
Branch Coverage Did the team run branch coverage during automated testing? Does the repo contain or link to coverage reports? Is the branch coverage at least 85%? If 85% was not achieved, is there some explanation why not in the README or elsewhere in the repo?	This area will be used by the assessor to leave comments related to this criterion.	5 pts
Bug Finder Did the team run a static analysis 'bug finder' tool on their entire codebase (except for the client and third-party code)? Does it run automatically during CI? Are bug finder reports included in or links frokm the repo?	This area will be used by the assessor to leave comments related to this criterion.	5 pts
End-to-End Testing Is there reasonable evidence that the team did substantial end-to-end testing of their client utilizing their service? It's ok if these tests are manual, but there should be a written checklist or some other mechanism that makes it easier to redo the exact same tests multiple times.	This area will be used by the assessor to leave comments related to this criterion.	5 pts
Third Party Code Do they document all third-party code used, whether library, API, database, etc., where it is in the repo (if applicable), and where they got it from (e.g., link to download or documentation)?	This area will be used by the assessor to leave comments related to this criterion.	1 pts
API Documentation Has their API documentation been updated to the 'final' version of their service?	This area will be used by the assessor to leave comments related to this criterion.	1 pts
Configuration Is sufficient documentation included in the repo to enable an arbitrary developer (who already knows their programming language and its ecosystem) to build, run and test their service? Are all necessary configuration files included in the repo?	This area will be used by the assessor to leave comments related to this criterion.	1 pts
		Total Points: 50