Product Backlog - Team 19

SquareCI

Members: Brandon Marx, Jonah Heeren, Jun Soo Kim, Shulin Ye, Vritant Bhardwaj

Problem Statement

Deploying software can be a cumbersome task that often occurs without complete testing of new features being pushed into production. To combat this, continuous deployment(CD) provides an automatic way to push new code out to production while still validating that your changes integrate and don't break existing features.

Most CD solutions are either proprietary or costly. We will develop a simplified version of continuous deployment that can be utilized easily and cheaply by anyone.

Background Information

- There are services that exist that run pull requests in Github through continuous integration tests before allowing them to be merged into master, like CircleCI. But they only work on branches in the version control system.

- Manually running tests on your codebase, and accordingly pushing the code to either alpha or beta test servers, or production servers is a tedious task that can be automated.

- Our application, SquareCI allows users to push features or updates into one location, and the tests run on it specify whether or not it is ready for alpha, beta or production.

- SquareCI has the unique ability to provide the user the big picture perspective that will help them think through the project testing effort in a structural way.

Environment

The front-end web interface will be a one stop place for the user where all interaction will occur, from adding a repository to seeing test logs. It will be built using the React.js library, as it allows for easy state management and real time updates. The component system also allows for a structured and modular interface for the user to monitor the tests being run on their code.

The backend will be hosted on an EC2 instance and written in Javascript and will use Node.js. The NPM packages available and the compatibility with the React.js

front-end is why we chose this option. We will also be interacting with the GitHub API, this was an obvious choice because it is the most widely used source control and provides a flexible API that will allow us to query the appropriate data about repositories.

Functional Requirements

BackLog ID	Functional Requirement	Hours	Status
F0	As a user, I would like to have a web interface to access the application.	10	Sprint 1
F1	As a user, I would like to connect my GitHub repository to the application.	8	Sprint 1
F2	As a user, I would like the application to check for commits into the master branch every user-defined units of time.	7	Sprint 1
F3	As a developer, I would like the application to retrieve from master once a new commit has been made in it.	9	Sprint 1
F4	As a user, I would like my updated master branch to be tested against user defined test cases.	6	Sprint 2
F5	As a developer, I would like to make 3 server endpoints (Alpha, Beta, Prod) for the deployment of a repo.	8	Sprint 1
F6	As a user, I would like the code deployed on a specific server endpoint based on the tests the code passes.	6	Sprint 2
F7	As a user, I would like to monitor the status of the tests being run on my code.	8	Sprint 2
F8	As a user, I would like to know what the results of the tests are.	6	Sprint 1
F9	As a developer, I would like the repos deployed after the servers go down	6	Sprint 2
F10	As a user, I would like my login to persist	6	Sprint 2

	even if the browser closes.		
F11	As a user, I would like to login to use the application.	8	Sprint 1
F12	As a user, I would like to have detailed information on the unpassed tests.	8	Sprint 2
F13	As a user, I would like to logs on every test run.	10	Sprint 2
F14	As a user, I would like instructions on how to setup the repository to be monitored	5	Sprint 1
F15	As a user, I would like to create an account	10	Sprint 1
F16	As a user, I would like to unlink my repository from the application.	6	Sprint 2
	Total	124	

Non-Functional Requirements

Backlog ID	Non-Functional Requirement	Hours	Status
NF1	As a user I would like my server credentials to remain secure.	10	Sprint 2
NF2	As a user I would like this service to run reliably.	11	Sprint 1
NF3	As a user I would like the web interface to be responsive.	6	Sprint 2

NF4	As a developer, I would like scalability to be auto-managed.	8	Sprint 1
NF5	As a developer, I would like logs to be as application/language agnostic as possible.	10	Sprint 1
NF6	As a user, I would like the web interface to be easily understood.	5	Sprint 2
NF7	As a first-time user, I would like the tester website to have instructions on all the functionality.	8	Sprint 2
	Total	58	

Use Cases

Case #1: First-time user getting started on the website

Action	System Response
1. User clicks create account button	2. Direct to the registration page
3. User enters information and submits the create account form	4. Display information on whether the account creation is successful.
	5. If registration is successful, direct user to landing page.

Case #2: User logs into account

Action	System Response
1. User clicks log in button	2. Direct user to log in page.
3. User enters information and submits login form	3. Direct to user's dashboard if successful, else display error.

Case #3: User begins to add repository information

Action	System Response
1. Click the button on the website "Connect your Project".	2. Direct user to add repository page.
3. User enters information and submits the link repository form	4. Display information on whether the linking is successful.

Case #4: First-time user getting tips for how the application works

Action	System Response
1. Click the button "Tips" on the website	2. Display graph and text instructions explaining how the flow of the tester is.

Case #5: user setting the time interval for the application to check the commits

Action	System Response
1. Click the button "Set time interval"	2. Display a box where the user can

on the website	type in a number
3. Type in a certain time interval	4. Set GitHub polling time

Case #6: user sets the endpoint for one of the 3 default testing phases

Action	System Response
1. In repository info, a user enters server information.	2. Information is encrypted.
	3. Information is securely transmitted to our server.
	4. Our server remotely connects to user servers.
	5. SSH-RSA auth keys are generated such that username & passwords are not stored permanently.

Case #7: user changes server config

Action	System Response
1. User makes modification to one of their servers that makes it unreachable.	2. Web browser notification is sent
	3. Error text is displayed on web UI

Case #8: user adds commits to master

Action	System Response
1. User makes/merges commits to/into master.	2. Our server polls for changes on user specified time.
	3. Changes are detected by comparing the latest commit hash to the last recorded one.
	4. Latest application state is downloaded.
	5. Source code is compiled
	6. Alpha tests ran
	7. On success, changes are deployed to alpha and logged. On fail, deployment stops and logs are generated specifying failed tests.
	8. Beta and prod tests run and work in the same fashion.

Case #9: user wants to see test progress

Action	System Response
1. User selects repository	2. Request is sent to the server for current progress of test
	3. Server responds with specific test logs
	4. Test progress logs are received by the client and displayed to the user

Case #10: user wants to see test logs

Action	System Response
1. User selects repository	2. Request is sent to the server for logs
	3. Server responds with specific test logs
	4. Test logs are received by the client and displayed to the user

Case #11: user removes repository monitoring

Action	System Response
1. User clicks remove repository monitoring.	2. Polling halts on our server.
	3. SSH-RSA auth keys are removed.
	4. Text displaying the result of the operation.

Case #12: user logs out of account

Action	System Response
1. User clicks the logout button	2. User is logged out and is directed to the landing page.