Incremental Testing and Regression Testing For Second Sprint 2

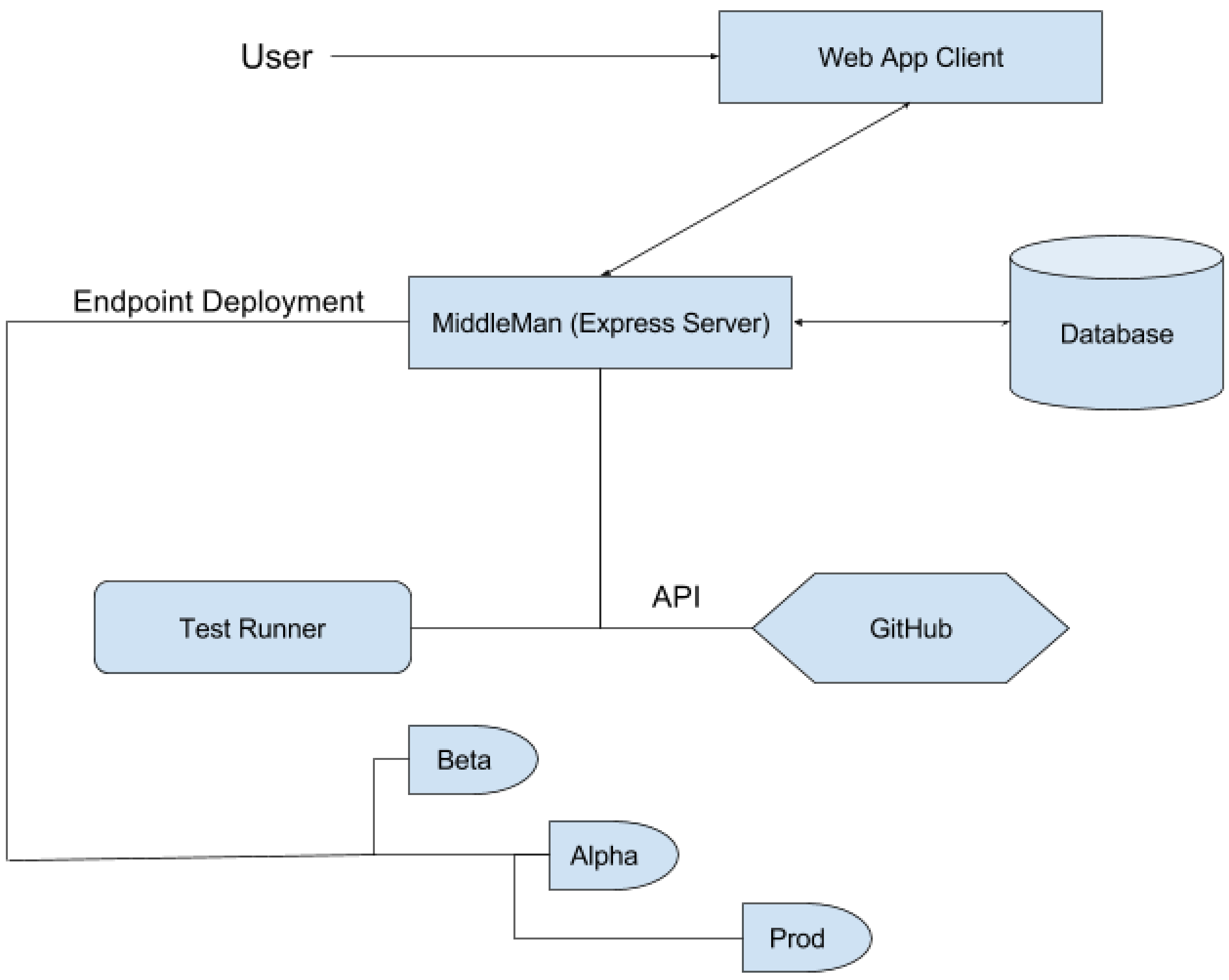
- Team 19

SquareCI

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

1. Classification of Components

1.1 Define all Components



Component	Input	Output	Dependent Components
Component 1: MiddleMan	RESTful calls from the Web App Client	Data about the users monitored repositories used to propagate the UI	Middleman Depends on the Database for stored user info, the Test Runner for information about testcase states, and GitHub for authentication and API calls
Component 2: Web App Client	User Input from site navigation	The User Interface and frontend logic that the user will continue to interact with	The web app client depends upon a user to request state changes and the MiddleMan server to securely access user data.
Component 3: Database	SQL queries from Middleman	Responses to SQL queries	Loosely tied to MiddleMan because it needs to receive input to insert and retrieve
Component 4: GitHub API	OAuth and Restful API calls to get additional User Data	User authentication tokens, and user data in the form of JSON objects	GitHub does not depend upon our application
Component 5: Test Runner	Test cases from user applications	Logs that detail the success or failure of user test cases	Test Runner requires test cases to be sent to it via MiddleMan
Component 6: Server Endpoints (Alpha, Beta, Prod)	User applications which have successfully passed that stage’s data.	Running instances of user applications on our backend server’s ports.	The server endpoints depend upon MiddleMan to deploy and start the applications.

1.2 Form of Incremental Testing Followed

Our team decided to follow **bottom-up** incremental testing.

This decision came from how our team's development style was structured. All of us worked on separate components once Access Tokens were generated from GitHub's OAuth. This allowed for each of us to be responsible for unit testing small sections and helped to create a deep understanding of how our components would respond to invalid and unusual input. Due to this, we were able to consider edge cases that helped in the security and stability of our application. Particularly, this was useful when we were making requests to GitHub's API. There were user fields that were optional that we hadn't considered, as well as thinking about cases where users have no repositories, and most importantly it allowed us to reflect on how we would manage authentication.

2. Incremental and Regression Testing

For our application, we chose to use Mocha and Chai as our testing frameworks. Both of these frameworks are widely utilized by Node.js and Javascript applications. Chai is a full featured assertion framework that provides near natural language usage to provide application assertions. Mocha is a test running suite that provides features such as test coverage generation, retry tests, diffs, etc.

While having clean and readable tests for our application was important, more importantly we needed to support testing frameworks that our application would use to report upon user application deployments. Mocha has a variety of reporters that we can use to change the output format. Most importantly, Mocha supports a JSON reporter which will allow us to easily and quickly pull this JSON into Javascript Objects and use the test data immediately without parsing effort.

Module	Component 1: MiddleMan

Incremental Testing

Defect No.	**Description**	**Severity**	**How to Correct**
1	Interpolated GitHub API urls were getting undefined references from variables.	1	Check for undefined variable references and throw exceptions in such cases.
2	GitHub API calls were failing.	1	Ensure that promise rejections were handled.
3	Attempting to get the user token was	1	Correct variable spelling that wasn’t

	failing.		correctly setting a global variable.
4	Correct payload not returned from webhook	2	Used different npm package one that uses express
5	Undefined output retrieved from webhook payload	3	Used correct variables from jsonObject instead directly from data
6	On monitoring a new user, was not getting proper data from the database	2	Was trying to access database entries before they were entered. Moved the code block down
7	On monitoring from webhooks, previous variables were undefined	2	Found correct variables from jsonObject previously made
8	When github repos were created, they were stored in an extra file layer	2	Added --stripcomponents=1 to remove top level folder and make sure all other functions worked off of this storage method
9	JSON objects were not being correctly parsed	2	Used a different NPM package to properly parse the json objects, did error catching to make sure it works properly

Regression Testing

Defect No.	Description	Severity	How to Correct
1	Undefined variables were being received because they were being used before API calls returned.	1	Use promises to create asynchronous blockers to ensure API calls had returned before their returned variables were used.
2	GitHub API calls were failing because they didn't include the user agent	1	Create additional headers to accompany the request that included the user agent.
3	Not having a repo monitored does not tell the client that things won't work, and proceeds as normal without getting any proper information.	2	Catch the promise rejection to let the user know
4	Could not download repos even after using the correct arguments	2	Changed User ID to the sender ID from the jsonObject to give access to their repositories
5	After accessing webhook payload, still had issues returning correct ID's and repo branch	2	Had to use the data parameter and convert it to a jsonObject to make it easier to access
6	Downloading ,unzipping, and running tests happen before the	2	Add promises to each function and properly use the .then function to

	previous one has completed.		control flow

Module	Component 2: Web App Client

Incremental Testing

Defect No.	**Description**	**Severity**	**How to Correct**
1	The map function combined with the filter function was causing index errors on the repo list	2	Changed to using a for each function
2	Status codes were incosistent	3	Created helper for status codes and used variables

Regression Testing

Defect No.	**Description**	**Severity**	**How to Correct**
1	After logging in, promise rejection was not caught if there was a problem in server's response	1	Added a reject catch in all promises in the client that display an error or redirect user.
2	Clicking on links to other routes refreshed the page	3	Added a wrapper component to the links to enable internal navigation
3	Going to a non existent route would show nothing	3	Redirect user to 404 page if route doesn't exist.

Module	Component 3: Database

Incremental Testing

Defect No.	**Description**	**Severity**	**How to Correct**
1	SQL inserts were allowing duplicate records	3	Add Primary Key constraints to SQL tables
2	SQL table merging doesn't give the desired result.	3	Make sure the function is called at right place where data is accessible.

Regression Testing

Defect No.	**Description**	**Severity**	**How to Correct**
1	Promise rejections were occurring with duplicate primary key constraint conditions.	3	Use INSERT IGNORE to automatically check if records exist before insertion.
2			

Module	Component 5: Test Runner

Defect No.	**Description**	**Severity**	**How to Correct**
1	Test output wasn’t being parsed correctly.	3	Ensure that mocha test reporter was outputting in JSON format.

Regression Testing

Defect No.	**Description**	**Severity**	**How to Correct**
1	A field was appearing undefined.	3	Ensure that variable usage is consistent.

Module	Component 6: Server Endpoints

Defect No.	**Description**	**Severity**	**How to Correct**
1	User applications were unable to deploy.	2	Ports were being used twice causing applications to not be able to be launched.
2	Deployment port list is returning empty.	2	Promises need to be used around exec functions.

Regression Testing

Defect No.	Description	Severity	How to Correct
1	Private repositories weren’t decompressing correctly.	2	Correct the unzip command that had an extra argument flag
2.	Deployment processes were not being killed after un monitoring a repo.	2	Track monitored repos and kill -9 by PIOD.

3. Update Product Backlog

BackLog ID	Functional Requirement	Hours	Status
F0	As a user, I would like to have a web interface to access the application.	10	Completed in sprint 1
F1	As a user, I would like to connect my GitHub repository to the application.	15	Completed in 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	Completed in 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	Completed in Sprint 2
F4	As a user, I would like my updated master branch to be tested against user defined test cases.	6	Completed in Sprint 2
F5	As a developer, I would like to make 3 server endpoints (Alpha, Beta, Prod) for the deployment of a repo.	8	Completed in 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	Completed in Sprint 2
F7	As a user, I would like to monitor the status of the tests being run on my code.	6	Completed in Sprint 2
F8	As a user, I would like to know what the results of the tests are.	4	Completed in Sprint 2
F9	As a developer, I would like the repos deployed after the servers go down	4	Completed in Sprint 2
F10	As a user, I would like my login to persist even if the browser closes.	5	Completed in Sprint 2
F11	As a user, I would like to login to use the application.	8	Completed in sprint 1
F12	As a user, I would like to have detailed	8	Completed in

	information on the unpassed tests.		Sprint 2
F13	As a user, I would like to see logs on every test run.	10	Completed in Sprint 2
F14	As a user, I would like instructions on how to setup the repository to be monitored	5	Completed in sprint 1
F15	As a user, I would like to create an account	10	Completed in sprint 1
F16	As a user, I would like to unlink my repository from the application.	6	Completed in 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	Completed in Sprint 2
NF2	As a user I would like this service to run reliably.	11	Completed in sprint 1
NF3	As a user I would like the web interface to be responsive.	6	Completed in Sprint 2
NF4	As a developer, I would like scalability to be auto-managed.	8	Completed in sprint 1
NF5	As a developer, I	10	Completed in Sprint

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