Integrated Tools Dashboard

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By
Jasnoor Brar, Varun Jain, Rohan Kamat, Abhishek Konduri
05/2020

APPROVED

Professor Dan Harkey, Project Advisor

Professor Dan Harkey, Director, MS Software Engineering

Professor Xiao Su, Department Chair

ABSTRACT

Integrated Tools Platform
by
Jasnoor Brar, Varun Jain, Rohan Kamat, Abhishek Konduri

There are a set of various tools used for software development and project management in any company. The tools differ based on the stage of the development cycle, for source code and version control tools like GitHub and Bitbucket are used. The build phase involves usage of tools like Jenkins, Bamboo, Travis CI etc. It is very important for the manager to keep track of the application development process, where tools like JIRA, Kanban are used. The last stage is the maintenance, each application generates logs during its build phase as well as production phase. The data from these logs are very crucial in determining the performance, security aspects of the applications, tools like Splunk, Sumo Logic are used for this purpose. In order to get information from these software's and tools, an individual has to go to that particular software web application and search for what he needs. An employee has to login to each of the software and navigates to different tasks. Logging in and navigating to what is required takes up some time. How good would it be if he finds everything, he needs at one single place and he does not need to spend time explicitly going to each of these tools and getting his task done? In order to address all such issues, we have developed a web application dashboard with everything integrated at one place. This is a single point of control for accessing basic information or doing basic tasks. If there is a requirement wherein the person has to use the extended application, the dashboard takes him to the main application.

Acknowledgment

The authors are deeply indebted to Professor Dan Harkey. We would like to express our special thanks of gratitude to our Project advisor Professor Dan Harkey, who always gave us his in-depth advice on certain components and the overall structure of the project which helped us in successfully moving forward with our design and implementation.

Table of Contents

Chapter 1. Project Overview 10

1.1 Introduction 10

1.1.1 JIRA 10

1.1.2 GitHub 11

1.1.3 Jenkins 11

1.1.4 Splunk 11

1.2 Current State of the Art 12

1.3 Project Requirements 13

1.3.1 Functional Requirement 13

1.3.1.1 Authentication 13

1.3.1.2 Administrative functions 13

1.3.1.3 Dashboard Functionality 13

1.3.1.4 Authorization levels 14

1.3.2 Non-functional Requirements 14

1.3.2.1 Reliability 14

1.3.2.2 Security 15

1.3.2.3 Usability 15

1.3.2.4 Scalability 15

Chapter 2. Project Architecture 16

2.1 High-Level Architecture 16

2.2 Description 16

Chapter 3. Technology Descriptions 18

3.1 Client Technologies 18

3.1.1 ReactJS 18

3.1.2 D3.js 18

3.1.3 How React.js and D3.js fit our project needs? 19

3.2 Server Technologies 19

3.2.1 Java 19

3.2.2 Microservices 19

3.2.3 Spring Framework 20

3.2.4 Spring Boot 20
3.2.4.1 Annotations 21
3.2.4.2 Spring CRUD 21
3.3 Data-Tier Technologies 21
3.3.1 MySQL 21
3.3.2 Flyway 22
Chapter 4. Project Design 23
4.1 Client Design 23
4.1.1 UI Screenshots 23
4.1.1.1 Login and Signup 23
4.1.1.2 Landing page 25
4.1.1.3 Issues in the sprint 26
4.1.1.4. Number of commits and repository downloads 27
4.1.1.5. Builds and Logs 30
4.2 Server Design 33
4.2.1 Class Diagram 33
4.2.2 Sequence Diagram 35
4.2.3 User Stories 37
4.2.4 Use Case Diagram 38
Chapter 5. Project Implementation 42
5.1 Client Implementation 42
5.1.1 DOM Development 42
5.1.2 Component Communication 42
5.1.3 Third-Party Library Integration 43
5.1.4 Setup Backend Communication 44
5.2 Server Implementation 44
5.2.1 Build Microservices 44
5.2.2 Setup Database Connection 45
5.2.3 Connect RESTful APIs 46
5.3 Data-Tier Implementation 46
Chapter 6. Testing and Verification 47
6.1 Testing 47
6.1.1 Unit Testing - Third-Party Interface testing 47

6.1.2 Integration Testing 50
6.1.3 System Testing 52
Chapter 7. Performance 53
7.1 Scalability 53
7.1.1 Client-side scalability 53
7.1.2 Docker Scalability 53
Chapter 8. Deployment and Maintenance 55
8.1. Amazon Web Services Deployment 55
8.2 Docker Containers 56
8.3 Version Code Management 58
Chapter 9. Conclusions and Future Work 60
9.1 Conclusion 60
9.2 Future Work 61

List of Figures

Fig 1. Project architecture 16
Fig 2. Login page 24
Fig 3. Signup page 24
Fig 4. Landing page 25
Fig 5. Task view 26
Fig 6. Active sprint details 26
Fig 7. Interactive pie chart and histogram for every sprint about status and issue 27
Fig 8. Branches details 27
Fig 9. Commit history of GitHub repository 28
Fig 10. Issue details 28
Fig 11. Pull requests 29
Fig 12. Commit history 29
Fig 13. Jenkins Jobs 30
Fig 14. Build history for a job 31
Fig 15. Build information 31
Fig 16. Build logs 32
Fig 17. Interactive pie chart and histogram for every Job about status and history 32
Fig 18. Class diagram 34
Fig 19. Sequence diagram 36
Fig 20. Use case authentication 38
Fig 21. Use case diagram 39
Fig 22. Integration testing 51
Fig 23. Test procedure flow chart 52
Fig 24. Black box testing 52
Fig 25. AWS console for all deployed microservices 55
Fig 26. AWS console for all deployed Linux servers 55
Fig 27. Jenkins docker container 56
Fig 28. Splunk docker container 56
Fig 29. GitHub docker container 57
Fig 30. Jira docker container 57
Fig 31. GitHub Code Repository 58
Fig 32. Project task board 59

List of Tables

Table 1. USER STORIES 37
Table 2. UNIT TESTING 47

Chapter 1. Project Overview

1.1 Introduction

Working in an industry, especially Software Development, we use a number of tools while working on projects. Each of them individually makes our work easier as we go along with our Software Development Lifecycle. The developers work in a sequential workflow, by using different tools at different stages of project development, which collaboratively improve work efficiency for the developers. However, one of the major concerns is to manually synchronize the project state in every tool, one by one. Whereas, it is possible to link all these different tools through an optimized workflow [1]. To understand how adding coordination between these tools can improve the workflow of the project cycle, let us discuss some significant tools used in software development lifecycle:

1.1.1 JIRA

As per Atlassian's official website, Jira is a very useful progress tracking and issue-ticketing tool for project management and agile software development. Project work is divided across the team members by creating tasks/stories which are tracked on a daily/weekly basis through sprint planning on a JIRA board. Usually, a custom workflow is generated by the board creator (for example, Blocked, On Deck, In Progress, Under Review, Validate, Done) to track the status of the tasks/stories. JIRA also has a comments section to make comments about any recent update. These tickets can be linked to the GitHub code so as to make an automated pull request from the JIRA board itself.

1.1.2 GitHub

Git is a Version Control System to maintain track of all iterations of changes made to the software and GitHub is the website that hosts Git [4]. Oftentimes, a JIRA ticket links to the GitHub code for which the task is created, so that it is easy to link to the exact file/directory relating to the task. However, a developer has to manually generate a pull request in order to get the most recent code and start working and then again push the code to the source repository in GitHub to keep up-to-date code [1]. This involves a lot of manual effort as one has to switch between different platforms and memorize the sequence of steps in order for the work to be updated properly.

1.1.3 Jenkins

Jenkins is used for continuous integration in software projects, i.e. continuous building and testing of software so that developers can easily make new changes in the projects and deliver bug-free software. This is achieved by new testing and deployment techniques continuously into the product development cycle [3]. So basically, Jenkins uses the source code from GitHub and builds the code.

1.1.4 Splunk

Every application or website creates some machine-generated data which is hard to understand and Splunk is the platform that makes this job easier by analyzing this data to make logical conclusions of this collected data. In Software development, Splunk is significant in the sense to store and analyze logs generated from software for bug solving or metrics. Though all these tools individually reduce the manual effort for developers, still a developer has to be the connecting hub for all these tools in order to coordinate them all. By adding communication between these different tools used, the whole process can be automated to improve efficiency and workflow [1]. Even a developer can escape the burden of logging in to different websites for the purpose of coordinating all the work of a project, whereas the same time can be utilized in delivering quality software.

1.2 Current State of the Art

Although there is no such platform that has automated the workflow between every tool used in the Software Development Lifecycle, there do exist some existing solutions interconnecting some of the tools, or tools in pairs communicating with each other. One such solution as posted by Acquia blog is the integration of JIRA, GitHub, Jenkins, and Slack [1]. Let us see how automated workflow between these tools reduces the developer effort: Tickets created in JIRA can reference the source code on GitHub and therefore pull requests can be generated from there itself. In the comments section of JIRA, test results can be posted, and the tests will be run automatically. Afterward, the code is pushed to the repository automatically. And finally, all this will be communicated to users through an automated Slack direct message. As we noticed, there was no need for a user to write pull/push requests from GitHub as JIRA tickets took care of that on their own.

Apart from this solution, there are other significant ways of connecting tools in pairs, such as JIRA and GitHub, Jenkins and GitHub (for continuous integration), JIRA, and Jenkins using JIRA plugin

(JIRA tickets reflecting the build status from Jenkins), Jenkins and Slack using a plugin (for build notifications to Slack). All these small integrations pave a way for our project to integrate on a higher level and connect more tools with each other than already existing solutions do.

1.3 Project Requirements

1.3.1 Functional Requirement

Functional Requirement defines the features and functionality of a product. It tells a lot about the ability and implementation that system will perform and scenarios it will handle.

1.3.1.1 Authentication

1. Authenticate employees/admin to use the application.
2. Enable admin to allow employees to access the system through registration.
3. Enable admin/employees to change the password if he/she forgets.

1.3.1.2 Administrative functions

1. Enable admin to create teams/groups of employees as a collaborative workspace environment.
2. Enable admin to restrict access of tools to any team/group.

1.3.1.3 Dashboard Functionality

1. Enable employees to toggle/access available tools within his/her created team.
2. Allow an authorized employee to utilize GitHub functionalities.
    a. Commits
    b. Merges
    c. Branching
3. Allow an authorized employee to monitor logs through Splunk.
4. Allow employees to build components through Jenkins.
5. Allow employees to monitor issues using JIRA.

1.3.1.4 Authorization levels

1. Restrict Employee accessibility to an authorized workspace.
2. Allow Manager to access multiple workspaces based on the projects assigned.

1.3.2 Non-functional Requirements

Non-functional requirements tell us about the system characteristics and behavior in a real-time environment.

1.3.2.1 Reliability

The system should be based on a minimal possibility of a service outage. This can be attained by the properties of a microservice architecture.

1.3.2.2 Security

The employees are separated in domain or conflict of interest should not be able to establish communication by means of the dashboard.

1.3.2.3 Usability

The system should be easy to handle any task with a minimal number of clicks.

1.3.2.4 Scalability

There should be a provision of scaling up the servers as the number of users increases.

Chapter 2. Project Architecture

2.1 High-Level Architecture

We have built a microservice architecture with Model, View, Controller (MVC) architecture pattern. Microservice architecture is best when a team is working independently on different modules/services that can be built and deployed independently. This will give us the freedom to deploy our code without interfering with the main pipeline.

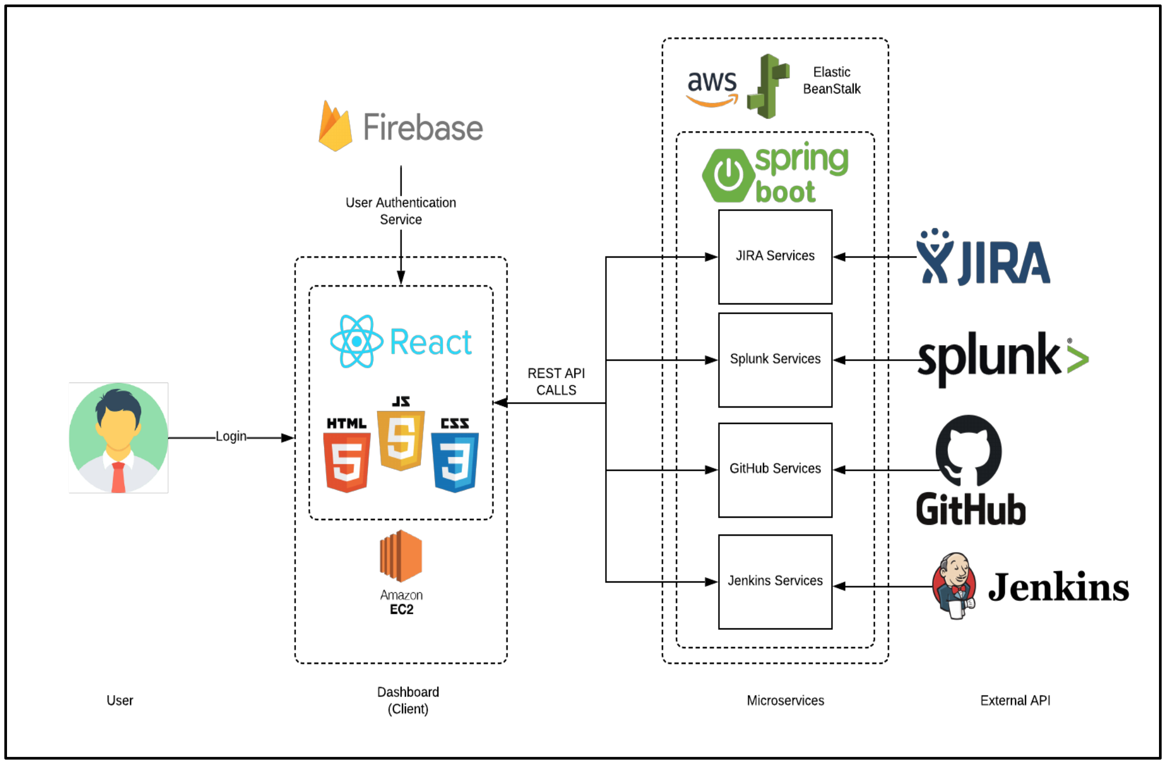


Fig 1. Project architecture

2.2 Description

For the hosting and deployment of our application, we have used Amazon Web Services (AWS) and Kubernetes for high scaling of the application. In Kubernetes, you need to provide your docker container and decide the number of replicas you want and scale your application accordingly. The Kubernetes handles the deployment, DNS resolving, health checks, and application version management. A newer version with small change can be deployed on the same server in a few minutes using Kubernetes.

The backend server is built in JAVA language with the Spring Boot Framework. Spring Boot makes it easy to segregate the code, separation of concern, and small setup-up code because of annotations based configuration. The Model and controller part are built on the Spring Boot (Server) side while the View is separately built on the client-side. Talking about the client, the frontend is written in JavaScript along with HTML and CSS using a React framework. The client is directly accessed by the user via a browser. As our application is mainly an integrated dashboard tool, we have also integrated some graph libraries like D3.js. The client will make HTTP REST calls to a server that will interact with our tools like JIRA, Jenkins, GitHub, and Splunk.

Chapter 3. Technology Descriptions

3.1 Client Technologies

3.1.1 ReactJS

A widely used JavaScript library in web development to build interactive parts for a single page website to large scale websites. This library has brought a lot of changes in web development helping to develop an application client-side and server-side using the same code making it a fast experience for the end-users. Its ability to provide stateful and reusable components makes it very popular. Components are like functions that take state and properties as inputs and return the description of User Interface. Due to this feature, they can be used to re-render themselves based on state change and can be reused where required. Also, as its name suggests, ReactJS reacts to state changes and reflects the User Interface changes to the Document Object Model. It provides the ease of changing the number of components once at a time.

3.1.2 D3.js

A JavaScript library widely used in data visualizations used to manipulate data based on changes in the data being fed. It stands for Data-Driven Documents. The graphical and visual representation created by this library uses HTML, CSS, and SVG. We can apply various transformations on the selected elements, for example, changing the font and size of the bar chart headings. Since the application that we are building involves various kinds of data and our requirements track those changes in the data, this library helped us in marking those changes promptly. It focuses on how to bind data to DOM elements. The most convincing reason to use

D3 for data visualization is that D3 is written in JavaScript which means its functional style will help use the code and scale itself with the increase in data. Its robust characteristic helps us manipulate and style our data accordingly to make it more interactive.

3.1.3 How React.js and D3.js fit our project needs?

According to our project needs, we need a dashboard to display information which is better understandable like the number of commits done in a certain period. React when combined with D3 makes it reusable and integration easy. Using state and props properly in React helps update the charts. React also helped in adding responsiveness to the charts.

3.2 Server Technologies

3.2.1 Java

We are using Java 8 for this application. Java is an object-oriented language commonly used to build enterprise styled applications. Java 8 comes up with new utilities like Java Streams which help us to process large data for our dashboard. We have written multiple lambda functions on the server-side which efficiently communicate with clients.

3.2.2 Microservices

It enables us with the continuous delivery and deployment of large, complex applications. The service is relatively small and so is easier to understand and change. In our team, multiple developers are contributing to writing server code. The microservices are smaller and faster to test. The services can be deployed independently. Each developer can develop, test, deploy, and scale their services independently of all of the other members in a project.

3.2.3 Spring Framework

Spring is a Java-based framework mainly used to build enterprise systems. It came as a better alternative to the Java 2 Platform Enterprise Edition (J2EE). It comes up with a large number of helper classes and modules which can be used in application development.

The main reason for using Spring in our project is because spring makes use of concepts like Inversion of Control (IoC) and Dependency Injection. The IoC helps to transfer control of objects to a container or a framework from classes itself. It helps us to integrate our dependency once and write code afterward without instantiating and injecting the dependencies.

3.2.4 Spring Boot

It gives the pre-starter dependency helps to configure and form a base of application very easily and quickly. It helps in better configuration with the help of AutoConfiguration.

@SpringBootApplication annotation is itself a combination of three annotations @Configuration, @EnableAutoConfiguration, and @ComponentScan that quickly set up the development environment.

3.2.4.1 Annotations

Spring applications can be configured in three ways - XML, Java, or Annotation based. We are using Annotations as they are very short, hence reducing the code complexity and number of lines. Annotations are self-descriptive and easy to use.

3.2.4.2 Spring CRUD

To communicate with the data layer, we will be requiring some ORM that acts as middleware between the server layer and the data layer. We are using Spring CRUD, a spring project implementation, that comes with preset transactional database methods and interfaces which can be overridden to serve our use case.

3.3 Data-Tier Technologies

3.3.1 MySQL

MySQL is a standard SQL database system. It provides a relational database management system that helps us to easily create the relation between the entities (tables) and pull the data from multiple tables using joins and views. Our data is mostly structured, where we are storing the user information for authentication and authorization of the user for the application. MYSQL is ideal for small storage systems and it is easy to implement and deploy. MySQL comes with two properties which we need at the priority for our system - Consistency, and Availability. We want our system to always be consistent in terms of data and always be available to the user. MySQL will take care of ACID properties which can aid in transactional queries.

3.3.2 Flyway

Flyway is a version control management service for the database. It manages the database snapshots which stores the schema of the database at any time. It also validates the schema, with the most recent snapshot of the database.

Chapter 4. Project Design

We are using the Spring Boot framework for Java backend servers. Sprint Boot brings all spring projects under an umbrella to create a single application that can be executed just by the .jar file. We are using Maven to build the project and maven repository for including external and 3rd party libraries and jar in our application. Spring boot eliminates the use of lengthy XML configurations and makes use of Java annotations.

For the design of our frontend dashboard, we are using the React MVC framework for JavaScript and HTML enabled web pages.

4.1 Client Design

4.1.1 UI Screenshots

Below are the Integrated Tools Platform UI screenshots:

4.1.1.1 Login and Signup

The login or sign up should be with the GitHub credentials in order to see the workflow of projects which are on GitHub. Using the git repository associated with the project, further JIRA, Jenkins, and Splunk integration and flow can be established.

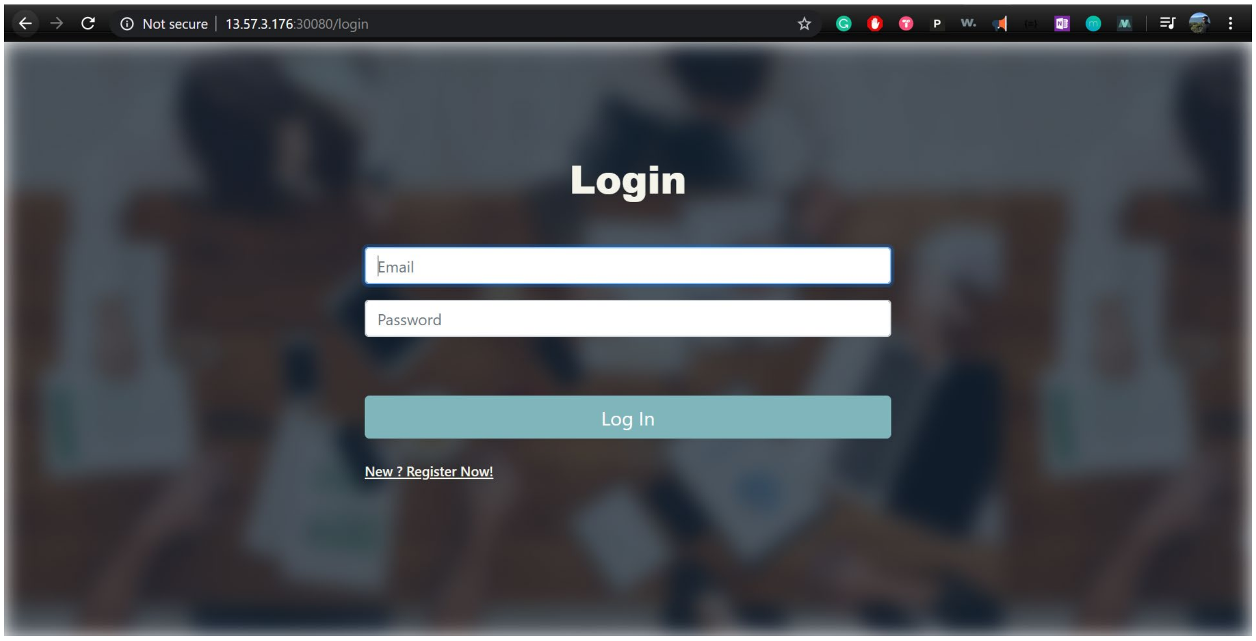


Fig 2. Login page

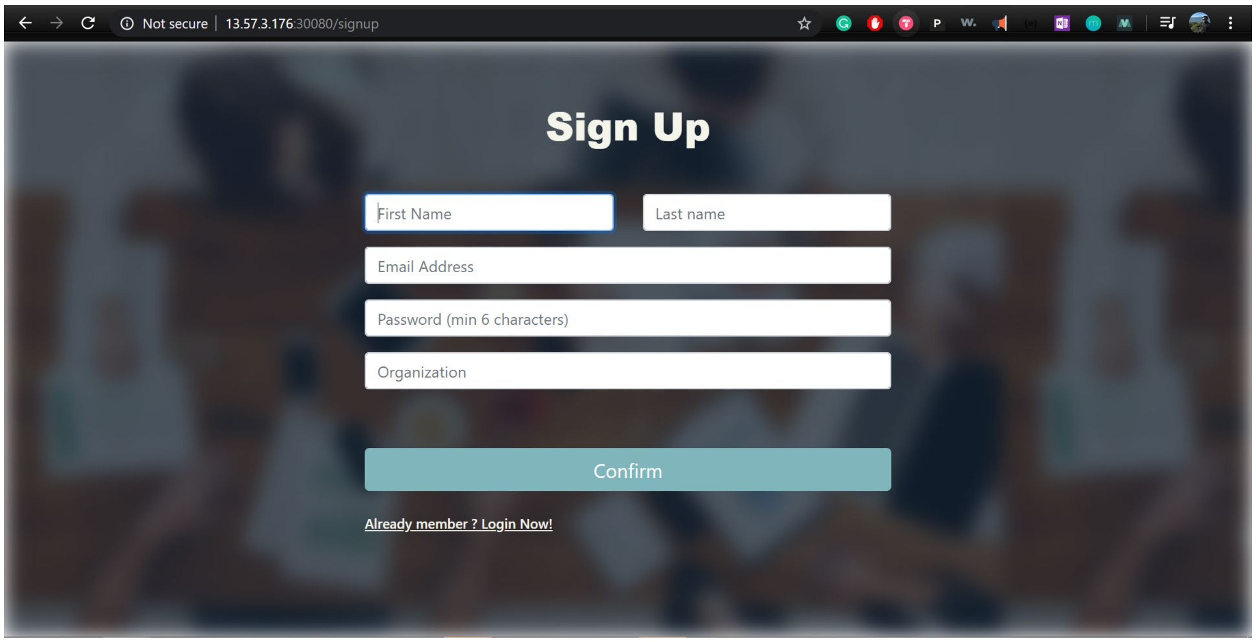
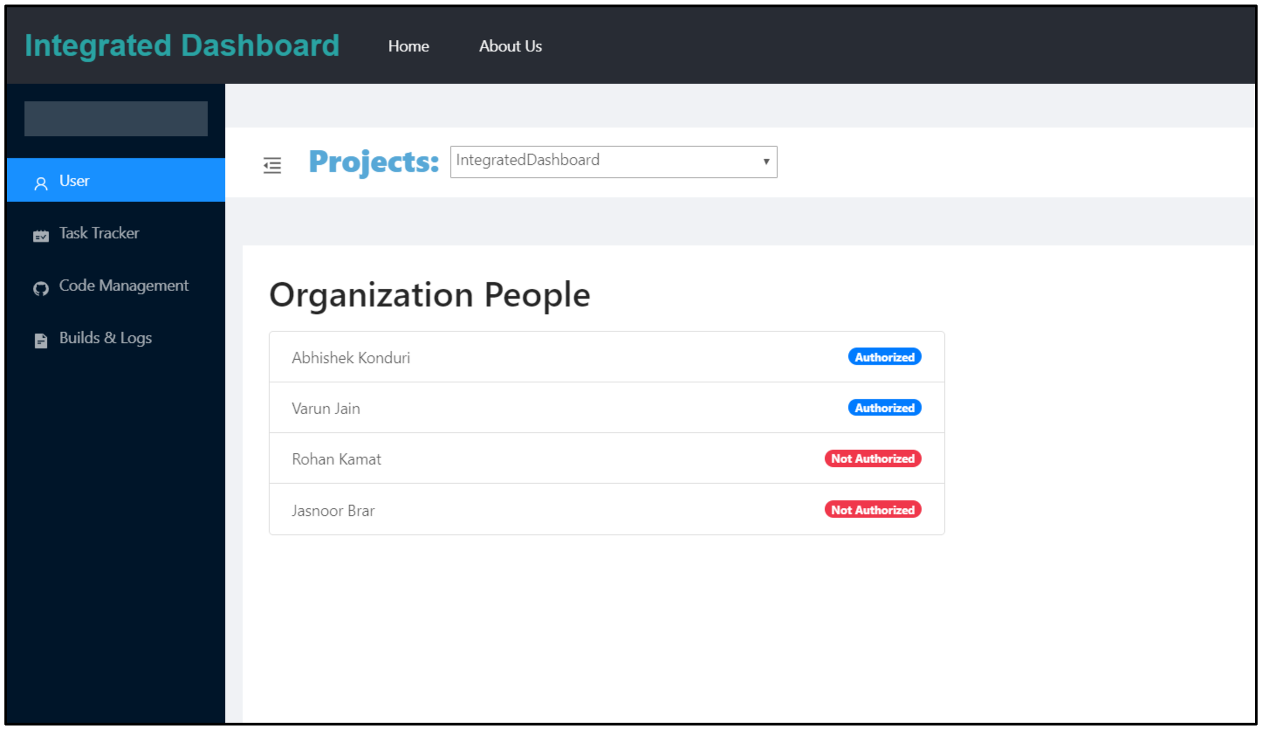

Not secure | 13.57.3.176:30080/signup
Sign Up
First Name
Last name
Email Address
Password (min 6 characters)
Organization
Confirm
Already member ? Login Now!

Fig 3. Signup page

4.1.1.2 Landing page

The landing page displays four boards: JIRA, GitHub, Jenkins, and Splunk. The details on these boards are with respect to a particular project on GitHub of the user who is logged in. The JIRA board displays the latest active sprints of the projects, i.e. links to the active sprints. These sprints contain source code links to the GitHub repository so that the task in JIRA is automatically connected to the GitHub code. The GitHub board displays the latest issues created in the git repository of the project, so the user can quickly go to the issues and resolve or close them. Jenkins' board displays the recent build results generated for the project. It is automatically linked to the GitHub repository, takes the code from there, and generates a build. The last board, Splunk displays the log reports. These log reports are generated from the build results provided by the Jenkins tool.



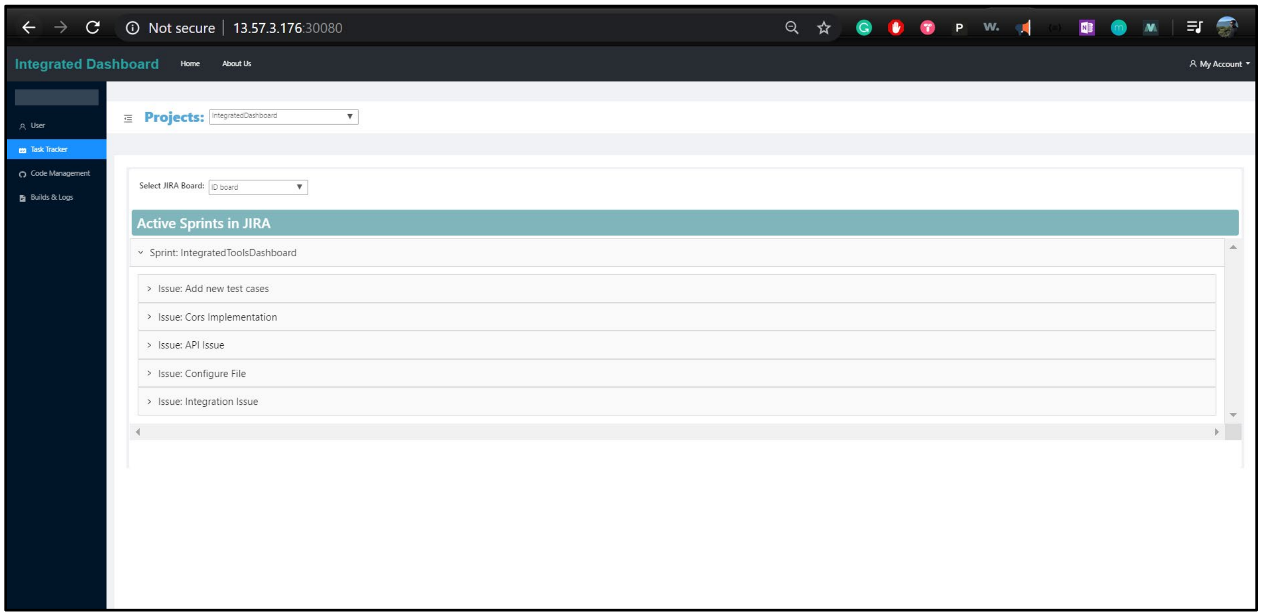
Organization People

Abhishek Konduri	Authorized
Varun Jain	Authorized
Rohan Kamat	Not Authorized
Jasnoor Brar	Not Authorized

Fig 4. Landing page

4.1.1.3 Issues in the sprint

The 'Issues in Sprint' of the JIRA board displays a detailed view of issues in each sprint. The user has the option to toggle between JIRA boards for a particular project.



Sprint: IntegratedToolsDashboard

Issue: Add new test cases

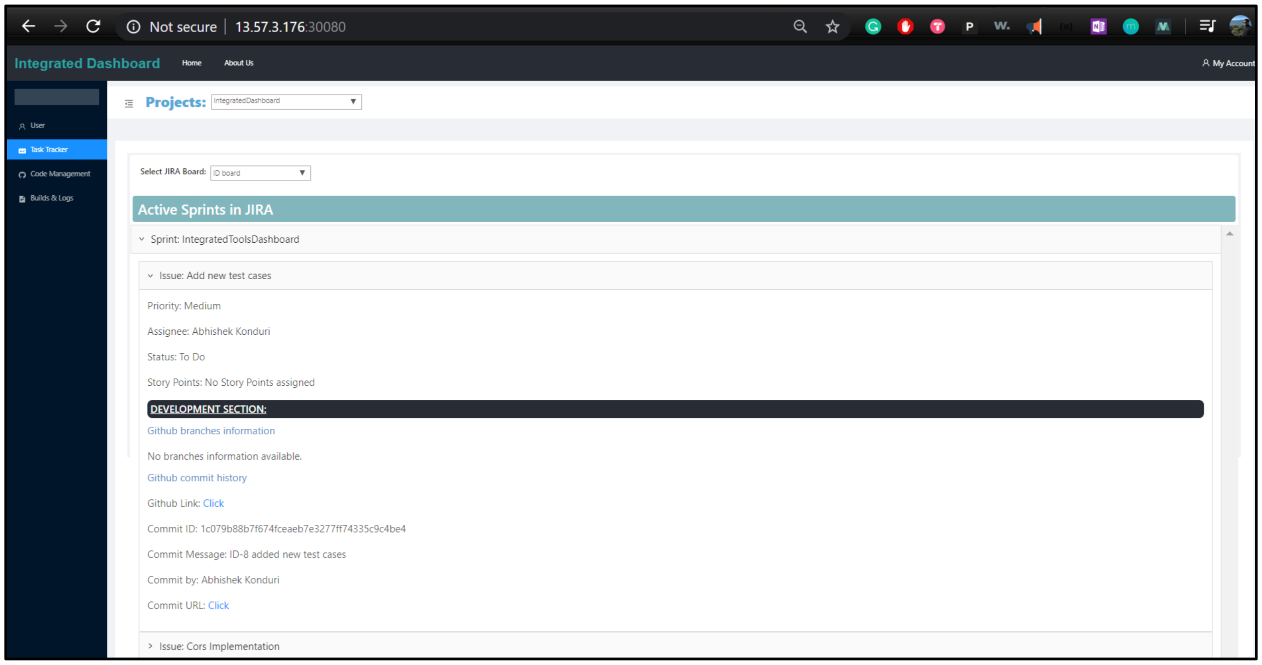
Issue: Cors Implementation

Issue: API Issue

Issue: Configure File

Issue: Integration Issue

Fig 5. Task view



Sprint: IntegratedToolsDashboard

Issue: Add new test cases

Priority: Medium

Assignee: Abhishek Konduri

Status: To Do

Story Points: No Story Points assigned

DEVELOPMENT SECTION:

Github branches information

No branches information available.

Github commit history

Github Link: Click

Commit ID: 1c079b88b7f674fceaeb7e3277ff74335c9c4be4

Commit Message: ID-8 added new test cases

Commit by: Abhishek Konduri

Commit URL: Click

Issue: Cors Implementation

Fig 6. Active sprint details

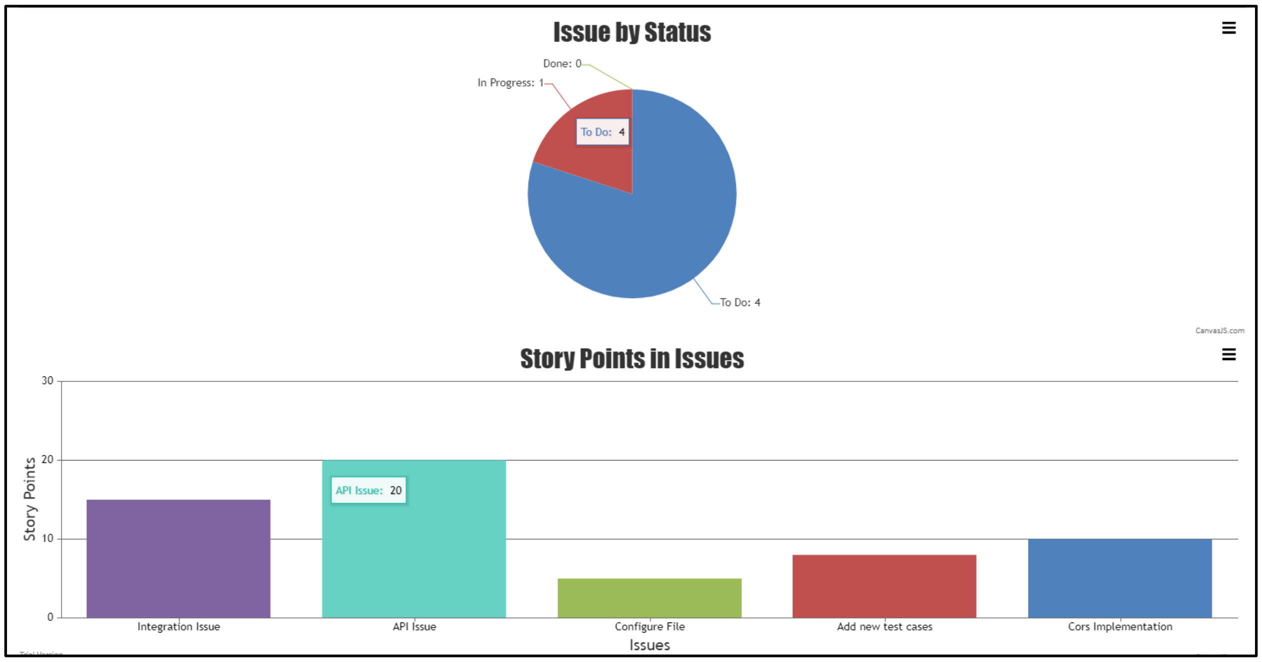
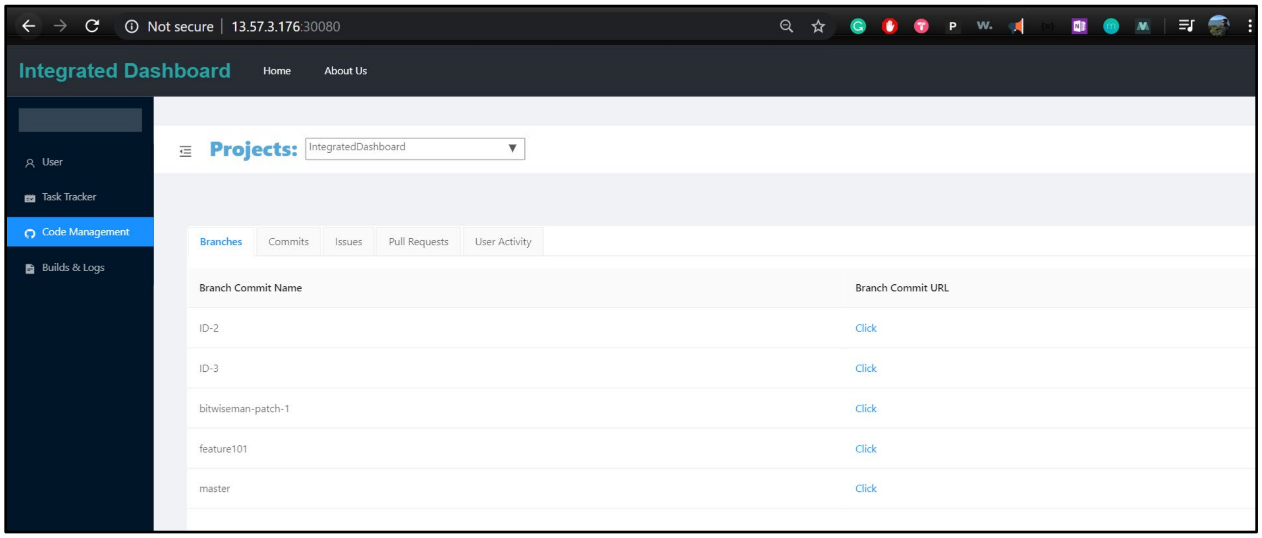


Fig 7. Interactive pie chart and histogram for every sprint about status and issue

4.1.1.4. Number of commits and repository downloads

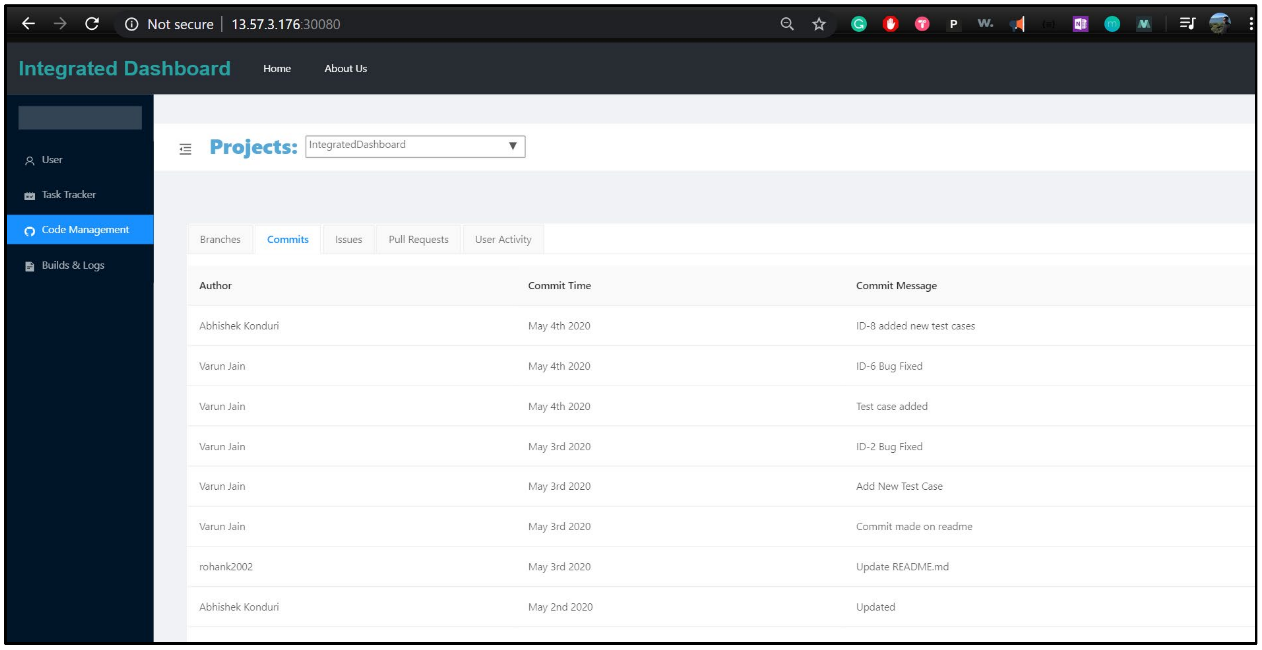
This is a statistical wherein the user can get detailed overview of the repository. There are specific windowpanes for each section. The following figures show each of the view. The user can also view the graph providing the user's contribution towards the repository.



Branch Commit Name	Branch Commit URL
ID-2	Click
ID-3	Click
bitwiseman-patch-1	Click
feature101	Click
master	Click

Fig 8. Branches details

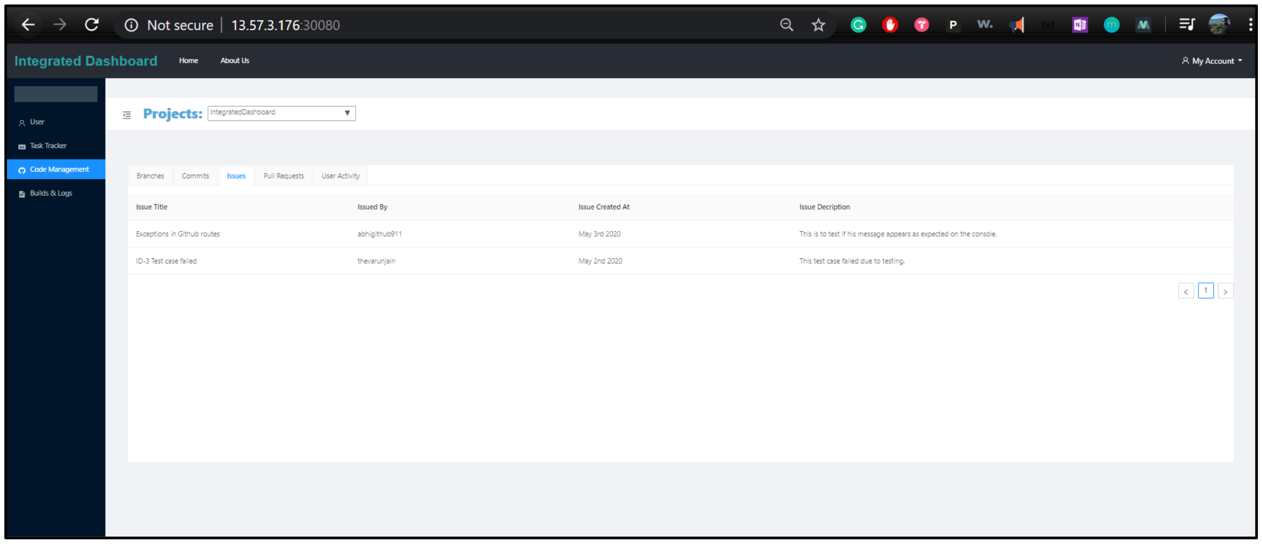
The user can view the commit details of the project by chosing the project name from the toggle bar.



Author	Commit Time	Commit Message
Abhishek Konduri	May 4th 2020	ID-8 added new test cases
Varun Jain	May 4th 2020	ID-6 Bug Fixed
Varun Jain	May 4th 2020	Test case added
Varun Jain	May 3rd 2020	ID-2 Bug Fixed
Varun Jain	May 3rd 2020	Add New Test Case
Varun Jain	May 3rd 2020	Commit made on readme
rohank2002	May 3rd 2020	Update README.md
Abhishek Konduri	May 2nd 2020	Updated

Fig 9. Commit history of GitHub repository

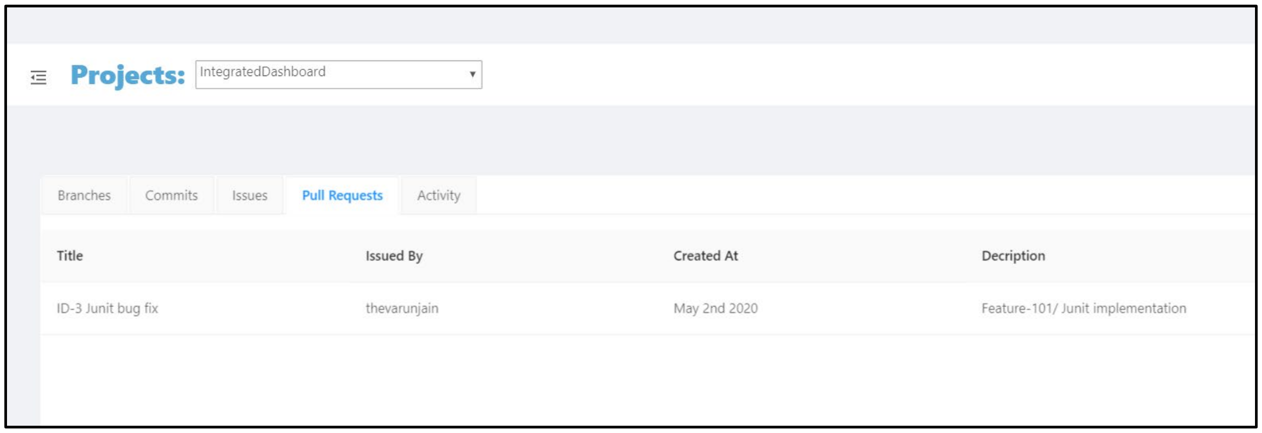
The repositories can have multiple issues, the issues are linked to the JIRA sprint. The user can get details of the issue pertaining to a particular project.



Issue Title	Issued By	Issue Created At	Issue Decription
Exceptions in Github routes	abhigithub911	May 3rd 2020	This is to test if his message appears as expected on the console.
ID-3 Test case failed	thevarunjain	May 2nd 2020	This test case failed due to testing.

Fig 10. Issue details

There can be multiple people working on a single project. This requires each person to pull the code into his/her repo. The user can check out the pull requests details to identify the people working on the project.



Title	Issued By	Created At	Decription
ID-3 Junit bug fix	thevarunjain	May 2nd 2020	Feature-101/ Junit implementation

Fig 11. Pull requests

A visual cue is very important in a dashboard, which makes concepts lucid. The user activity graph denotes the commit activity of the user for a particular project.

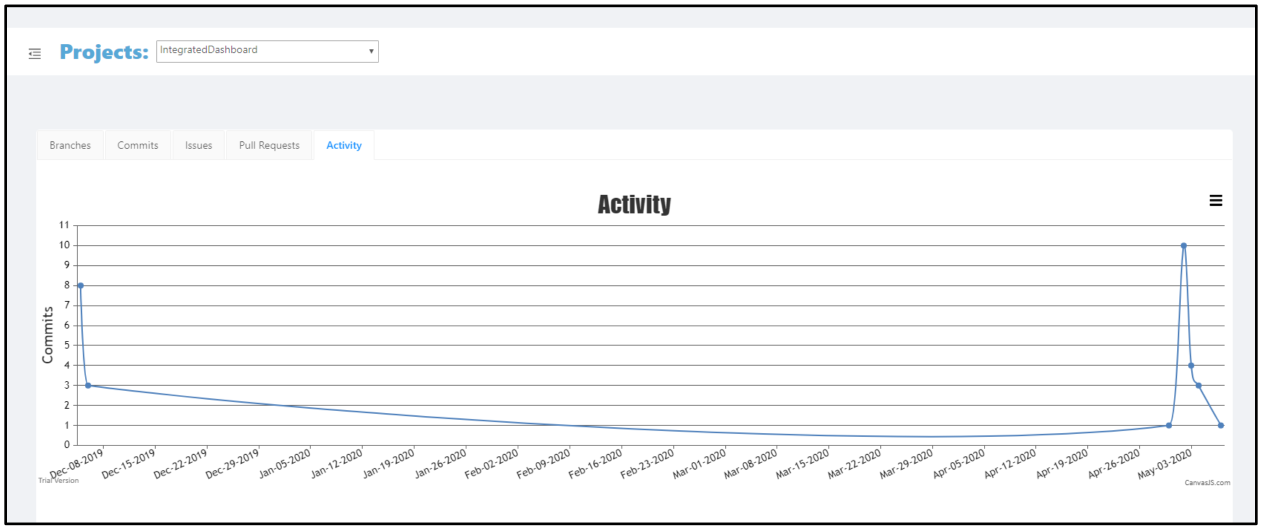
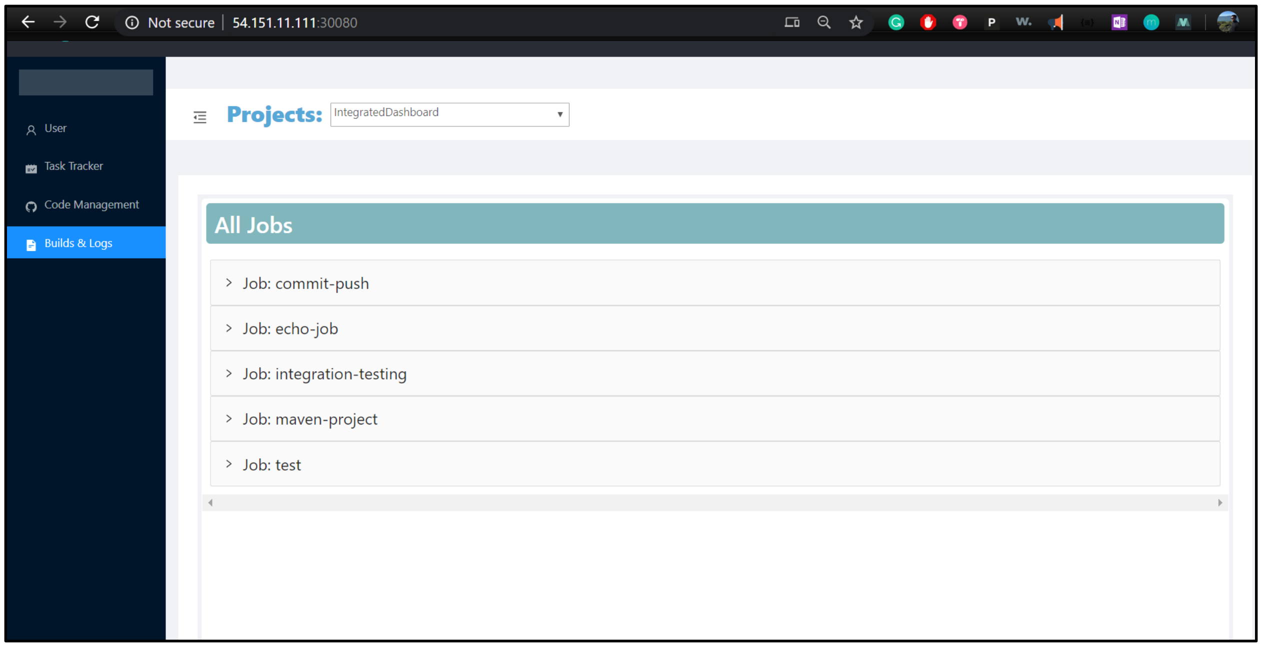


Fig 12. Commit history

4.1.1.5. Builds and Logs

The builds and logs section provides the information about Jenkins jobs and logs from Splunk.



Job: commit-push
Job: echo-job
Job: integration-testing
Job: maven-project
Job: test

Fig 13. Jenkins Jobs

A user would be able to view all builds that have been done for a job. An embedded build now button has also been added in case a user wants to build it manually. Also, if the user wants to see the actual job, hyperlinks are provided with every job.

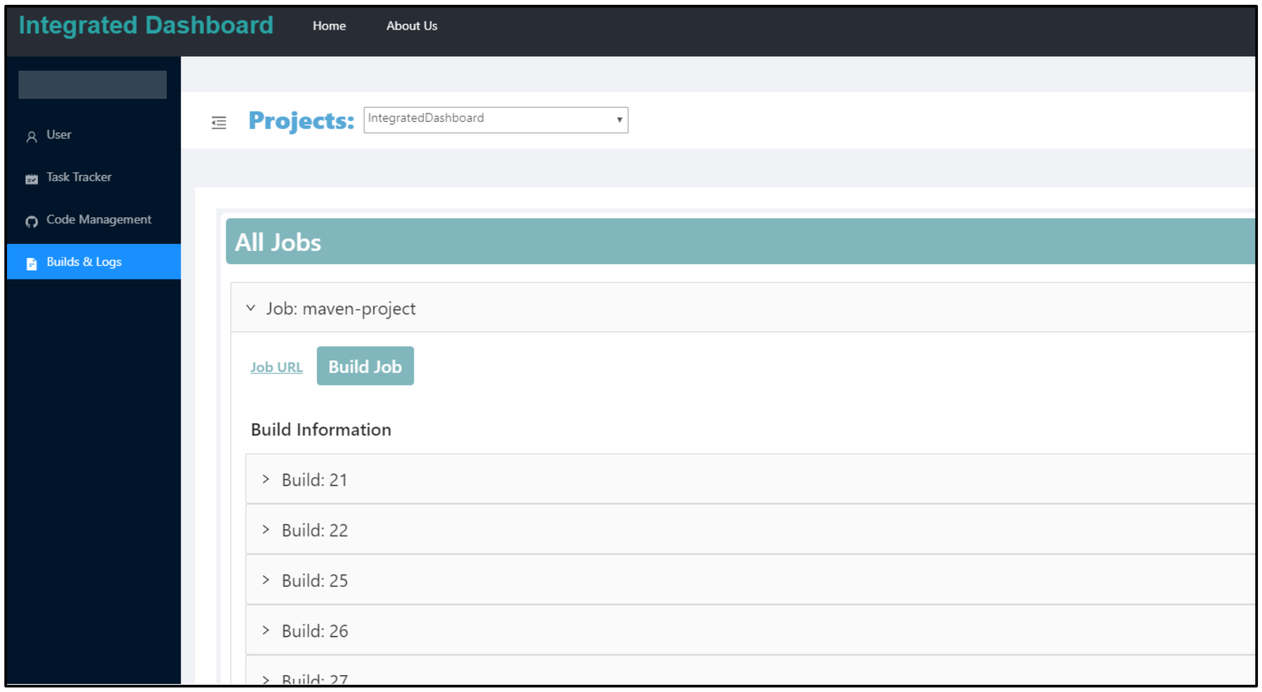
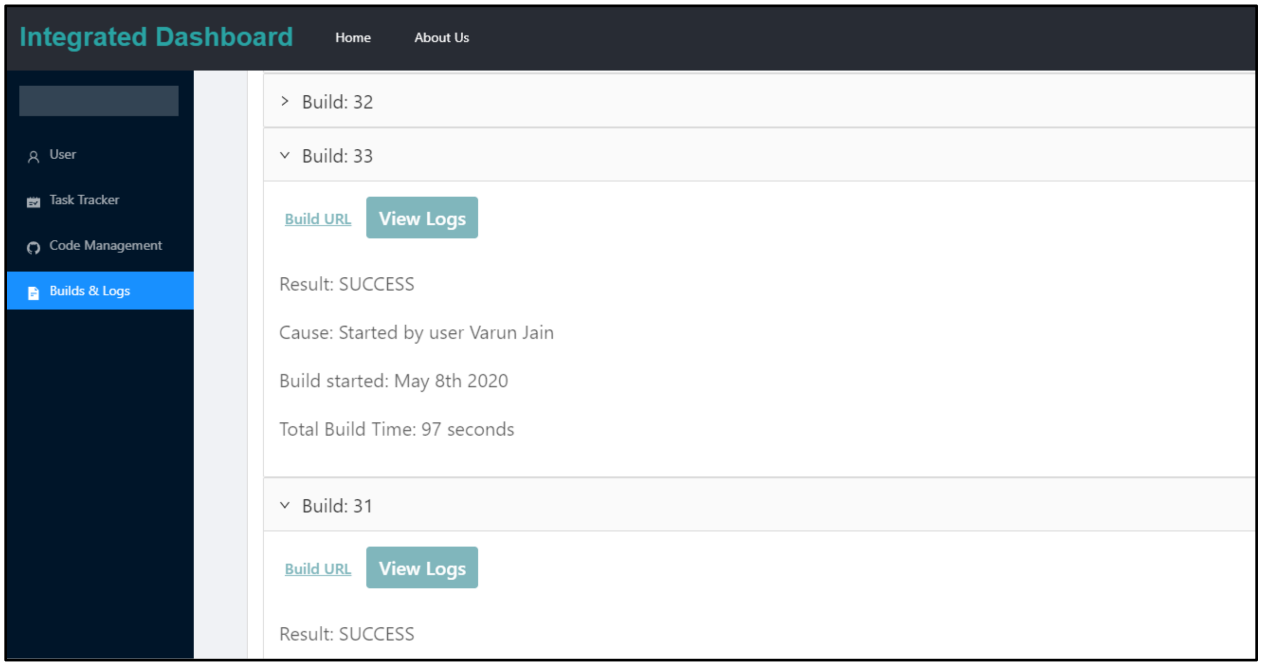


Fig 14. Build history for a job

A user can get information like status, cause of the trigger when a build was initiated and time it took. Also, if a user wants to see the actual build, hyperlinks are provided with every build.



Result: SUCCESS

Cause: Started by user Varun Jain

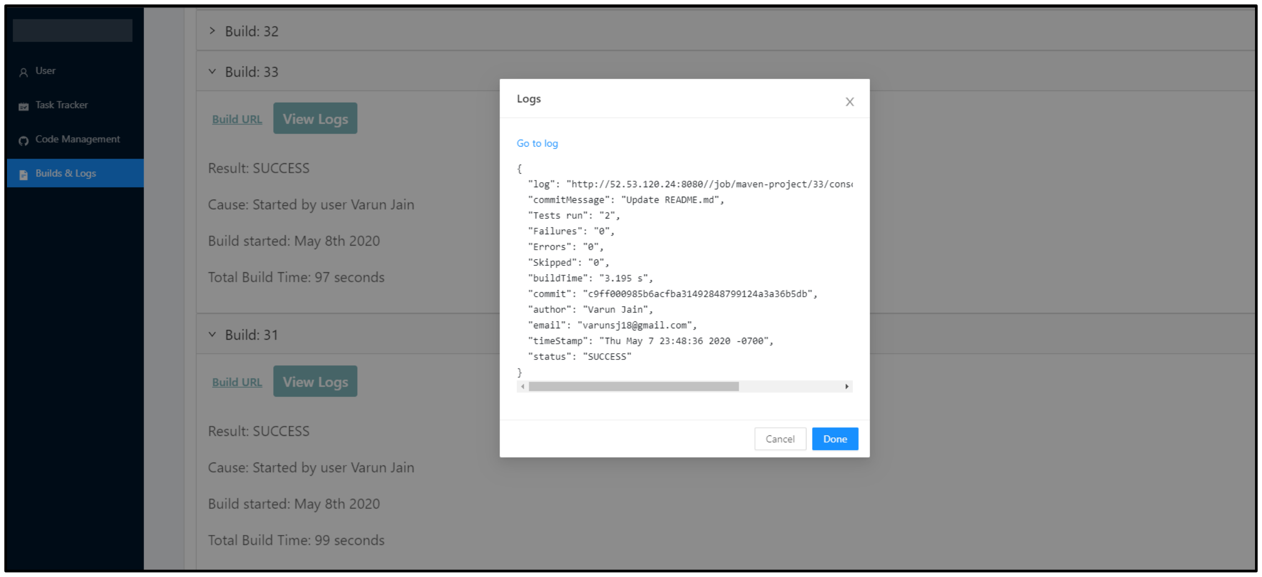
Build started: May 8th 2020

Total Build Time: 97 seconds

Result: SUCCESS

Fig 15. Build information

The user can view build logs directly from dashboard, the fact that logs are taken from Splunk server is abstracted from the user.



Result: SUCCESS

Cause: Started by user Varun Jain

Build started: May 8th 2020

Total Build Time: 97 seconds

Result: SUCCESS

Cause: Started by user Varun Jain

Build started: May 8th 2020

Total Build Time: 99 seconds

```
{
  "log": "http://52.53.120.24:8080//job/maven-project/33/cons
  "commitMessage": "Update README.md",
  "Tests run": "2",
  "Failures": "0",
  "Errors": "0",
  "Skipped": "0",
  "buildTime": "3.195 s",
  "commit": "c9ff000985b6acfba31492848799124a3a36b5db",
  "author": "Varun Jain",
  "email": "varunsj18@gmail.com",
  "timeStamp": "Thu May 7 23:48:36 2020 -0700",
  "status": "SUCCESS"
}
```

Fig 16. Build logs

A visual graph per job is created which denotes number of failed builds to successful builds. The timeline of builds can also be viewed.

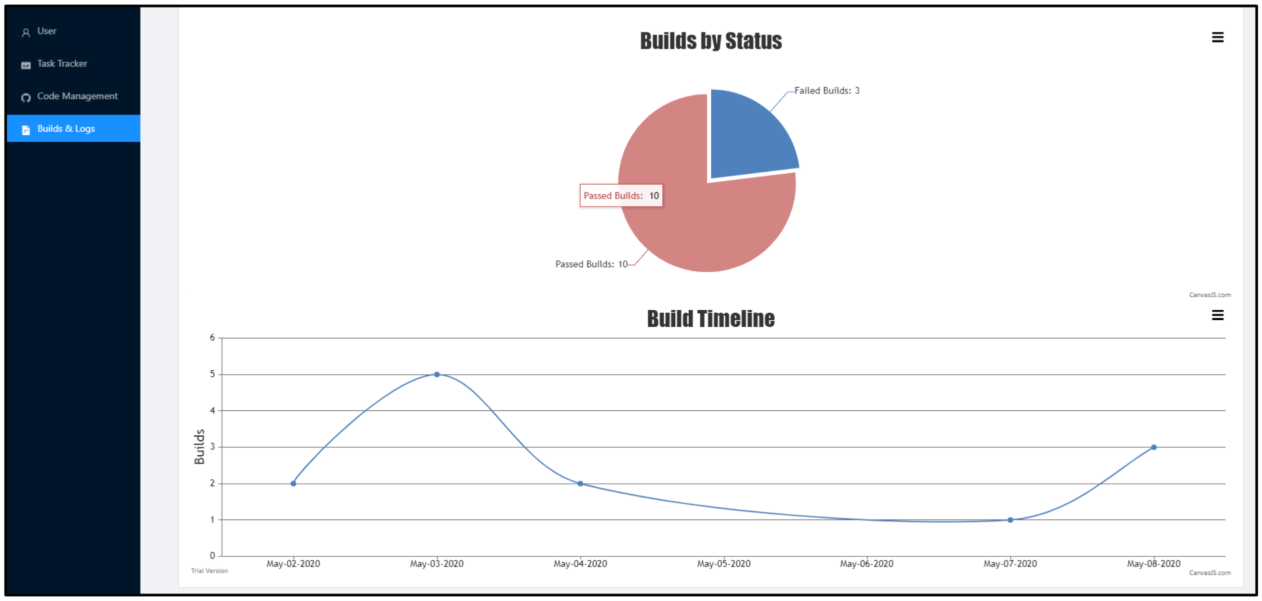


Fig 17. Interactive pie chart and histogram for every Job about status and history

4.2 Server Design

4.2.1 Class Diagram

The class diagram below depicts our entire project class structure. Our project is dependent on 4 third party services, namely JIRA, Jenkins, GitHub, Splunk. SDK (Software Development Kit) for all services are available, which makes it easy for the usage of Spring Boot framework. Each class has a “projectId” which is initiated by the user. JIRA is a tool used by software developers to analyze the project development cycle through the status of the sprint. JIRA has 3 methods which are getSprint() which returns a given particular sprint, getIssue() which returns the issue based on issueId and getStoryPoints() which returns the story points for individual stories based on a particular id.

The next class is Jenkins, which is a tool that enables software developers to build applications. The method startBuild() starts to build a given particular job. The job which is being built can fail as well, so getBuildStatus() is a method used to get the status of the build process, that is whether it has passed or failed. A particular sprint can have multiple jobs, so getJobs() returns all the jobs for a particular spring. GitHub class has the basic functional methods which fetch the commits, issues, pull requests for a given repository. Splunk is a service that detects any server-side failures and analyzes the logs. The methods getLogs() returns the entire logs for the server, and the run query is a command which ensures case-specific logs.

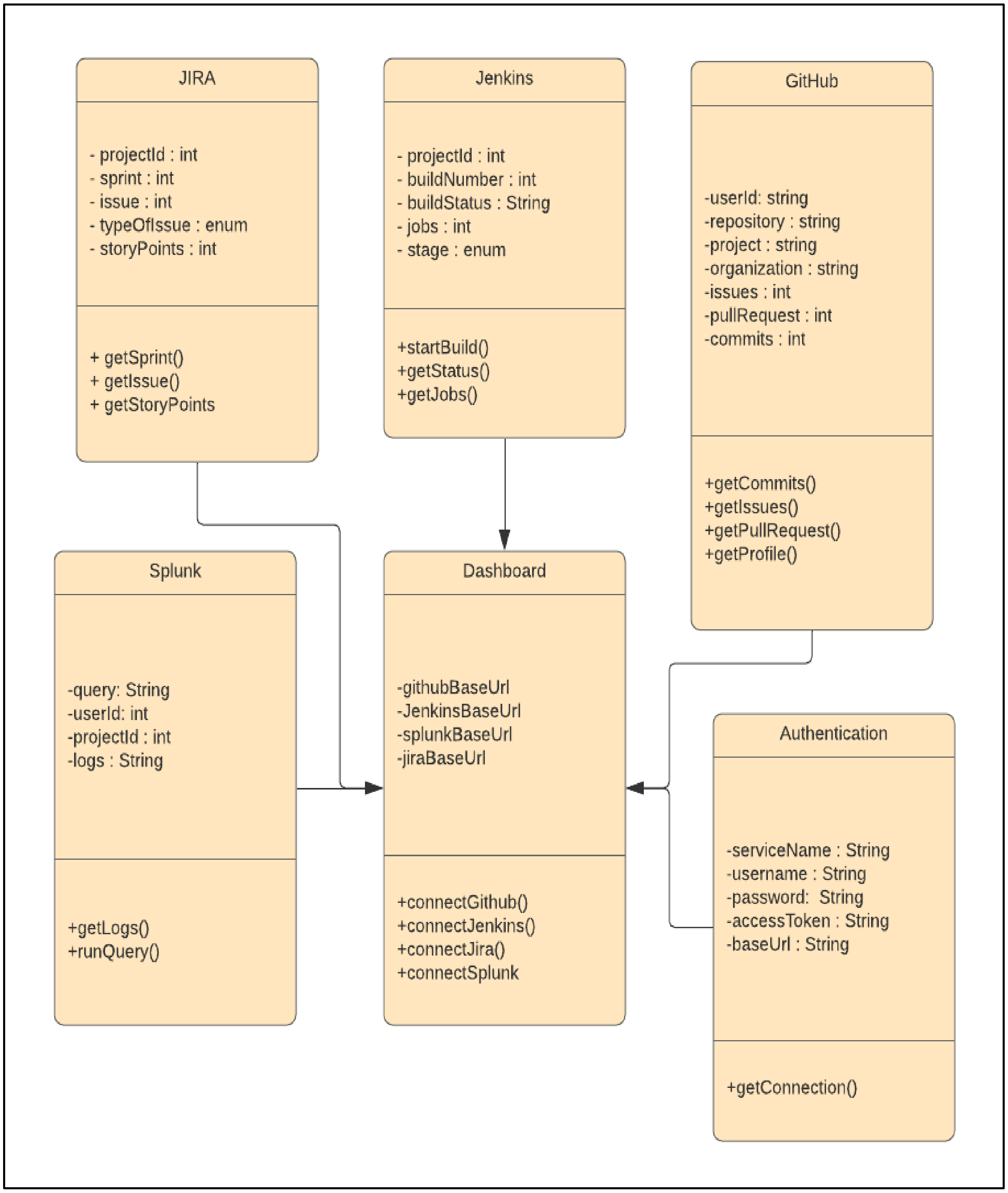


Fig 18. Class diagram

4.2.2 Sequence Diagram

The sequence diagram explains the sequence of events that occur within our application. The user initializes the project with a JIRA sprint from the dashboard. This initialization of sprint in JIRA will result in the creation of a GitHub repository. The manager/developer will then get access to the number of commits/ issues etc. The next step of development is the build process, Jenkins takes care of it. The GitHub commits can trigger the builds in Jenkins. The logs generated will be reflected in the logs from the Splunk server. This is the entire workflow is described using the diagram.

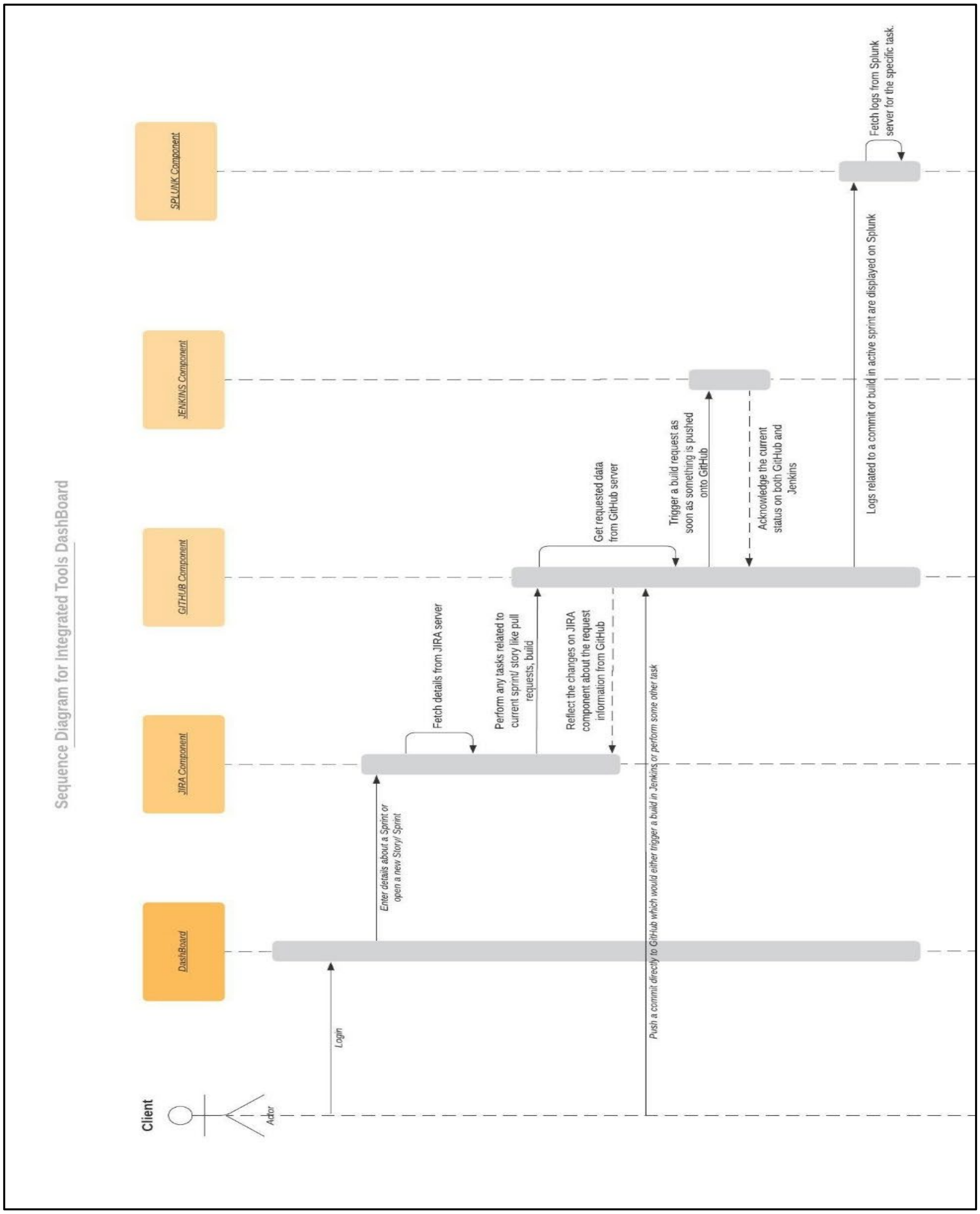

Sequence Diagram for Integrated Tools DashBoard
Client
Actor
DashBoard
JIRA Component
GITHUB Component
JENKINS Component
SPLUNK Component
Login
Enter details about a Sprint or open a new Story/ Sprint
Fetch details from JIRA server
Perform any tasks related to current sprint/ story like pull requests, build
Reflect the changes on JIRA component about the request information from GitHub
Push a commit directly to GitHub which would either trigger a build in Jenkins or perform some other task
Get requested data from GitHub server
Trigger a build request as soon as something is pushed onto GitHub
Acknowledge the current status on both GitHub and Jenkins
Logs related to a commit or build in active sprint are displayed on Splunk
Fetch logs from Splunk server for the specific task.

Fig 19. Sequence diagram

4.2.3 User Stories

Table 1. USER STORIES

Actor	User Story
Admin	1. As an admin, I want to add employees to the team(workspace). 2. As an admin, I want to create a team(workspace) that can have tools accessibility.
Employee	1. As an employee, I want access to the profile of any employee from my team. 2. As an employee, I want access to any possible tool assigned to my workspace. 3. As an employee, I want to get all the information from all the services I have access to.
Product Manager	1. As a product manager, I want to access the logs through Splunk. 2. As a product manager, I want to monitor issues through JIRA software. 3. As a product manager, I want to know the status of builds through Jenkins.
Developer	1. As a developer, I want to make a new commit through the dashboard. 2. As a developer, I want to check the issues assigned to me through JIRA.
Tester	1. As a tester, I want to view the logs through the dashboard available through Splunk API. 2. As a tester, I want to monitor the issues through JIRA. 3. As a developer, I want to create a new issue.

4.2.4 Use Case Diagram

Below is the use case diagram for authentication and profile viewing. The employee/developer and admin can log in through the system and view profiles. The use case diagram depicts the functioning of the system. The use case diagram just specifies the expected behavior, unlike the class diagram which actually explains the flow of how exactly the behavior has occurred.

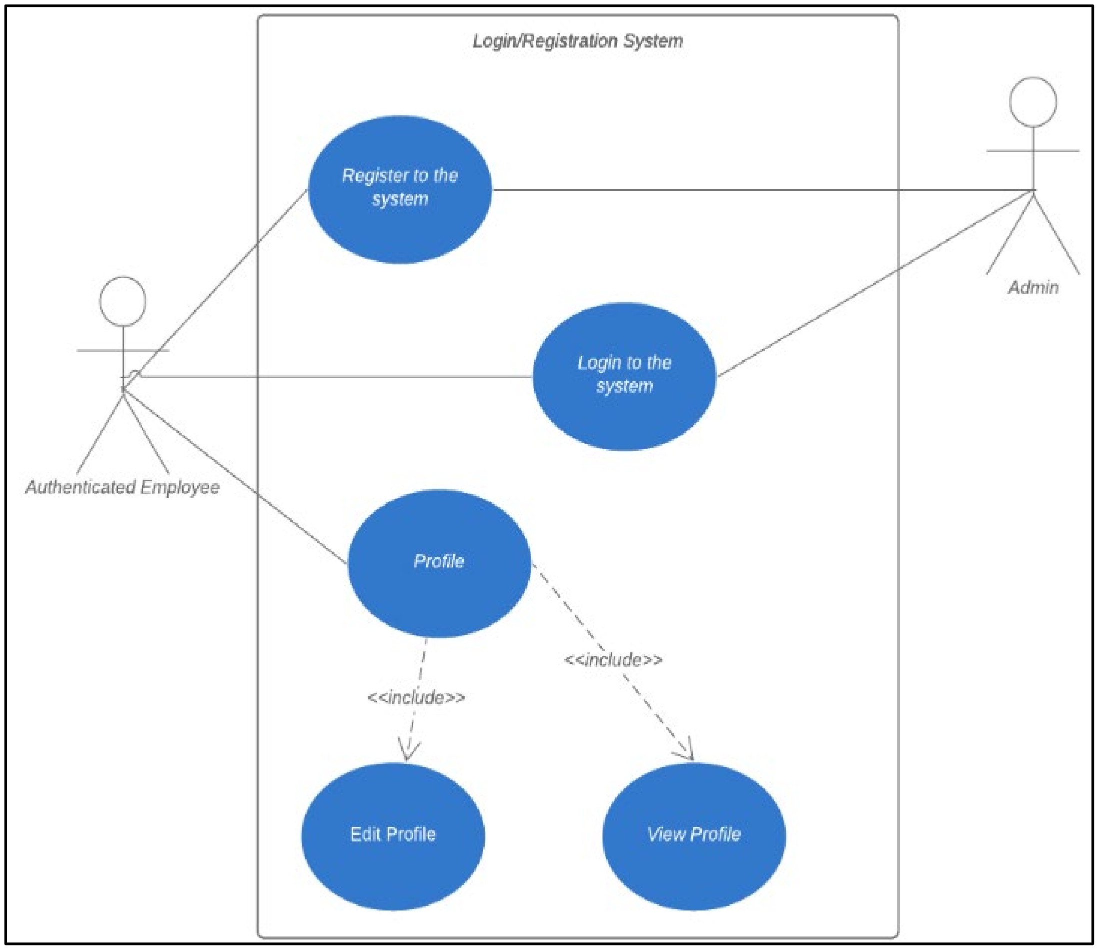


Fig 20. Use case authentication

Each project is initiated with 4 services which are Jenkins, Splunk, GitHub, JIRA. The services communicate with each other using in-built plugins and mediator APIs created by us. The diagram depicts the relation between the mediator services and the main backbone dashboard service. The two actors in concern are admin and the authenticated employee.

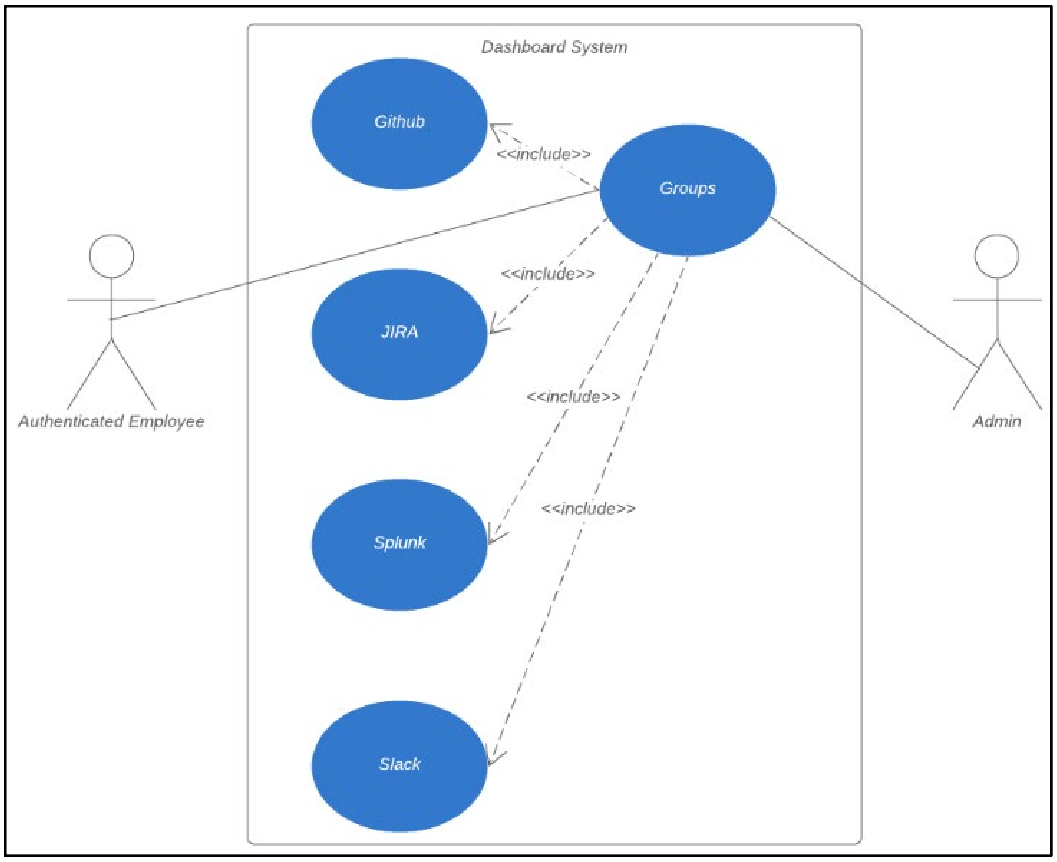


Fig 21. Use case diagram

JIRA

It is used in the agile development of Software 4 Development Life Cycle Management and congruity. By using the intuitive dashboard, the JIRA embraces the following functionalities:

a) It allows assigning user stories to an epic and helps to locate the story points associated with every story.

b) By using the dashboard, a user can easily move user stories from the existing sprint to backlog and separate sprint.

c) The dashboard allows the user to quickly change the user story status between ”To Do”, ”In Step” and ”Completed”.

d) It displays how many stories a user is assigned.

GitHub

It is a version control system but also distributed source code governance is usually the primary mechanism for most software companies to use. The dashboard interface integrates the programming interface of the GitHub application to support frequent tasks usually performed by programmers including:

a) We can also find the number of open issues or pull requests pending in a repository and details such as title, assignor, comments, and sub-task to be completed.

b) A project manager can access any number of latest commits and developers associated with any repository.

c) Fetch the most number commits and the contributor who has made the maximum contribution. We can display the theses on the basis of two metrics, the first number of commits, and the second number of lines of codes.

d) The dashboard will allow visualizing the pie-chart to give a project structure view at a glance.

Jenkins

It is an open-source development server that is used for continuous integration and continuous delivery. The dashboard application will support the following functionalities of the Jenkins Server:

a) The dashboard will provide the functionality to begin a construct of a pipeline from the source code of GitHub.

b) A user can easily find the failed builds. From the dashboard only we can retrigger all of the failed builds and can monitor the progress of it.

c) The dashboard displays all of the configured slave machines associated with the master machine and can display the status of all machines, pending builds, and current running status.

Splunk

A tool used to analyze the logs and to aggregate data from various sources generated by machines. The Splunk integrated interface includes the following log analysis concerns:

a) The dashboard helps us to get the Splunk app status and the number of apps running on it.

b) The information and data related to Jenkin’s number of builds failed between the time period provided by the client and the user can also be visualized.

Chapter 5. Project Implementation

5.1 Client Implementation

5.1.1 DOM Development

We started with initially making a setup of Integrated Development Environments. We imported the right project structures with all of the components modularized. Setting up and installing all the necessary third-party libraries.

Once the initial setup was done, we created Project Structure for frontend using ReactJS. A clear project structure helped us in outlining the components we wanted to work on. According to the component structure, we started with developing the frontend components which gave us an idea of how the dashboard would look. We then designed and developed a basic frontend web page according to our use case.

5.1.2 Component Communication

Once the DOM implementation was done, we started with making the individual components work. We developed the front end of our application using React and there are different components developed in React which have to be communicated with each other to give a complete flow to the application. There are different components on the main page of the integrated dashboard like the graph components, the analytics component, the component which shows information about a task. All of these components are to be made in Sync with each other. React creates props for all of such components and we made all of these components to communicate with each other for it to give a synchronized look.

There are two basic ways to make the component communication possible in react which are

1. Render props/props.

2. Context

5.1.3 Third-Party Library Integration

One of the main features of our application is to show analytics and graphs about the status of the ongoing tasks in a company/project and other details like a number of different tasks in GitHub, stories assigned to a particular team member in Jira, etc.

For all of these uses cases to work we had to make use of some external graph libraries which solves the purpose of displaying all of the information in graphical format using pie charts, bar graphs, etc.

We had to integrate third-party libraries in our client application for the analytical part to work. We made use of D3.js which is a library for graphs. D3.js is mainly used for producing dynamic, interactive data visualizations in web browsers. So in our use case, we implemented D3.JS graphs library to populate data for a specific time period.

5.1.4 Setup Backend Communication

We had a bunch of Microservices running in the backend with their respective databases. There are REST APIs for each of the use cases. These Rest APIs provide all of the live dynamic data from different software applications like GitHub, Jira, Jenkins, Splunk. For all of the dynamic data from these applications to work and display on the front end, we had to make communication between the frontend and backend. We implemented the modules to set up backend communication with the frontend.

5.2 Server Implementation

5.2.1 Build Microservices

We have created a well-modularized project structure for backend Spring Boot Java keeping in mind different entities needed for the entire backend. We have segregated all of the components of the backend like source, database repository, integration component.

We have microservices for each of the software applications being used like Jira, Jenkins, GitHub, Splunk. Each of the microservice has REST API's which fetch the data from its respective software application according to the use case and stores that data in the backend databases according to the use case.

We have built a cluster of microservices where each of the microservice handles its respective software and has REST API calls to that software. We have containerized each of these microservices and hosted them on the cloud.

These are some of the features we implemented during the development of our microservices:

1. Testing APIs and access tokens for various tools.

2. Setting up connection API's with tools like GitHub, Jira, Jenkins, Slack

3. Develop routes and HTTP method requests in the backend and test it using postman.

4. Get the authorization token/access key from each of the tools.

5. Host Jenkins and JIRA server on AWS and link to GitHub.

6. Build backend services for each of the tools and their functionalities as a microservices architecture.

7. Dockerize each of the microservices.

8. Maintain docker images on the Docker Hub for hosting.

9. Integrate all the microservices in the backend to give an end to end flow for backend services.

5.2.2 Setup Database Connection

We have used Spring Boot to develop the backend services and hence the database connection is done using Spring Boot. It provides great support to create a data source for databases. We have used the Spring JPA driver connection to connect our application backend to the database. We have created a schema for the MySQL database and configured our application to connect to the database which would push the data to the database from our application.

5.2.3 Connect RESTful APIs

As previously described, we have implemented REST APIs in Java Spring MVC which would fetch the required data according to the use case from each of the software applications like Jira, Jenkins, GitHub, and Splunk. The data would be dynamic. We display the data on the frontend of our application using these REST APIs.

5.3 Data-Tier Implementation

1. We have designed an ER diagram with a detailed layout of all the entities that are being used.

2. According to the use case of the system, we decided on the database to be used.

3. Also for the database to scale well according to the requirement, we used a NoSQL database.

4. For certain components that are read-heavy, we have stored the data in a database with master-slave architecture.

5. We made use of a cluster of MongoDB databases to read heavy components.

6. We are using highly available databases for components which are write-heavy. The write-heavy systems need to be always available to give users a seamless experience with minimum downtime.

Chapter 6. Testing and Verification

6.1 Testing

The main focus of this chapter is to ensure that the quality of the application is retained. The bottom-up approach is used for the testing of the application, which starts with the granular level of testing the routes of each of the individual mediator services. The testing phase includes unit testing, Integration testing, and black-box testing.

6.1.1 Unit Testing - Third-Party Interface testing

The mediator routes are tested in this phase. This is the bottom-most and the most granular testing phase to ensure that all the routes which in turn are the functionalities of the third-party tools are working.

Table 2. UNIT TESTING

Tool	APIs Testing description
Jenkins	1. PAM Authentication The Jenkins service requires authentication of users for security. The authentication is done through the API. The test ensures that the authentication route for Jenkins mediator service connects to the Jenkins server and validates the user. 2. API testing (Postman/Junit)

	This test is performed to validate all the routers of the Jenkins mediator service. 3. Builds Ensure builds are initiated through the API routes. 4. Get Build Logs The API also has a route for getting the Jenkins built logs, which is validated in this case.
Splunk	Splunk API + Java SDK: 1. Authentication The Splunk service can only be accessed by authorized users. The authentication can be done through one of the routes of the mediator service. The API validates the proper function of this functionality. 2. Search queries The Splunk server generates the logs which are used to analyze the Jenkins builds. The tests ensure the logs are returned in the desired format (JSON). 3. Save reports There is a route available to save the reports of the log file in various file formats which is tested in this phase.

JIRA	1. Authentication The Jira service requires authentication of users for security. The authentication is done through the API. The test ensures that the authentication route for the Jira mediator service connects to the Jira server and validates the user. 2. Get Sprints Gets details of the sprints of a project that a software developer wants to look at. 3. Get Stories Gets all the stories in a particular sprint that the user has selected. 4. Get Status Gets the status of each of the stories that the user wants to have a look at. 5. Get issues/Epic Gets the issues under a selected sprint.
GitHub	1. Connection to GitHub using tokens The authentication of the connected user. 2. Get number of Commits

	Get the number of commits to a project repository to keep the user updated on the commits that are happening and also the information of the commit to knowing what exactly the commit is about. 3. Get number of Downloads 4. Get number of Issues Get the total number of issues and its description to have an idea of what are the ongoing issues in a project.

To test each of the components developed at a modular level, we have used Junit for testing. Junit is a unit testing framework for Java. Also, because we have a microservices architecture, unit testing the microservices was much needed. The backbone of the project would be the dashboard backend class which would be accessing all the APIs like authentication and the 4 orchestrator services to access the third-party APIs.

6.1.2 Integration Testing

The application consists of many functions that have a dependency on each other. To ensure smooth functioning, there are mediator services that can communicate with the main dashboard service. It is important to ensure the compatibility between these services. The integration and compatibility between the services are tested in this phase.

System Overview

The integration testing is done between APIs which act as intermediary services. The dashboard service communicates and displays the analytics part of the product. The APIs are tested using Postman. For Integration Testing, the bottom-up approach is chosen.

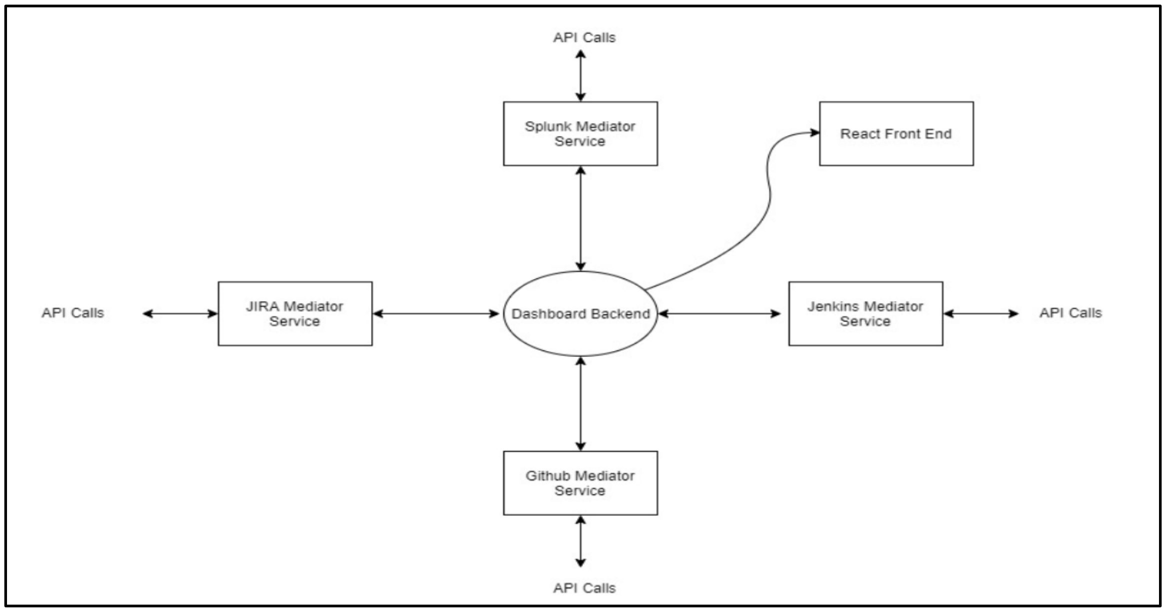


Fig 22. Integration testing

Each individual module is tested first, in the unit testing. The integration of the SDK/APIs with the mediators are then tested. This approach is considered as the entire system is based on the functioning of individual third-party modules. The diagram shows the dependency between modules and the approach taken for testing them.

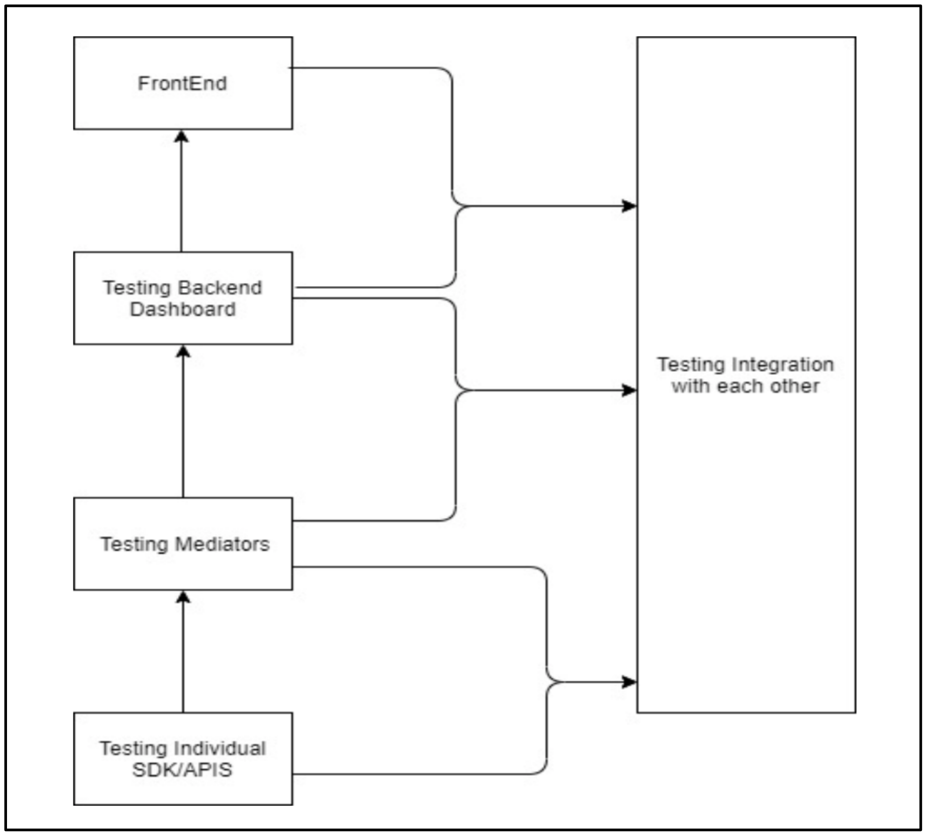

FrontEnd
Testing Backend Dashboard
Testing Mediators
Testing Individual SDK/APIS
Testing Integration with each other

Fig 23. Test procedure flow chart

6.1.3 System Testing

The internal functioning of the system is ignored in System testing. The focus completely relies on what kind of inputs the user provides and what is the output of the system. This is the final step of the testing procedure wherein the interactivity of the application is tested. This type of testing enables the developers to understand the gap between the expected action and the actual action of the system given a certain type of input.

Input
Black Box
Output

Fig 24. Black box testing

Chapter 7. Performance

7.1 Scalability

As the load on the application increases, it becomes increasingly important to maintain the performance of the application when the workload increases. The workload can be characterized by storage capacity, concurrent users accessing the website et cetera. The software design should be such that in the future, there are minimal modifications needed to be done to maintain the efficacy of the application. Our project focuses on scalability on client-side and using dockers.

7.1.1 Client-side scalability

When we build a frontend with React, we use components and these components are split as per the requirements to achieve better structure for the application. We do this splitting efficiently as this makes the components reusable and we can edit the components without messing up the whole structure. It involves Higher-Order Components that are extracted reusable components. These Higher-Order Components can be applied to other components rather than making big components with a large number of dependencies. Also, react.js uses Virtual DOM and only reflects the changed components on the original DOM that makes development faster.

7.1.2 Docker Scalability

Docker is mainly used to create, run, and deploy your application using containers. When you have a microservices architecture you need to package each of the microservices into a container and spin the containers which would effectively help your application to be deployed seamlessly on the cloud. Containers basically allow a developer to package up an application with all the necessary components in it like external libraries and other packages and deploy it as a single package so that we do not have to care about anything else but just run the docker container.

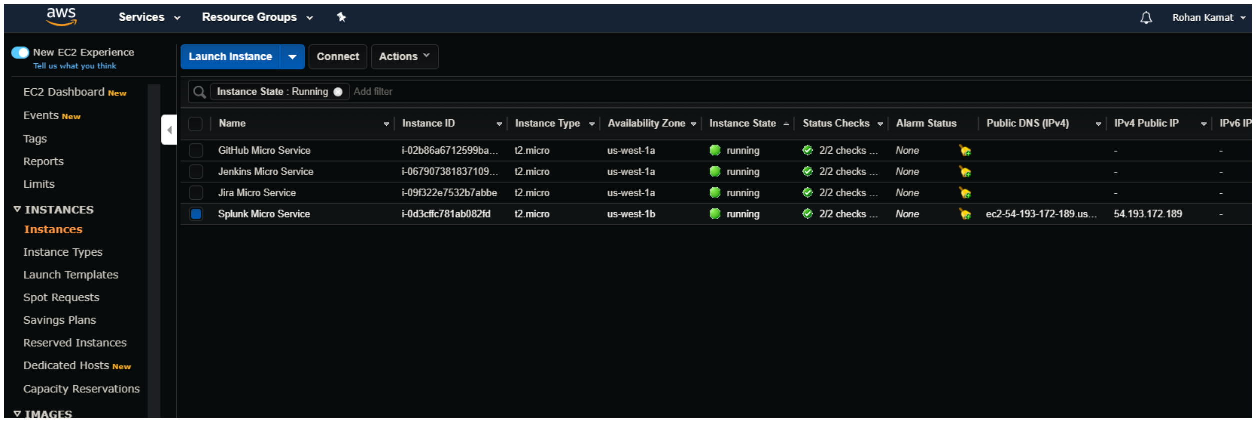
We can always use docker to also scale our application when there is an increased load on the server. We can spin up multiple instances of a service and distribute the load onto multiple services instead of them going onto a single server.

Chapter 8. Deployment and Maintenance

8.1. Amazon Web Services Deployment

Amazon Web Services is the main platform for the deployment of our services. The backend is divided into multiple services that are deployed separately for independent isolation.

Microservices hosted on AWS EC2

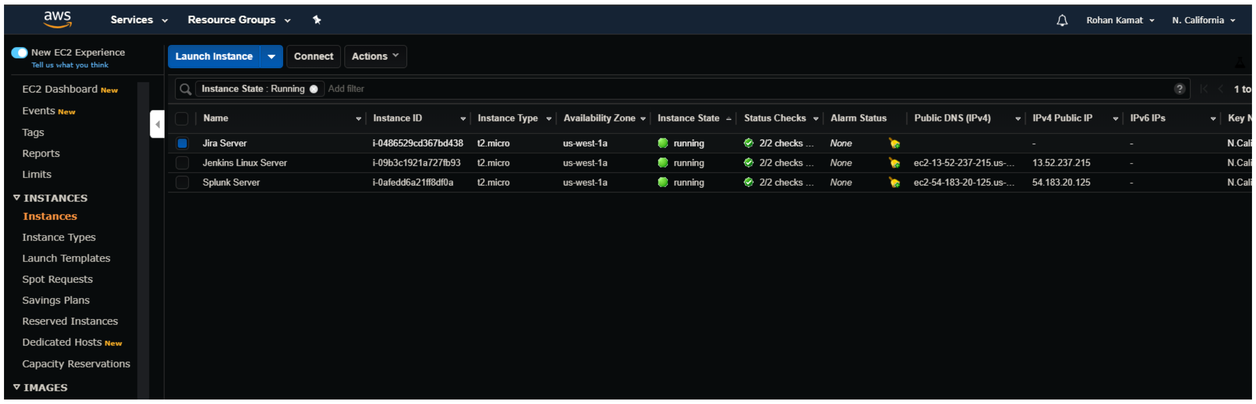


Instance State : Running Add filter

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IP
GitHub Micro Service	i-02b86a6712599ba...	t2.micro	us-west-1a	running	2/2 checks ...	None		-	-
Jenkins Micro Service	i-067907381837109...	t2.micro	us-west-1a	running	2/2 checks ...	None		-	-
Jira Micro Service	i-09f322e7532b7abbe	t2.micro	us-west-1a	running	2/2 checks ...	None		-	-
Splunk Micro Service	i-0d3cffc781ab082fd	t2.micro	us-west-1b	running	2/2 checks ...	None	ec2-54-193-172-189.us...	54.193.172.189	-

Fig 25. AWS console for all deployed microservices

Server for Jenkins, Jira, and Splunk hosted on EC2 Linux machines



Instance State : Running Add filter 1 to

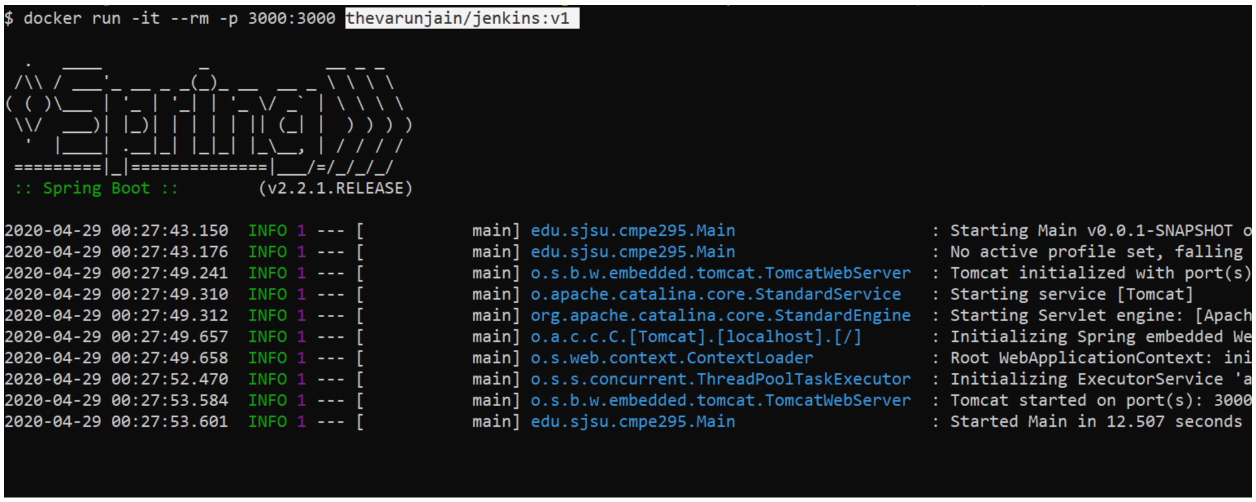
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key N
Jira Server	i-0486529cd367bd438	t2.micro	us-west-1a	running	2/2 checks ...	None		-	-	N.Cali
Jenkins Linux Server	i-09b3c1921a727fb93	t2.micro	us-west-1a	running	2/2 checks ...	None	ec2-13-52-237-215.us-...	13.52.237.215	-	N.Cali
Splunk Server	i-0afedd6a21ff8df0a	t2.micro	us-west-1a	running	2/2 checks ...	None	ec2-54-183-20-125.us-...	54.183.20.125	-	N.Cali

Fig 26. AWS console for all deployed Linux servers

8.2 Docker Containers

Docker is a containerization technology which is a popular tool for microservice deployments. All our third-party mediator microservices are deployed using docker containers.

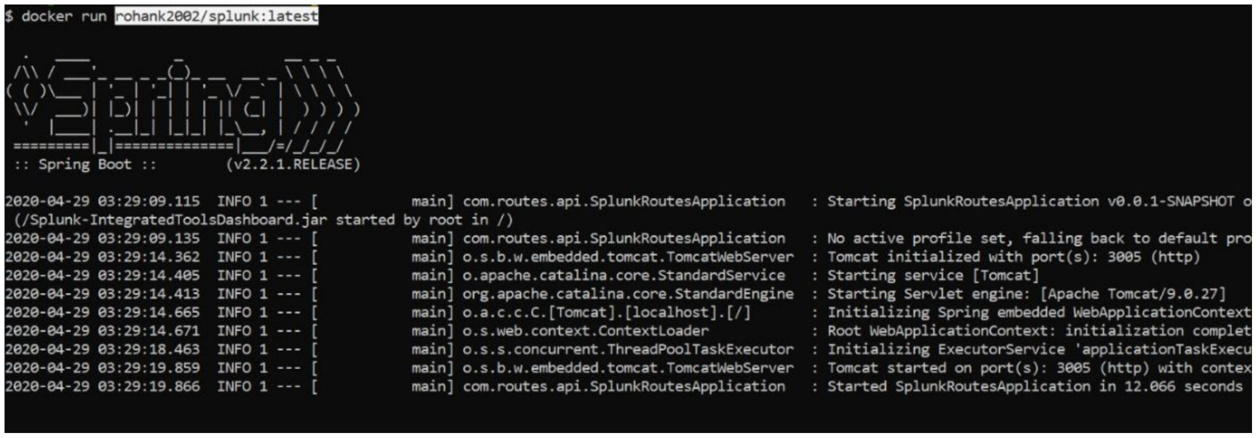
Jenkins microservices running in a docker container



```
$ docker run -it --rm -p 3000:3000 thevarunjain/jenkins:v1

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.1.RELEASE)

2020-04-29 00:27:43.150  INFO 1 --- [           main] edu.sjsu.cmpe295.Main                    : Starting Main v0.0.1-SNAPSHOT o
2020-04-29 00:27:43.176  INFO 1 --- [           main] edu.sjsu.cmpe295.Main                    : No active profile set, falling
2020-04-29 00:27:49.241  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s)
2020-04-29 00:27:49.310  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-04-29 00:27:49.312  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apach
2020-04-29 00:27:49.657  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded We
2020-04-29 00:27:49.658  INFO 1 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: ini
2020-04-29 00:27:52.470  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'a
2020-04-29 00:27:53.584  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 3000
2020-04-29 00:27:53.601  INFO 1 --- [           main] edu.sjsu.cmpe295.Main                    : Started Main in 12.507 seconds
```

Fig 27. Jenkins docker container

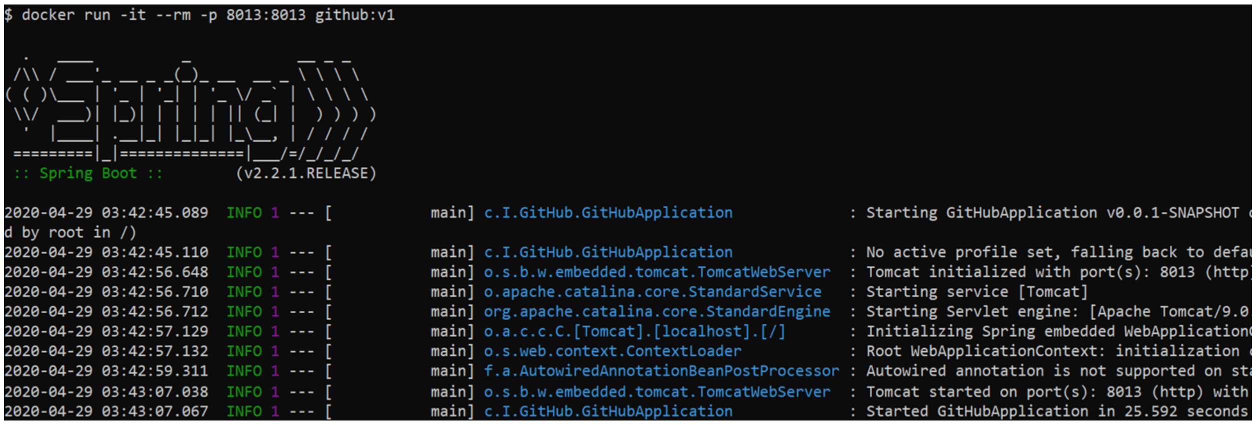
Splunk microservice running in a docker container



```
$ docker run rohank2002/splunk:latest

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.1.RELEASE)

2020-04-29 03:29:09.115  INFO 1 --- [           main] com.routes.api.SplunkRoutesApplication   : Starting SplunkRoutesApplication v0.0.1-SNAPSHOT o
 (/Splunk-IntegratedToolsDashboard.jar started by root in /)
2020-04-29 03:29:09.135  INFO 1 --- [           main] com.routes.api.SplunkRoutesApplication   : No active profile set, falling back to default pro
2020-04-29 03:29:14.362  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 3005 (http)
2020-04-29 03:29:14.405  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-04-29 03:29:14.413  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.27]
2020-04-29 03:29:14.665  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2020-04-29 03:29:14.671  INFO 1 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization complet
2020-04-29 03:29:18.463  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecu
2020-04-29 03:29:19.859  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 3005 (http) with contex
2020-04-29 03:29:19.866  INFO 1 --- [           main] com.routes.api.SplunkRoutesApplication   : Started SplunkRoutesApplication in 12.066 seconds
```

Fig 28. Splunk docker container

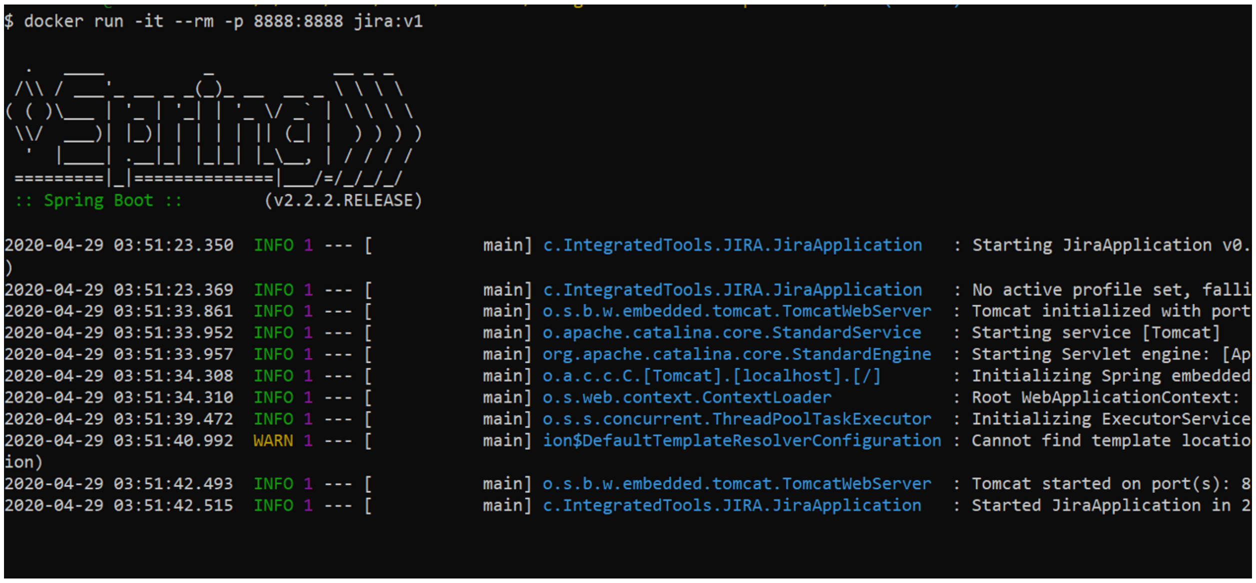
GitHub microservices running in a docker container



```
$ docker run -it --rm -p 8013:8013 github:v1

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.1.RELEASE)

2020-04-29 03:42:45.089  INFO 1 --- [           main] c.I.GitHub.GitHubApplication             : Starting GitHubApplication v0.0.1-SNAPSHOT o
d by root in /)
2020-04-29 03:42:45.110  INFO 1 --- [           main] c.I.GitHub.GitHubApplication             : No active profile set, falling back to defa
2020-04-29 03:42:56.648  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8013 (http
2020-04-29 03:42:56.710  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-04-29 03:42:56.712  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0
2020-04-29 03:42:57.129  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplication
2020-04-29 03:42:57.132  INFO 1 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization 
2020-04-29 03:42:59.311  INFO 1 --- [           main] f.a.AutowiredAnnotationBeanPostProcessor : Autowired annotation is not supported on st
2020-04-29 03:43:07.038  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8013 (http) with
2020-04-29 03:43:07.067  INFO 1 --- [           main] c.I.GitHub.GitHubApplication             : Started GitHubApplication in 25.592 seconds
```

Fig 29. GitHub docker container

Jira microservices running in a docker container

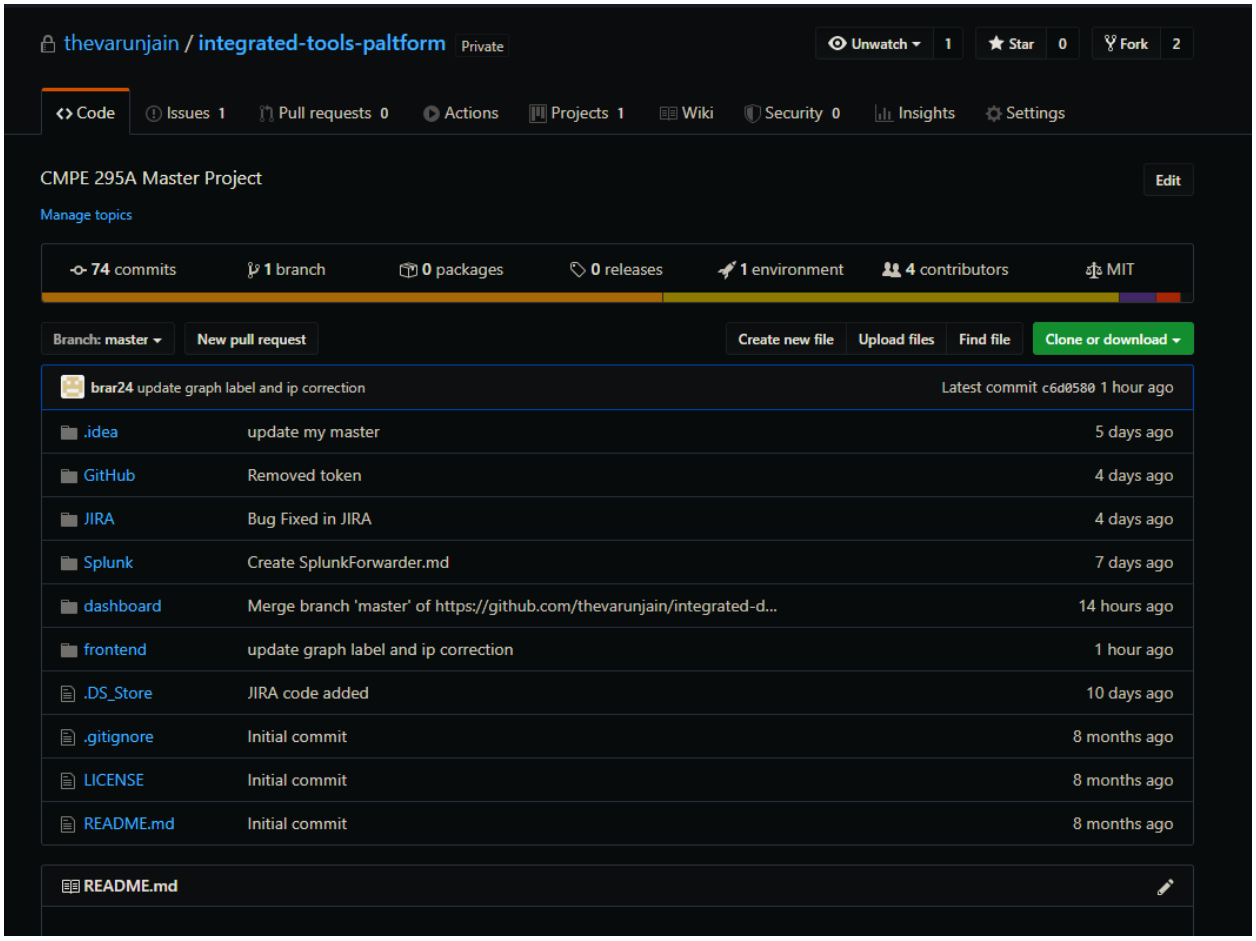


```
$ docker run -it --rm -p 8888:8888 jira:v1

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.2.RELEASE)

2020-04-29 03:51:23.350  INFO 1 --- [           main] c.IntegratedTools.JIRA.JiraApplication   : Starting JiraApplication v0.
)
2020-04-29 03:51:23.369  INFO 1 --- [           main] c.IntegratedTools.JIRA.JiraApplication   : No active profile set, falli
2020-04-29 03:51:33.861  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port
2020-04-29 03:51:33.952  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-04-29 03:51:33.957  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Ap
2020-04-29 03:51:34.308  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded
2020-04-29 03:51:34.310  INFO 1 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: 
2020-04-29 03:51:39.472  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService
2020-04-29 03:51:40.992  WARN 1 --- [           main] ion$DefaultTemplateResolverConfiguration : Cannot find template locatio
ion)
2020-04-29 03:51:42.493  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8
2020-04-29 03:51:42.515  INFO 1 --- [           main] c.IntegratedTools.JIRA.JiraApplication   : Started JiraApplication in 2
```

Fig 30. Jira docker container

8.3 Version Code Management

We have used the GitHub repository for collaboratively contributing the code and maintaining the version. We have also used the GitHub project board to assign tasks and take a follow up in our meetings.



thevarunjain / integrated-tools-paltform Private

Unwatch 1 Star 0 Fork 2

Code Issues 1 Pull requests 0 Actions Projects 1 Wiki Security 0 Insights Settings

CMPE 295A Master Project

Manage topics

Edit

74 commits 1 branch 0 packages 0 releases 1 environment 4 contributors MIT

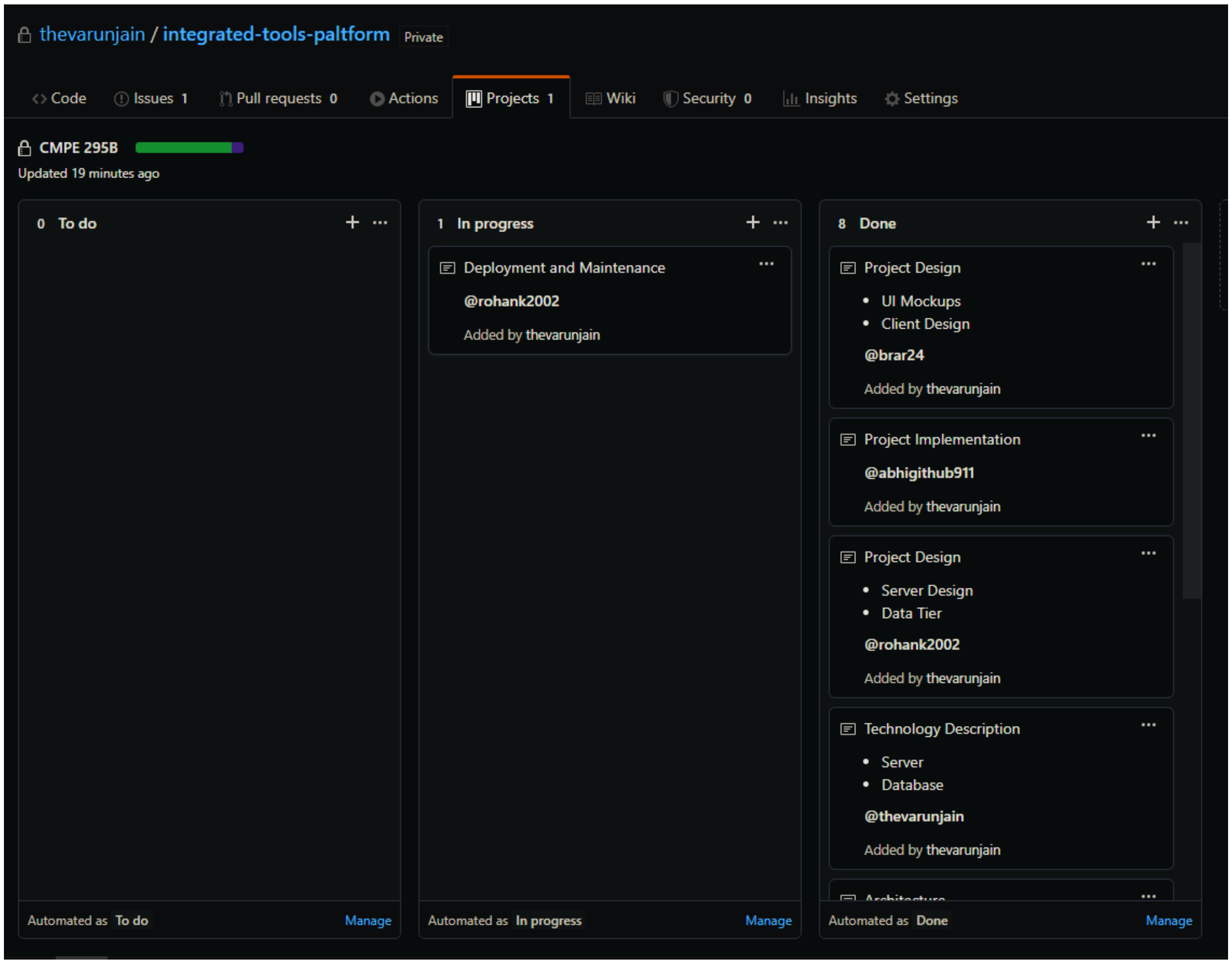
Branch: master New pull request Create new file Upload files Find file Clone or download

brar24 update graph label and ip correction Latest commit c6d0580 1 hour ago

.idea	update my master	5 days ago
GitHub	Removed token	4 days ago
JIRA	Bug Fixed in JIRA	4 days ago
Splunk	Create SplunkForwarder.md	7 days ago
dashboard	Merge branch 'master' of https://github.com/thevarunjain/integrated-d...	14 hours ago
frontend	update graph label and ip correction	1 hour ago
.DS_Store	JIRA code added	10 days ago
.gitignore	Initial commit	8 months ago
LICENSE	Initial commit	8 months ago
README.md	Initial commit	8 months ago

README.md

Fig 31. GitHub Code Repository



thevarunjain / integrated-tools-paltform Private

CMPE 295B

Updated 19 minutes ago

0 To do

Automated as To do Manage

1 In progress

Deployment and Maintenance

@rohank2002

Added by thevarunjain

Automated as In progress Manage

8 Done

Project Design

- UI Mockups
- Client Design

@brar24

Added by thevarunjain

Project Implementation

@abhigithub911

Added by thevarunjain

Project Design

- Server Design
- Data Tier

@rohank2002

Added by thevarunjain

Technology Description

- Server
- Database

@thevarunjain

Added by thevarunjain

Automated as Done Manage

Fig 32. Project task board

Chapter 9. Conclusions and Future Work

9.1 Conclusion

Integrated Tools Dashboard effectively solves the challenges of integrating various software applications used in a company. The project also saves most programmers' precious time and make it simple and accessible to function. We have created a revolutionary idea to build a generic graphical interface and user experience for these devices and software applications onto a single platform called Integrated Dashboard.

We have developed a centralized system to interact with all of these software technologies and access all of the basic and essential information to get an overall idea of the current flow of the project. We have also developed some features which help the customer perform some basic operations like assigning a task to a teammate in Jira.

A software developer would save a lot of time using the Integrated Dashboard as he/she won't have to anymore go to each of the applications to get his/her task done, instead, he/she just has to login to the Integrated Dashboard application and get an overall view of his/her work. This also gives an advantage of having the ability to look at all of the tasks related to a project at one place which can help in better analytics and understanding of the current flow and help the team plan for future tasks at ease.

9.2 Future Work

For future work, we can plan it to be enhanced by software delivery in an optimized security company environment. The application can be improved to include more channels such as Google Assistant, Slack, Cortana, and Siri on their mobile devices which can help the user to work from anywhere and post the commands on the go without having a workstation or a home station or just a laptop dependency.

We can expand the scalability using container orchestration technology like Kubernetes. Kubernetes is a container management tool that enhances the scalability aspects of any containerized architecture. The docker containers can be scaled up/down using AWS auto-scaling groups and Kubernetes. Kubernetes also allows automatic deployments during version up-gradation. It has an automated rollout procedure that ensures minimalistic downtime during maintenance.

References

[1] Wray, Stacy. “Integrating JIRA, GitHub, Jenkins, and Slack in your workflow” *Acquia,* Acquia Inc., 27 September 2019 https://support.acquia.com/hc/en-us/articles/360005167214-Integrating-JIRA-GitHub-Jenkins-and-Slack-in-your-workflow

[2] Zadrozny P., Kodali R. (2013) *Getting Data into Splunk*. In: Big Data Analytics Using Splunk. Apress, Berkeley, CA, https://doi-org.libaccess.sjlibrary.org/10.1007/978-1-4302-5762-2_2

[3] Kulshrestha, Saurabh. “What is Jenkins? | Jenkins For Continuous Integration | Edureka” *edureka*, Brain4ce Education Solutions Pvt. Ltd, 22 May 2019, https://www.edureka.co/blog/what-is-jenkins/

[4] Blischak JD, Davenport ER, Wilson G (2016) A Quick Introduction to Version Control with Git and GitHub. PLoS Comput Biol 12(1): e1004668. https://doi.org/10.1371/journal.pcbi.1004668

[5] Dreyfuss, Josh. “7 JIRA Integrations to Optimise Your Java Development Workflow” OverOps, OverOps Inc. 2019, 28 January 2015, https://blog.overops.com/7-jira-integrations-to-optimize-your-java-development-workflow/

[6] Husejko, Himyr, Gonzalez, Koloventzos, Asbury, Trzcinska, Agtzidis, Botrel, and Otto. "Self-service for Software Development Projects and HPC Activities." *Journal of Physics: Conference Series* 513.5 (2014): 8. Web.

[7] Perscheid, Michael, Benjamin Siegmund, Marcel Taeumel, and Robert Hirschfeld. "Studying the Advancement in Debugging Practice of Professional Software Developers." *Software Quality Journal* 25.1 (2017): 83-110. Web.

[8] IBM, Inc. (2000, October 5). *WiredAnwhere.* Retrieved from http://www.alphaworks.ibm.com/tech/wiredanywhere.