International Institute of Information Technology, Bangalore

Software Production Engineering Mini Project

Webboard 2.0

In the Guidance of : Prof. B. Thangaraju

Guide: Ansh Goyal

Satvik Ramaprasad
IMT2016008

Ananth Shreekumar
IMT2016129

Archit Kashyap
IMT2016064

Sudharsanaraj
IMT2016130

1 Table of Contents

2 Abstract

Interactive Smart boards are rapidly replacing traditional chalkboards and white boards. From the pedagogical perspective, smart boards are superior because it enables the students to experience things in a more visual and interactive manner. Smartboards reduce the effort for the teacher as the amount of time spent in class to write content as the content can be prepared beforehand. Moreover, the content can be easily revisited in the future for reference. As technology improves, the costs are going down as well, leading to rapid adoption.

3 Introduction

Webboard is a novel idea which takes smartboards one step forward by moving to the cloud. While smart boards have been around for a long time now, teachers have difficulty in managing the documents on them. Students still write down notes the traditional way. Teachers don't have access to last year's notes as they have to manage the documents themselves. If a teacher starts a lecture in one classroom, he or she needs to manually copy the documents if they need to relocate to another class.

Webboard solves these problems in a unique manner. Webboard saves everything on the cloud. Students can easily access this content anytime by logging in. Teachers can revisit or update the content by simply going to their course page and viewing currently recorded lectures.

Webboard is one of its kind. We have a full-fledged CMS where the administrator can manage students, lecturers, courses etc.

The various advantages of using our system are as follows:

- Reduced effort for instructors.
- The board can be projected at multiple places making it easier for students in big classrooms.
- Lectures can be exported in pdf format which can be shared easily with the class.
- Easier to draw certain shapes.
- Supports standard CAD features like the ability to undo, redo, cut, copy and paste.
- Our system autosaves changes to the cloud. In case the internet is absent, it autosaves locally and syncs to the cloud once a connection has been established.

- Our system has a unique playback feature which enables the students to view the lecture exactly as it happened, i.e. they can play the lecture like a video, skip ahead or rewind to see how the lecture progressed.

3.1 Why DevOps?

- Easy collaboration between team members, across development and operations.
- Enable building production software which is free of "bugs discovered very late in the pipeline".
- Easy integration with different applications that aid in testing and deployment.
- Faster release of features into market.
- Better cooperation between different teams

3.2 Work done as part of the Project

Webboard was an ongoing project managed by one of the teammates (Satvik Ramaprasad). Another teammate (Ananth Shreekumar) has been an extensive user of Webboard as a Teaching Assistant for courses. This is a unique instance where the user is also the developer! The beta deployment can be seen **here**. In this project, we incorporated several DevOps tools and added several highly requested features. Some features added are:

1. Undo-Redo functionality
2. Lecture Playback
3. Auto Save
4. Selection
5. Cut
6. Copy
7. Delete
8. Paste
9. Integrated Admin Dashboard

4 System Configuration

Ruby on Rails was used to write the application.

We used 4 Azure Vms in the cloud:

1. Admin node -
    - Ubuntu 18.04 LTS Operating System.
    - 30 GB SSD, 2GB RAM, 1 vCPU
    - Kernel version - 5.3.0-1020-azure
    - Jenkins and Rundeck run on this node.
2. Deployment node 1 -
    - Ubuntu 18.04 LTS Operating System.
    - 30 GB SSD, 2GB RAM, 1 vCPU
    - Kernel version - 5.3.0-1020-azure
    - Application deployed here.
    - Logstash sends logs to the monitoring node.
    - Chef used to configure the environment.
3. Deployment node 2 -
    - Ubuntu 18.04 LTS Operating System.
    - 30 GB SSD, 2GB RAM, 1 vCPU
    - Kernel version - 5.3.0-1020-azure
    - Application deployed here.
    - Logstash sends logs to the monitoring node.
    - Chef used to configure the environment.
4. Monitoring node -
    - Ubuntu 18.04 LTS Operating System.
    - 30 GB SSD, 8GB RAM, 2 vCPU
    - Kernel version - 5.3.0-1020-azure
    - ElasticSearch receives logs from both deployment nodes and prepares indexes here. Kibana also runs here providing the visualizations.

All 4 VMs had SSH only access for security. The 4 VMs were also added to an azure private network, to allow secure communication between the VMs. All efforts were taken to ensure maximum security possible.

5 Software Development Life Cycle

5.1 Installation Procedure

5.1.1 Docker

To install docker [1] follow this installation guide and choose the setup for your operating system accordingly,

1. Add GPG key for the official docker repository to your system.

   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

2. Add the docker repository to APT sources.

   sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

3. Update the package database.

   sudo apt-get update

4. Install docker.

   sudo apt-get install -y docker-ce

5. Add user to docker group

   sudo usermod -aG docker <username>

5.1.2 Jenkins

We installed Jenkins [3] following the instructions on the official website. We had to install Java as Jenkins uses Java. We had to copy the password and install jenkins as shown below

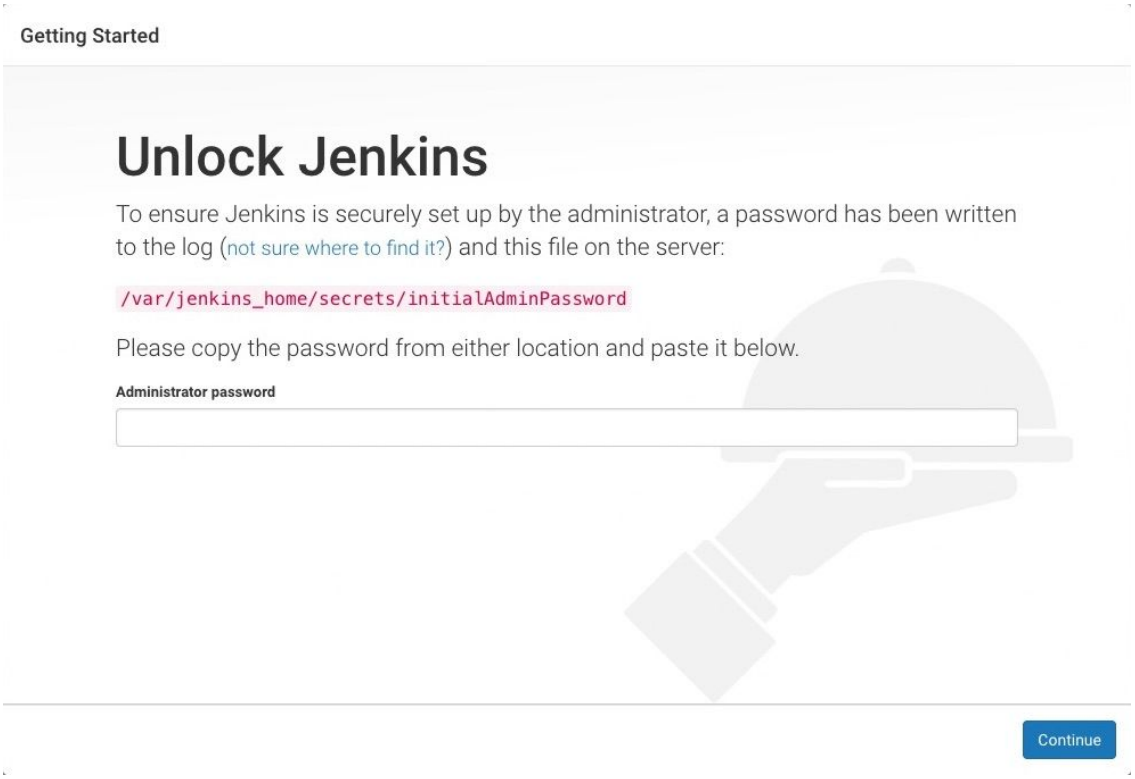

Fig 1: Jenkins asking for administrator password for the initial login

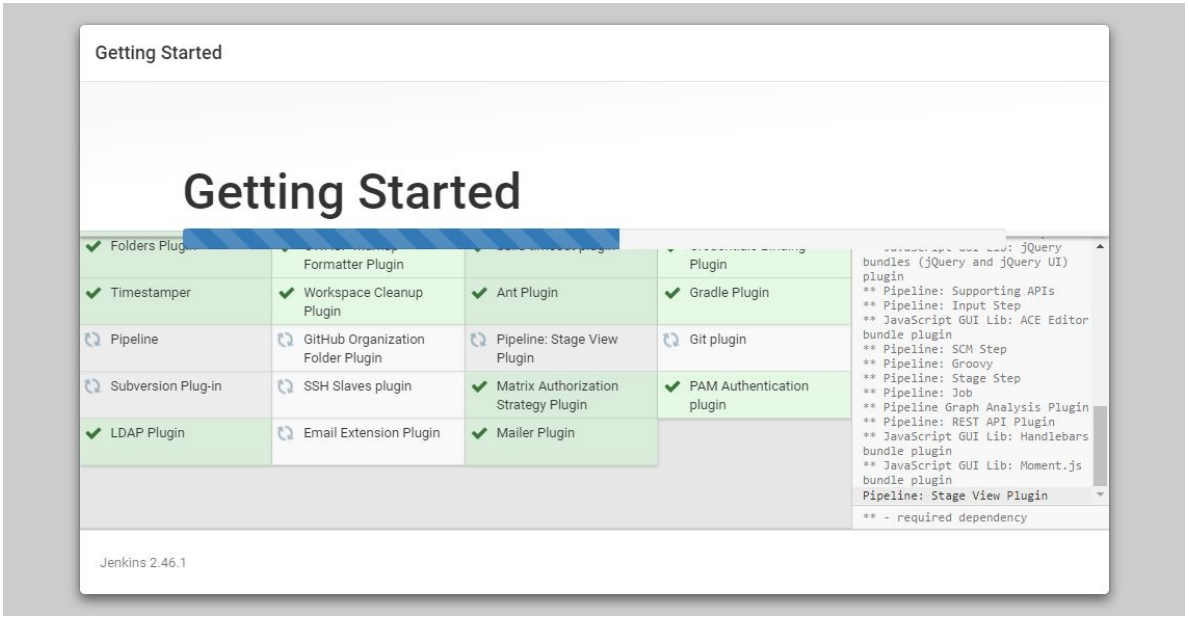

Fig 2: Jenkins installing recommended plugins

Visit our deployed Jenkins @ jenkins.webboard-spe.tk.

5.1.3 Rundeck

To install rundeck [6],

1. Download the latest rundeck debian package from **Rundeck Debian packages.**
2. Install the package using the following command:

sudo dpkg -i {rundeck package name}

After starting the service, access the server at localhost:4440.

You will see a login page, Login as:

- Username: admin
- Password: admin

We had to set the java version in the rundeck config file.

As we deployed rundeck in a public VM, we changed the default password.

The 4 VMs were added as nodes to the rundeck instance by editing the *resources.xml* file as below. Note that the 4 nodes are authenticated via SSH keys only. We had to add rundeck's public ssh key into the authorized key list located at *~/.ssh/authorized_keys*.

```
<?xml version="1.0" encoding="UTF-8"?>

<project>
  <node name="WebboardNode1"
        description="Webboard server node" tags=""
        hostname="52.234.213.149"
        osArch="amd64" osFamily="unix" osName="Linux" osVersion="5.3.0-1020-azure"
        username="satvik"
        ssh-keypath="/var/lib/rundeck/id_rsa"/>
  <node name="WebboardNode2"
        description="Webboard server node" tags=""
        hostname="13.72.87.194"
        osArch="amd64" osFamily="unix" osName="Linux" osVersion="5.3.0-1020-azure"
        username="Sudharshan"
        ssh-keypath="/var/lib/rundeck/id_rsa"/>
<node name="WebboardNode_Admin"
        description="Webboard server node" tags=""
        hostname="52.170.254.153"
        osArch="amd64" osFamily="unix" osName="Linux" osVersion="5.3.0-1020-azure"
        username="ananth"
        ssh-keypath="/var/lib/rundeck/id_rsa"/>
  <node name="WebboardNode_ELK"
        description="Webboard server node" tags=""
        hostname="23.96.88.64"
        osArch="amd64" osFamily="unix" osName="Linux" osVersion="5.3.0-1020-azure"
        username="archit"
        ssh-keypath="/var/lib/rundeck/id_rsa"/>
</project>
```

Fig 3: Adding nodes to Rundeck - resources.xml

Visit our deployed Rundeck @ rundeck.webboard-spe.tk.

5.1.4 Chef

We used a hosted Chef [7] server and personal computers as Chef workstations. To install a Chef workstation on your PC:

1. Go to **Chef Downloads** and download the appropriate version for your system.
2. Create chef directory and execute the downloaded chef installation file.
3. Download the starter kit for your organization from the hosted chef server.
4. Go to chef directory and unzip the starter kit package, it will create chef-repo directory. The chef-repo directory can be used by knife to push and pull cookbooks.

The chef-client ran on all 4 Azure [5] VMs. To register a node to the chef server, the following command can be used:

knife bootstrap <username>@<ip-address> --ssh-identity-file ~/.ssh/id_rsa -N <chef_node_name> --sudo

Here, we allow only ssh-based authentications for security reasons, hence we had to add the workstation's public ssh key to the authorized list of keys in the 4 VMs. This process was automated by rundeck [6]. The details of the job can be seen later in the rundeck session. Also, we need to allow superuser privilege, hence we require the --sudo flag.

Once all 4 nodes were registered, the chef server shows them as given below:

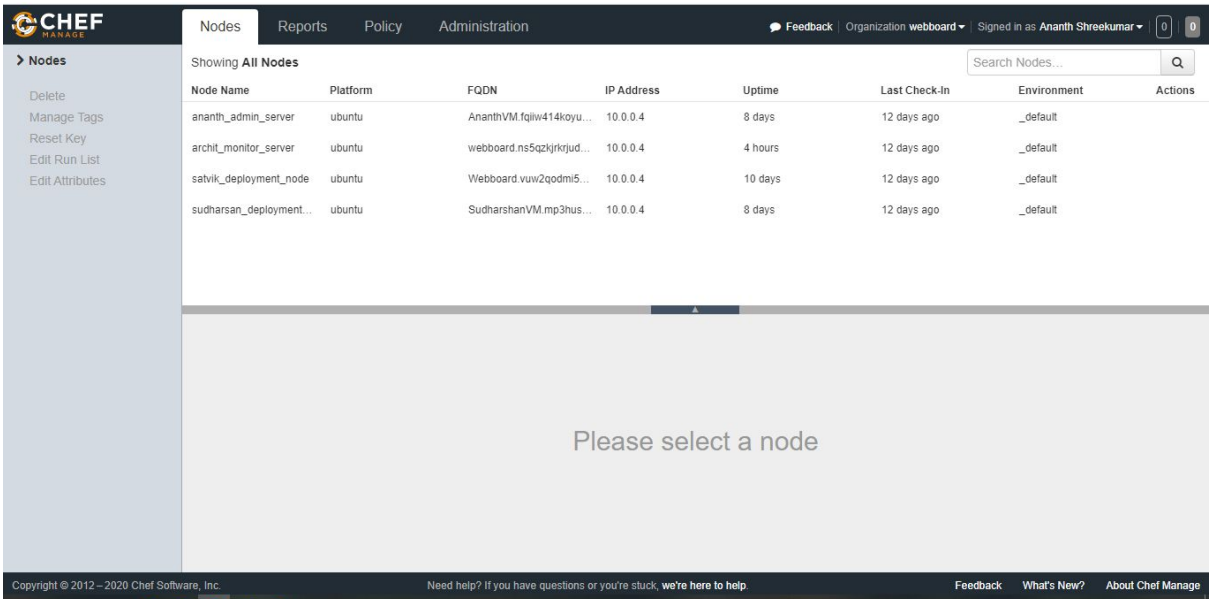

Fig 4: Chef server showing registered nodes

5.1.5 ELK stack

5.1.5.1 Logstash

Logstash [9] is supposed to run on the deployment nodes, while ElasticSearch [8] and Kibana [10] are supposed to run on the monitoring node. Thus, Logstash was installed natively on both the deployment nodes as given below:

1. Download and install the Public Signing Key:

   wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -

2. Save the repository definition by:

   echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main"

   | sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list

3. Install Logstash by:

   sudo apt-get update && sudo apt-get install logstash

5.1.5.2 ElasticSearch

First we need to create a docker network so that elasticsearch and kibana can communicate properly.

docker network create elasticsearch-kibana

We pull the image as follows

docker pull docker.elastic.co/elasticsearch/elasticsearch:7.7.0

ElasticSearch [8] was installed by the usage of a docker container. We used this approach since it was sufficient for our requirements. We mounted the elastic data on to a volume so that we can persist the data safely.

```
1  docker run --name elasticsearch
2      -p 9200:9200 -p 9300:9300
3      -e "discovery.type=single-node"
4      -v elasticdata:/usr/share/elasticsearch/data
5      --network elasticsearch-kibana
6      docker.elastic.co/elasticsearch/elasticsearch:7.7.0
```

Fig 5: *Running ElasticSearch instance in a container*

5.1.5.3 Kibana

Kibana [10] was also installed in a docker container.

The image was pulled from dockerhub:

docker pull docker.elastic.co/kibana/kibana:7.7.0

The kibana container was created and connected to the elastic container as follows:

```
docker run
  --name kibana
  --publish 5601:5601
  --network elasticsearch-kibana
  --env "ELASTICSEARCH_URL=http://elasticsearch:9200"
  docker.elastic.co/kibana/kibana:7.7.0
```

Fig 6: *Running and connecting a Kibana container to an ElasticSearch container*

The deployed version of Kibana is accessible @ kibana.webboard-spe.tk

5.1.6 Cloudflare

We used cloudflare to manage our DNS records. There were many reasons to use cloudflare. Some of them were:

1. Easy and Clean interface to update DNS records
2. Cloudflare DNS records update almost instantly as compared to GoDaddy and others
3. Free DDoS protection
4. Free automatic CDN
5. Free and automatic HTTPs support
6. Provides basic security by hiding our server's IP

Type	Name	Content	TTL	Proxy status	
A	jenkins	52.170.254.153	Auto	Proxied	Edit
A	kibana	23.96.88.64	Auto	Proxied	Edit
A	metabase	52.234.213.149	Auto	Proxied	Edit
A	node1	52.234.213.149	Auto	Proxied	Edit
A	node2	13.72.87.194	Auto	Proxied	Edit
A	rundeck	52.170.254.153	Auto	DNS only	Edit
A	test	52.170.254.153	Auto	Proxied	Edit

Fig 7: DNS records

5.1.7 Reverse Proxy

In order to access the different services by their domain instead of their IP address, we set up Nginx [15] to use it's reverse proxy capabilities.

We need to add a config into */etc/nginx/sites-enabled/* as follows:

```
server {
  listen 80;
  server_name kibana.webboard-spe.tk;
  location / {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_set_header X-NginX-Proxy true;
        proxy_pass http://localhost:5601;
        proxy_redirect http://localhost:5601/ https://$server_name/;
        }
}
```

Fig 8: Adding a reverse proxy for nginx

5.2 Source Code Management (Git)

We used Git [4] for version control and GitHub for source code repository. We used github because we wanted to make our project open source. The source code for the application can be found at **satu0king/webboard_publish**.

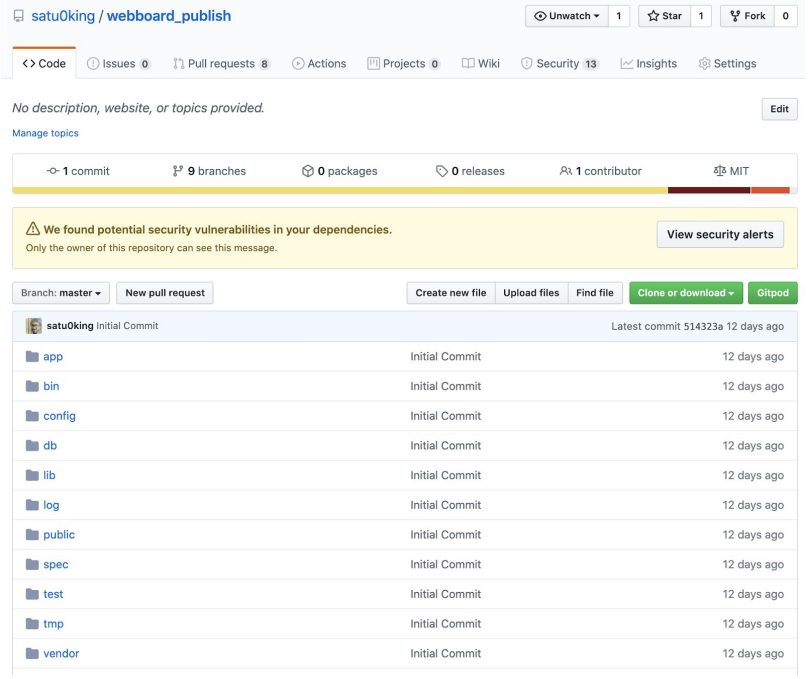

Fig 9: Github repository

5.3 Development Technology and Frameworks (Ruby on Rails)

We used Ruby on Rails [2] as the backend framework. RoR is a fast growing framework which enables developers to bootstrap a backend system very rapidly. For medium scale applications, RoR is an excellent choice. One of the biggest attractions for using ruby on rails is gems. Rails has an excellent developer community which maintains several libraries called "gems". These gems allow application developers to rapidly develop their system.

Fig 10: Ruby on Rails logo

The main drawing app itself has been written in vanilla javascript from scratch. No canvas library or frameworks has been used because we wanted complete control and flexibility for implementing features and fine tuning performance.

5.4 Build (Bundler)

Ruby is a completely interpreted language. It doesn't even have bytecode level compilation like Java. Therefore there is no building as such. However, we do use a package manager called **Bundler** which is analogous to Maven for Java. The specification is written in a file called *Gemfile,* analogous to *pom.xml* for Maven. The exact configuration is generated by bundler and stored in a file called *Gemfile.lock*. Bundler supports the concept of different environments as well. Therefore we can have slightly different configurations for development, testing and production environments.

Fig 11: Bundler log

5.5 Testing Framework (RSpec)

RSpec is the most popular test framework for Ruby applications. RSpec is quite unique in its design as it doesn't test how an application works, instead it tests the behaviour of the application. Another major variation between RSpec and other test frameworks is that RSpec uses expectations instead of assertions.

In order to get the tests published in a format that can be used by the pipeline(Jenkins), we used a gem *rspec_junit_formatter* which formatted the output in the required format.

Fig 12: Rspec logo

Sample test cases shown below:

```
44    describe "GET #new" do
45      it "returns a success response" do
46        get :new, params: {}, session: valid_session
47        expect(response).to be_successful
48      end
49    end
```

Fig 13: Sample Test Case 1

```
7     it "routes to #new" do
8       expect(:get => "/courses/new").to route_to("courses#new")
9     end
```

Fig 14: Sample Test Case 2

5.6 Publish Artifact (Docker Image)

Using Docker [1] container technology is one of the easiest ways to publish your application as it makes it very easy for users to use and deploy your product.

Since our system requires an application container as well as a database container, it was required to set up a *docker-compose* file.

We wrote a basic docker-compose file which quickly lets the user get started. ***Note that this is just a quick getting started configuration If the user wants to setup properly in production, the user will need to do some more work to persist the data, logs etc.***

The docker-compose file is shown below:

```
version: '3.1'
services:
  db:
    image: postgres
    environment:
      POSTGRES_DB: webboard_production
      POSTGRES_PASSWORD: postgres
  web:
    build: .
    entrypoint: /webboard/bin/docker_run
    volumes:
      - .:/webboard
      - ./config/database.docker.yml:/webboard/config/database.yml
    ports:
      - "3000:3000"
    depends_on:
      - db
```

The user can start the containers up by simply running:

docker-compose up

Conversely, the user can stop the containers by simply running:

docker-compose down

```
FROM ruby:2.5.1

# set up workdir
RUN mkdir /webboard
WORKDIR /webboard

# install dependencies
RUN apt-get update -qq && apt-get install -y imagemagick && apt-get clean

RUN curl -sL https://deb.nodesource.com/setup_10.x | bash \
 && apt-get update && apt-get install -y nodejs && rm -rf /var/lib/apt/lists/* \
 && curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add - \
 && echo "deb https://dl.yarnpkg.com/debian/ stable main" | tee /etc/apt/sources.list.d/yarn.list \
 && apt-get update && apt-get install -y yarn && rm -rf /var/lib/apt/lists/*

COPY Gemfile /webboard/Gemfile
COPY Gemfile.lock /webboard/Gemfile.lock
COPY yarn.lock /webboard/yarn.lock

RUN gem install bundler
RUN bundle install
RUN yarn install

# copy source
COPY . /webboard
```

Fig 15: Dockerfile

```
#!/bin/bash

echo "Waiting for db to be available"
for i in $(seq 1 30) ; do timeout 1 bash -c "echo > /dev/tcp/db/5432" \
   > /dev/null 2>&1 && status=0 && break || status=$? && sleep 1 ; done

if [ $status -ne 0 ]
then
    echo "Could not connect to db"
    exit 0
fi

echo "Running db:create"
RAILS_ENV=production bundle exec rails db:create
echo "Running db:migrate"
RAILS_ENV=production bundle exec rails db:migrate

rm -f /circuitverse/tmp/pids/server.pid

echo "Building static assets"
RAILS_ENV=production rails assets:precompile

echo "Starting on 127.0.0.1:3000"
RAILS_ENV=production bundle exec rails s -p 3000 -b '0.0.0.0'
```

Fig 16: docker_run

5.7 Infrastructure as a Code (Chef)

Chef [7] was used to set up the environments required in the deployment nodes.

5.7.1 Server

We used a hosted Chef Server since it was sufficient for our needs. The below figure shows the run list of a deployment node as seen on the hosted Chef server:

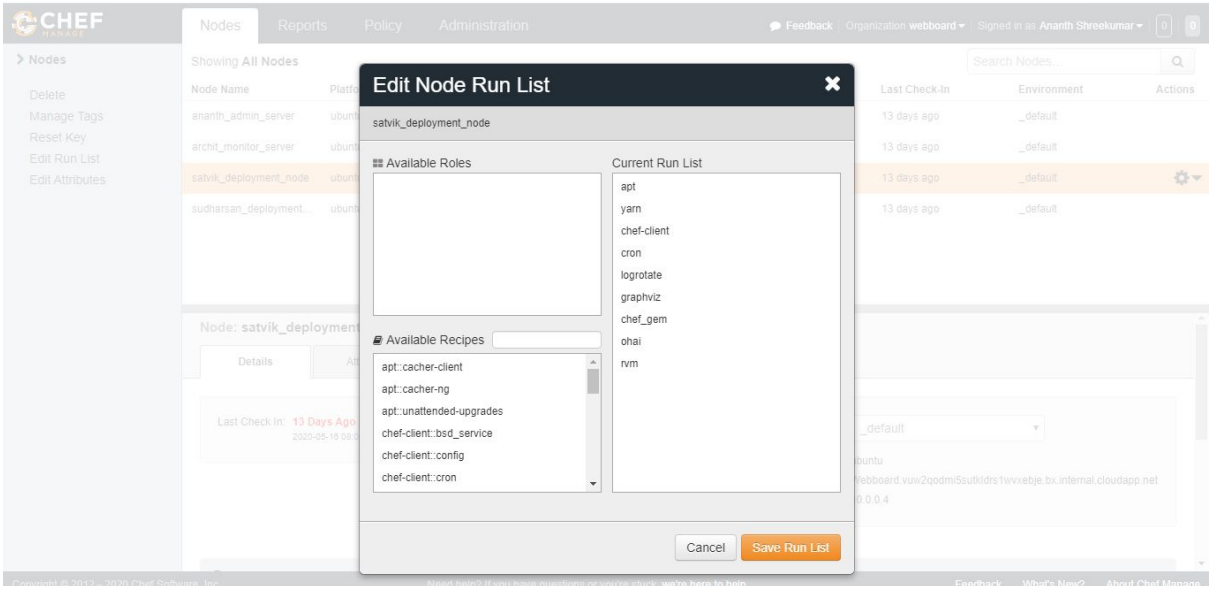

Fig 17: Chef server allows editing run list for a node

5.7.2 Workstation

We used our personal computers for workstations. Cookbooks were download from the Chef supermarket by the command:

knife supermarket download <cookbook>

Downloaded cookbooks were unzipped and moved to the cookbooks directory inside the chef-repo. Cookbooks were then uploaded to the Chef server by the command:

knife cookbook upload <cookbook>

5.7.3 Client

The deployment nodes ran chef-client to pull the cookbooks from the server and run them to set up the required environment.

5.8 Deployment (Rundeck)

Rundeck [6] was used to provide continuous deployment of added features to both the development nodes. A rundeck deploy job was created, and the Rundeck plugin was installed on Jenkins. The image below shows the jobs running successfully in both the deployment nodes.

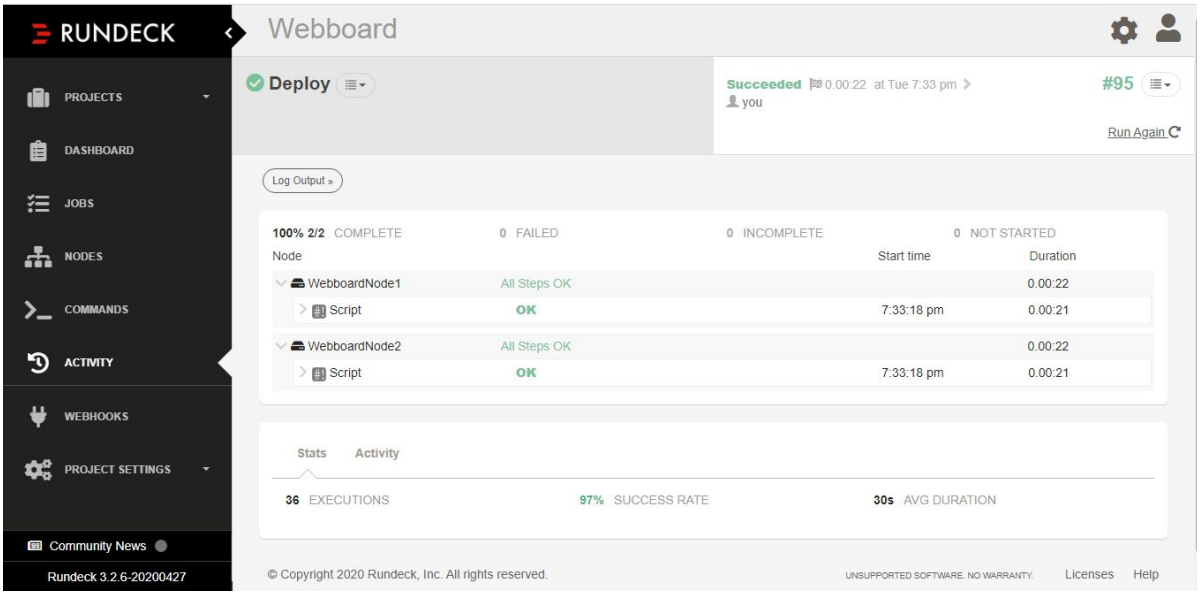

Fig 18: Rundeck showing successful deployment to both nodes

The deploy job was as follows. It updated the production code and built static assets. Then it restarted the server.

```
1    cd ~/Webboard
2    git pull
3    bundle install --with pg --without development test
4    RAILS_ENV=production rails db:migrate
5    RAILS_ENV=production rails assets:precompile
6    sudo service nginx restart
```

Fig 19: Deploying WebBoard to nodes

Rundeck was used to also run commands on all nodes, when common operations needed to be performed. We also created a "Add SSH Authorized Key" job as well to automatically add an SSH key to the authorized list of SSH keys.

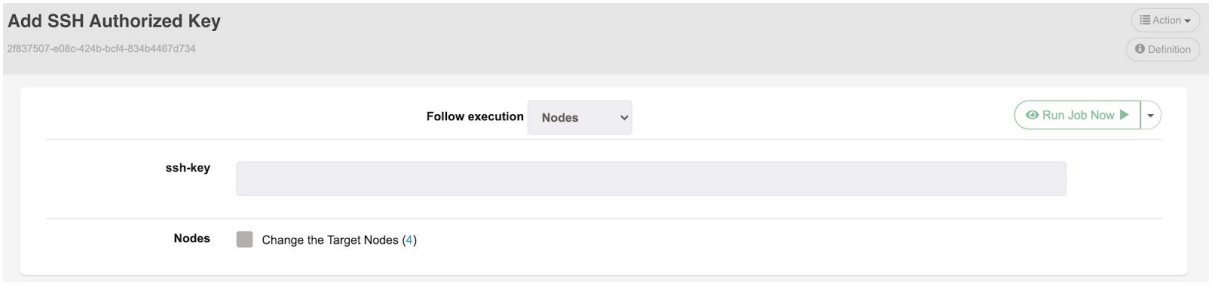

Fig 20: Adding authorized key to Rundeck

```
1    echo ${option.ssh-key} >> ~/.ssh/authorized_keys
```

Fig 21: Adding authorized key to Rundeck

5.9 Monitoring

We emphasized heavily on monitoring. We had a total of 4 monitoring tools integrated.

1. **DataDog** - Monitored production system logs
2. **ELK** - Monitored production application logs
3. **Metabase** - Business Intelligence tool for real time analysis of application data
4. **Google Analytics** - Business Intelligence tool for real time analysis of user traffic and user behavior

The combination of these 4 types of monitoring tools are very effective in understanding the performance of the product, the bottlenecks and the user response.

5.9.1 Datadog

Datadog is a SaaS based analytics platform which provides a monitoring service for monitoring of servers, containers and other forms of cloud applications. Datadog allows operations teams to visualize the performance of their full infrastructure in one place. It also provides support for adding alerts when certain critical events occur. A critical event could be low disk space, high load or a sudden drop in traffic.

Datadog was installed in all our VMs be executing the following command in Rundeck.

```
1   DD_AGENT_MAJOR_VERSION=7 DD_API_KEY=<INSERT API KEY HERE> \
2   bash -c "$(curl -L https://raw.githubusercontent.com/DataDog/datadog-agent/master/cmd/agent/install_script.sh)"
```

Fig 22: Installing Datadog through Rundeck

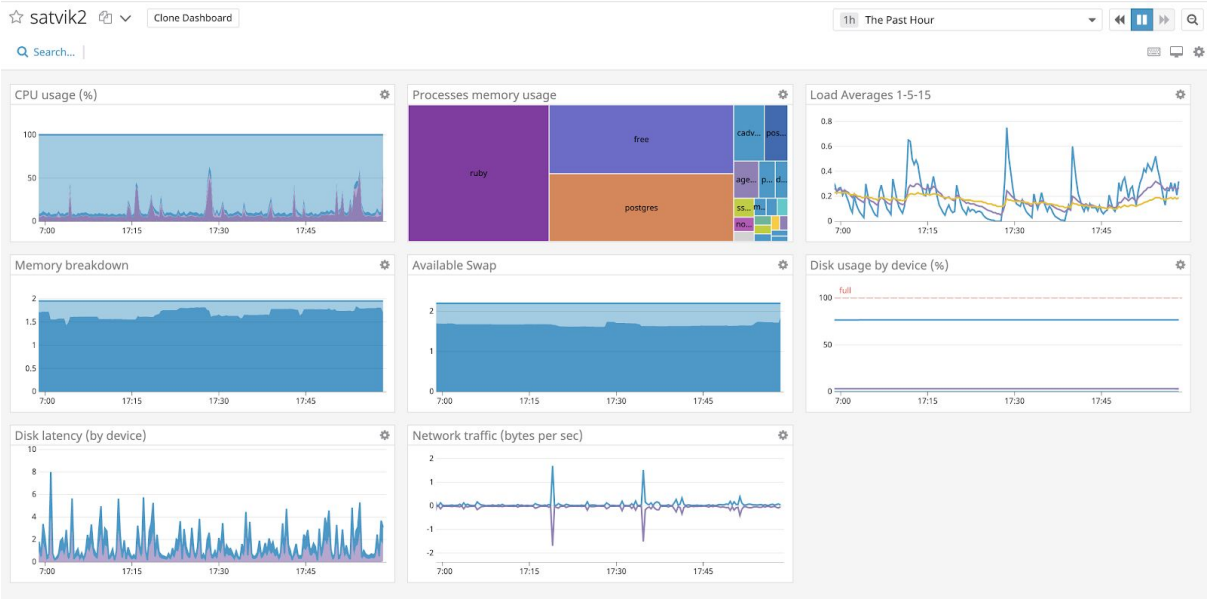

Fig 23: Datadog dashboard

5.9.2 ElasticSearch, Logstash, Kibana - ELK stack

Logstash [9] is a data processing tool that can collect data from multiple sources and send it to a desired location. In our application, Logstash instances run on each of the deployment nodes, sending logs to the ElasticSearch instance running on the monitoring node.

ElasticSearch [8] is an indexing and search engine for fast retrieval of data. In the context of our application, ElasticSearch receives logs from the Logstash instances running on both the deployment nodes and creates an index on this data for easy retrieval.

Kibana [10] is a visualization tool that can be built on top of either Logstash, ElasticSearch or any other data source. For us, Kibana provides easy visualizations that can help monitor the logs that are sent to ElasticSearch. The Kibana dashboard for our application is shown below

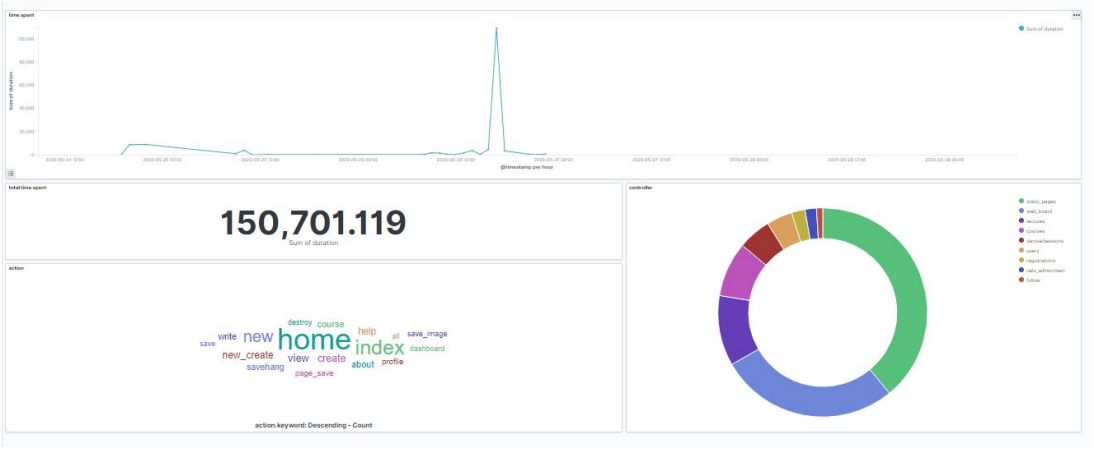

Fig 24: Kibana dashboard

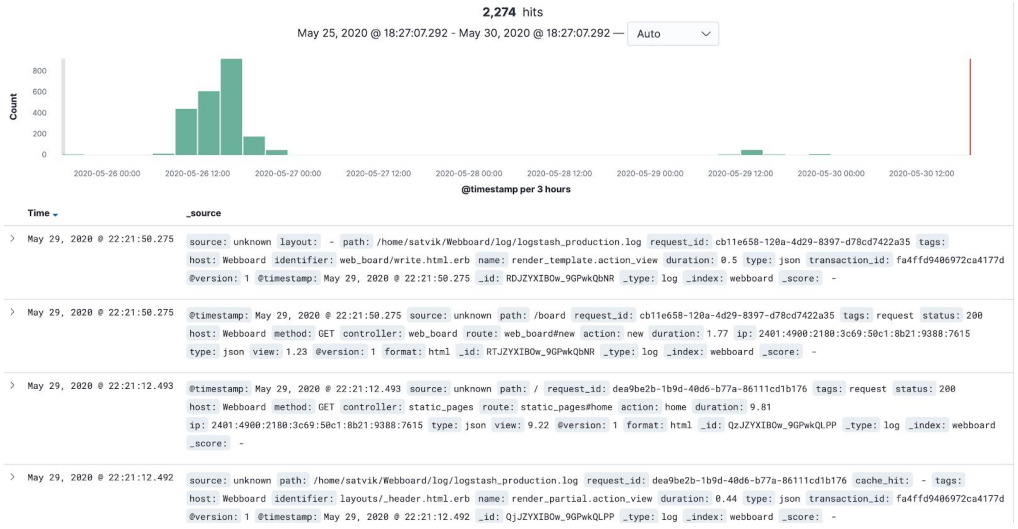

Fig 25: Kibana showing logs found by ElasticSearch

5.9.3 Metabase

Metabase is an extremely easy to use BI tool which allows users to create their own queries by using a very intuitive user interface. It also allows users to create excellent interactive dashboards. The following are screenshots from another project **CircuitVerse** which indicates how easy it is to design a query and create dashboards. We tried to create similar visualizations with webboard but due to lack of traffic, we couldn't create similar useful visualizations.

```
docker run -d -p 3000:3000 \
-v ~/metabase-data:/metabase-data \
-e "MB_DB_FILE=/metabase-data/metabase.db" \
--name metabase metabase/metabase
```

Fig 26: Creating a metabase container for WebBoard

Fig 27: Metabase showing new user visualization from CircuitVerse

Fig 28: Metabase visualizations from CircuitVerse

5.9.4 Google Analytics

Google analytics is a product by Google which allows website owners to analyze and monitor their website traffic real time. It also has support for other related produces like Google ad sense. It is an essential tool to track digital marketing effectiveness.

Google analytics is integrated by adding a snippet like the following in the html page.

```
<script>
    window.dataLayer = window.dataLayer || [];
    function gtag() { dataLayer.push(arguments); }
    gtag('js', new Date());

    gtag('config', 'UA-167527507-1');
</script>
```

Fig 29: Integrating google analytics to WebBoard

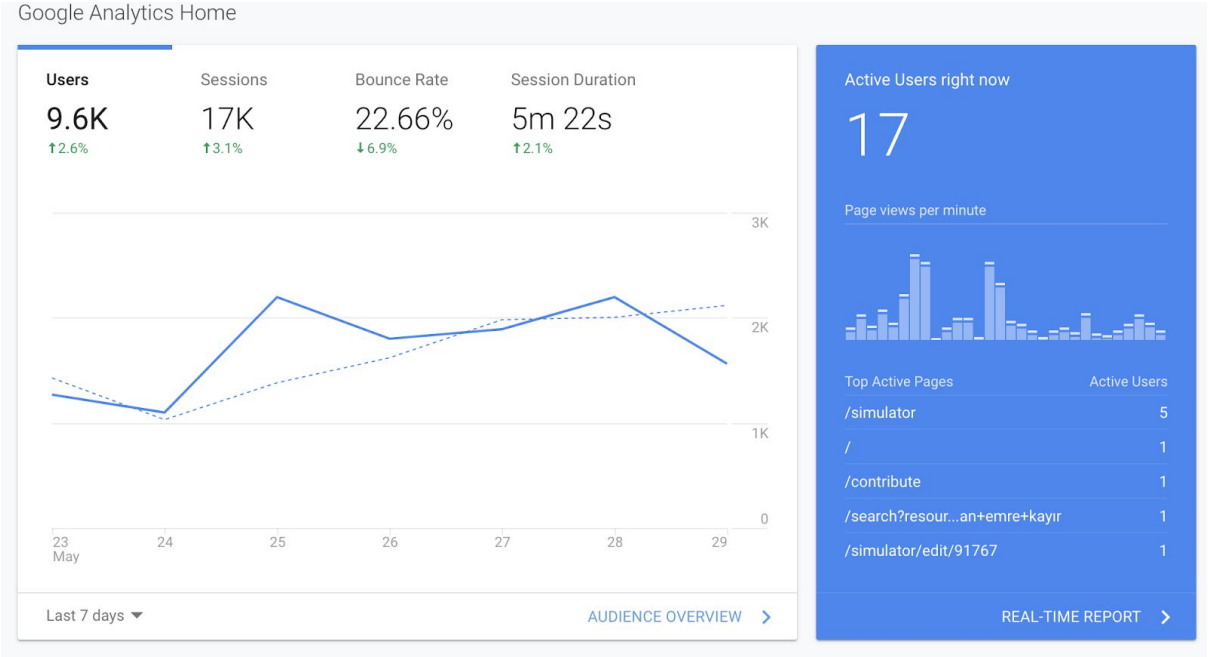

Fig 30: Google Analytics visualizations from CircuitVerse

6 Continuous Integration

Jenkins is a popular open source tool that allows us to setup a devops pipeline and practice CI, CD etc. What makes Jenkins so versatile are plugins which extend the functionality of Jenkins. We used several plugins like **Blue Ocean, Docker Pipeline, Git plugin, Github Plugin, JUnit Plugin, Mailer Plugin, Pipeline, Rake plugin, Rundeck plugin, Rvm, SSH Credentials Plugin** and many more.

The following is the screenshot of the entire pipeline.

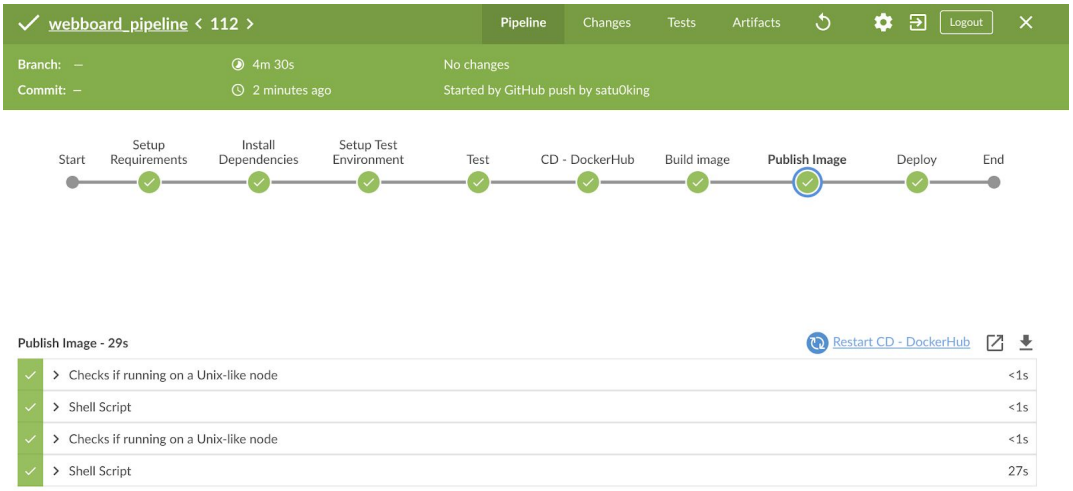

Fig 31: Pipeline on Jenkins

Our pipeline included the following stages:

1. Setup Requirements
2. Install Dependencies
3. Setup Test Environment
4. Test - run unit tests using RSpec
5. Build docker image - build a docker image using the Dockerfile
6. Publish docker image to dockerhub
7. Deploy to nodes - use Rundeck to deploy WebBoard to nodes

6.1 Github Webhook

In order to trigger the pipeline when a commit happens automatically, there were 2 options. First option was simple - use git poll SCM but this didn't seem like an attractive option as it was inefficient and felt not elegant. The second option was to use github webhooks. Essentially Github "notifies" Jenkins when a commit happens. Jenkins can then trigger the pipeline automatically then.

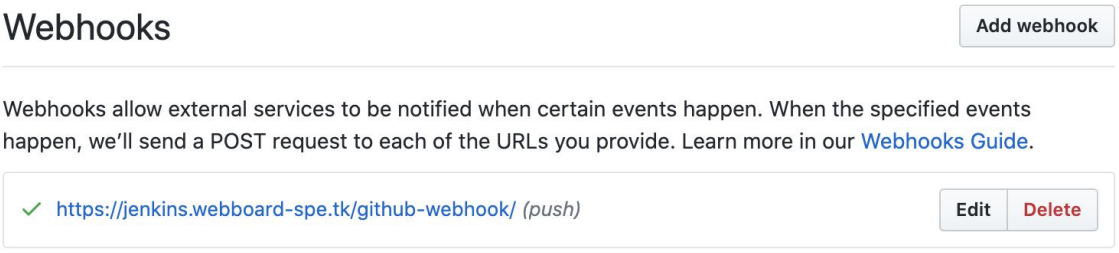

Fig 32: Adding Github webhook to Jenkins

6.2 Dockerhub Integration

We published the artifact (docker image) to docker hub. Dockerhub credentials was added to Jenkins as shown in the figure below.

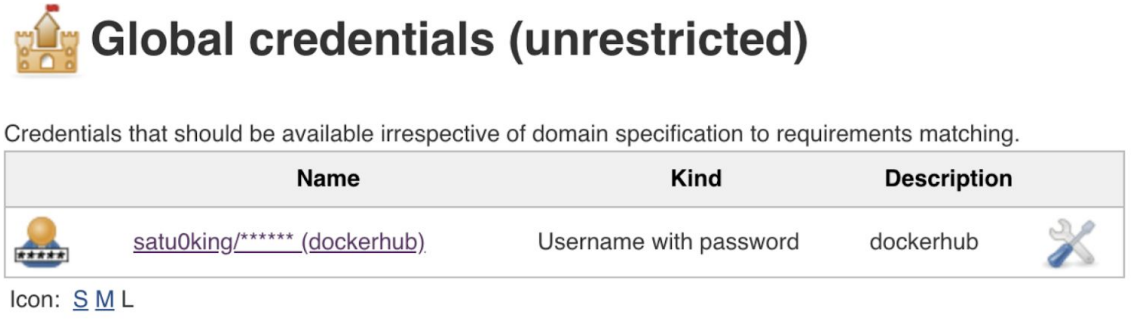

Fig 33: Adding dockerhub credentials to Jenkins

6.3 Rundeck Integration

Rundeck was integrated with Jenkins by adding the rundeck instance to Jenkins in the Configure Jenkins page as given in the figure below.

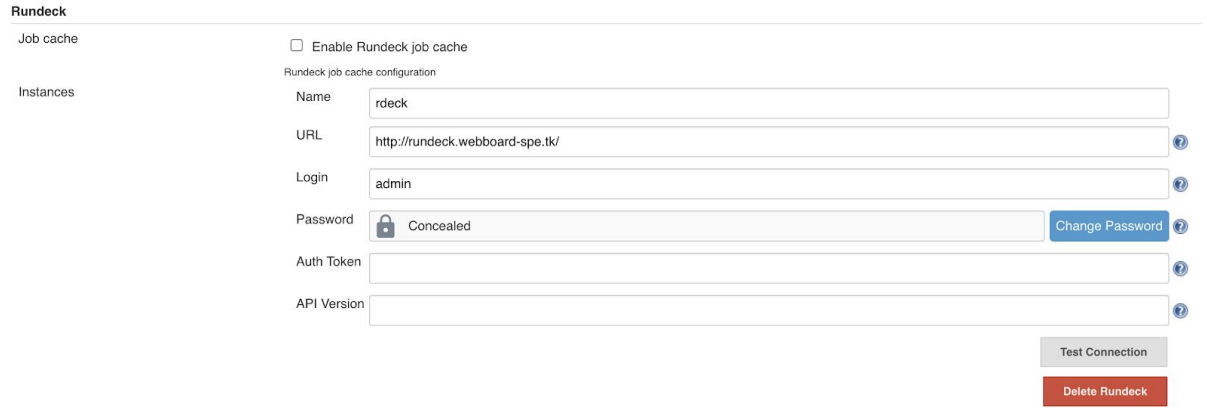

Fig 34: Adding rundeck instance to Jenkins

6.4 Pipeline Stages

6.4.1 Stage1: Setup Requirements

In this stage, we setup the environment for testing. Here we set the RVM home and installed bundler. We had a lot of trouble in setting up RVM. Then our guide suggested using environment variable and so we managed to write the script elegantly by using $RVM_HOME environment variable.

```
 9      stage('Setup Requirements') {
10        steps {
11          sh 'bash setup.sh'
12        }
13      }
```

Fig 35: Running the setup.sh script - Jenkinsfile

setup.sh

```
1    #!/bin/bash
2    source $RVM_HOME/scripts/rvm
3    gem install bundler
4    bundle install
```

Fig 36: setup.sh

6.4.2 Stage2: Install Dependencies

In this stage, all the required dependencies are installed. This is equivalent to *mvn install*. The first time it is run, it will take time to install the dependencies. Subsequent runs use the cached dependencies.

```
14      stage('Install Dependencies') {
15        steps {
16          sh 'bash build.sh'
17        }
18      }
```

Fig 37: Running the build.sh script - Jenkinsfile

build.sh

```
1    #!/bin/bash
2    source $RVM_HOME/scripts/rvm
3    bundle install
```

Fig 38: build.sh

6.4.3 Stage3: Setup test environment

We create the test environment database in this stage

```
19      stage('Setup Test Environment') {
20        steps {
21          sh 'bash test_setup.sh'
22        }
23      }
```

Fig 39: Running test_setup.sh script - Jenkinsfile

6.4.4 Stage3: Test

Run the actual tests and post results to JUnit plugin in Jenkins

```
24      stage('Test') {
25        steps {
26          sh 'bash test.sh'
27        }
28        post {
29            always {
30              junit 'rspec.xml'
31            }
32        }
33      }
```

Fig 40: Running test.sh script - Jenkinsfile

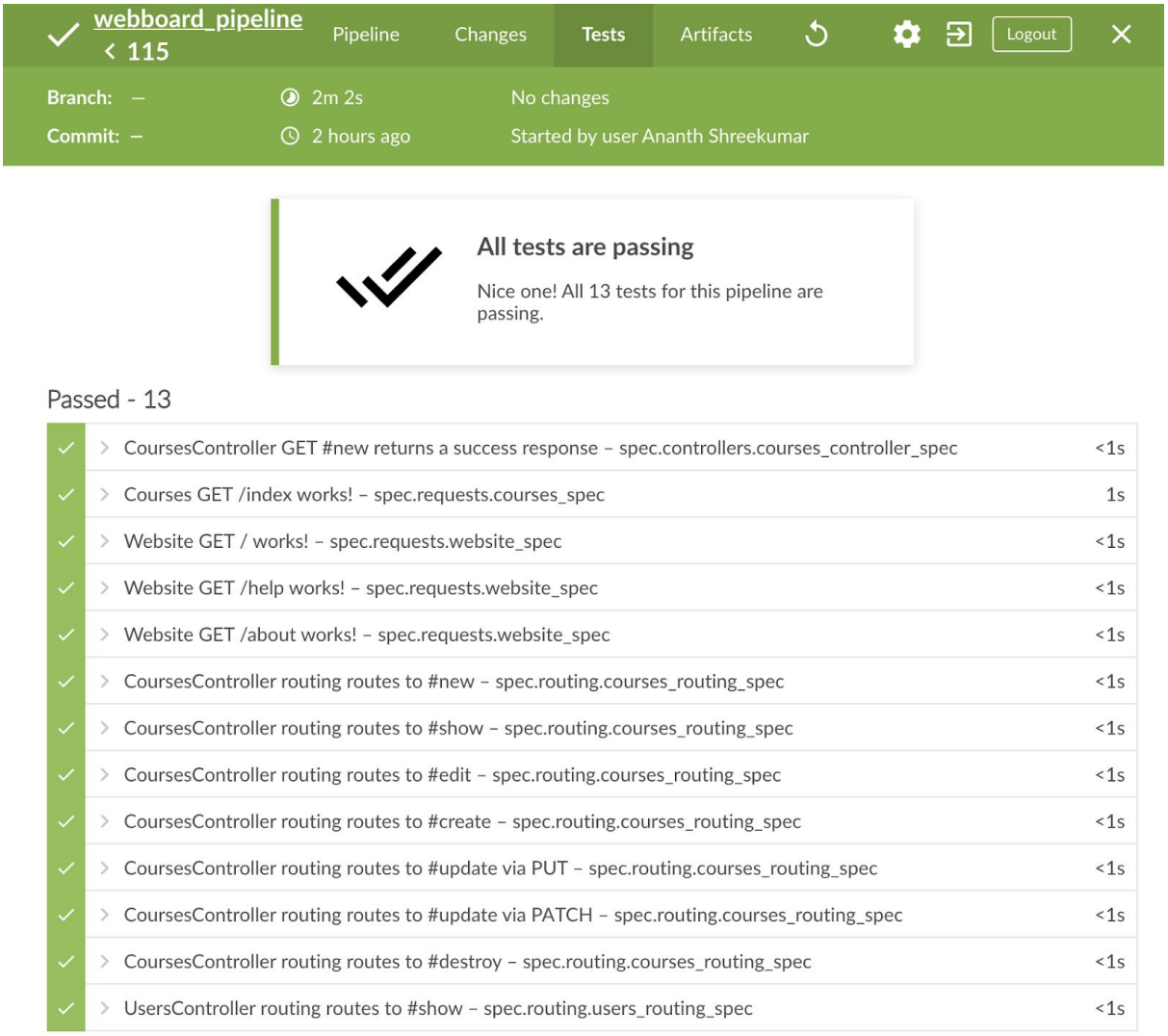

Fig 41: Tests passing on Jenkins

6.4.5 Stage 4: Build Docker Image

We build the docker image from the dockerfile and add the build number as tag.

```
36          stage('Build image') {
37            steps{
38              script {
39                dockerImage = docker.build registry + ":$BUILD_NUMBER"
40              }
41            }
42          }
```

Fig 42: Building image using Dockerfile - Jenkinsfile

6.4.6 Stage 5: Publish Artifact (Docker Image)

We push the docker image to dockerhub.

```
43          stage('Publish Image') {
44            steps{
45              script {
46                docker.withRegistry( '', registryCredential ) {
47                  dockerImage.push()
48                }
49              }
50            }
51          }
```

Fig 43: Publishing docker image to dockerhub - Jenkinsfile

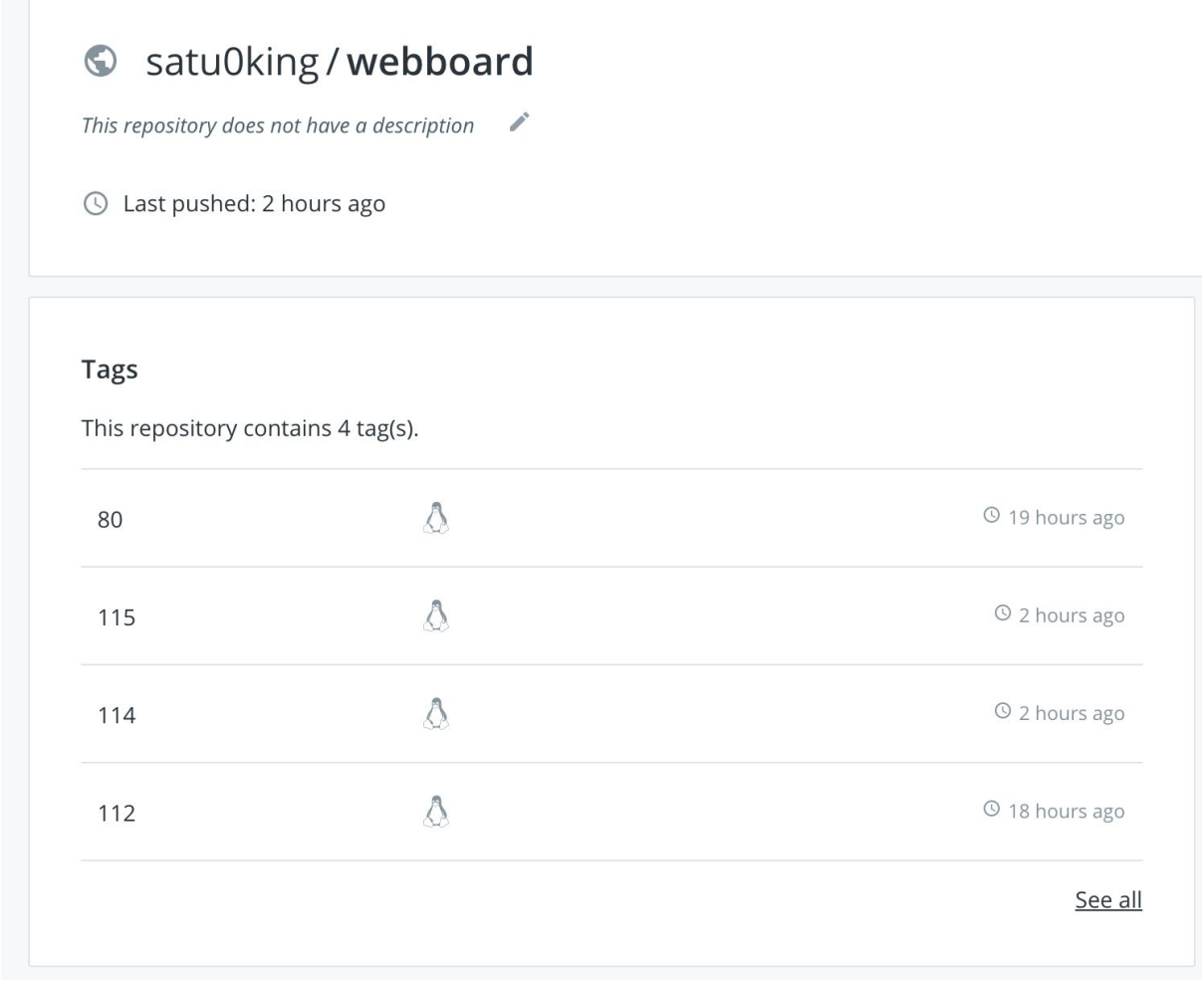

Fig 44: Dockerhub showing the published image for WebBoard

6.4.7 Stage 6: Deploy

We trigger the deploy job in the Rundeck instance.

```
54      stage('Deploy') {
55        steps {
56          script {
57            step([$class: "RundeckNotifier",
58            rundeckInstance: "rdeck",
59            jobId: "184e1419-091c-4728-a781-88da8438a91a"])
60          }
61        }
62      }
```

Fig 45: Triggering a rundeck job - Jenkinsfile

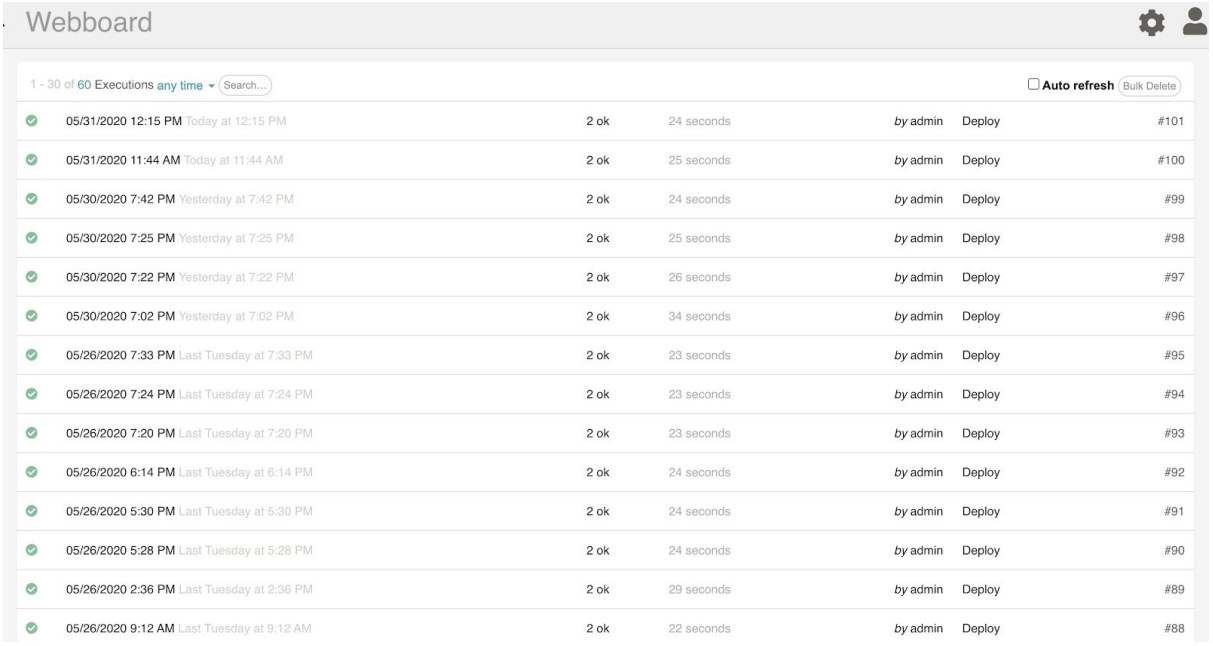

Fig 46: Successful deployments on rundeck

```
pipeline {
  environment {
    registry = "satu0king/webboard"
    registryCredential = 'dockerhub'
    dockerImage = ''
  }
  agent any
  stages {
    stage('Setup Requirements') {
      steps {
        sh 'bash setup.sh'
      }
    }
    stage('Install Dependencies') {
      steps {
        sh 'bash build.sh'
      }
    }
    stage('Setup Test Environment') {
      steps {
        sh 'bash test_setup.sh'
      }
    }
    stage('Test') {
      steps {
        sh 'bash test.sh'
      }
      post {
          always {
            junit 'rspec.xml'
          }
      }
    }
    stage('CD - DockerHub') {
      stages{
        stage('Build image') {
          steps{
            script {
              dockerImage = docker.build registry + ":$BUILD_NUMBER"
            }
          }
        }
        stage('Publish Image') {
          steps{
            script {
              docker.withRegistry( '', registryCredential ) {
                dockerImage.push()
              }
            }
          }
        }
      }
    }
    stage('Deploy') {
      steps {
        script {
          step([$class: "RundeckNotifier",
          rundeckInstance: "rdeck",
          jobId: "184e1419-091c-4728-a781-88da8438a91a"])
        }
      }
    }
  }
}
```

Fig 47: Jenkinsfile

7 Experimental Setup

7.1 Functional Requirements

7.1.1 Platform Requirements

1. Lecture notes should sync to cloud
2. Students should be able to securely access past lecture notes
3. Lecture notes should be editable in the future
4. Professor should able to organize the lecture notes
5. Administrators need to be able to manage courses, teachers etc.

7.1.2 Board Requirements

1. Professors should be able to write notes easily without much knowledge or training
2. Should support multiple colors, size and stroke styles
3. Should be able to open a PDF
4. Should save automatically
5. Should support standard CAD tools like cut/copy/paste and Undo Redo
6. Should have lecture playback feature

7.2 Nonfunctional Requirements

1. Should scale up well for upto at least 100 simultaneous users
2. Should be reliable
3. Should be robust to crashes and internet failures
4. Should not consume lot of bandwidth

7.3 Architecture and Design

7.3.1 Smart Board Architecture

A lot of good Design patterns and abstractions have been used in the software architecture. The board can be loaded on a lot of different types of displays, so the resolutions, the page size etc are important design decisions. Moreover, the board can be opened on different types of devices as well such as tablet, mobile, laptop, smartboard etc. Therefore it is essential to handle all these cases elegantly.

7.3.2 Abstraction 1: Canvas Wrapper API

This abstracts away the following things. The code for this can be found at *public/js/canvasApi.js*

1. Page Size
2. Display Resolution
3. Display Type
4. Page position

7.3.3 Abstraction 2: Listener Adapters

This abstracts away the following things. The code for this can be found at *public/js/listener.js*

The following devices have very different user inputs. This wrapper unifies them.

1. Mobile Devices
2. Laptop Devices
3. Tablet Devices
4. Smart Board Devices

7.3.4 Abstraction 3: Stroke Class

Every type of drawing object is a subtype of this *Stroke* class. This helps in generalization and reuse. The different types of stroke objects are *PenStroke, LineStroke, CircleStroke, QuadStroke, Eraser.*

```
class Stroke {
    constructor(startTime, width, color) {
        this.startTime = startTime;
        this.width = width;
        this.color = color;
    }

    update(position);
    mouseLeave(position);
    stop(position);
    render(position);
    contains(position);
    move(position);
    centerOfMass();
}
```

Fig 48: Stroke Class Structure

7.3.5 Command Pattern

Command pattern is used heavily in the system. Each stroke is also a command object. It is used for the following things.

1. Multiple Stroke Selections (Selection of specific commands)
2. Undo Redo - (Standard usage of command pattern)
3. Copy Paste - (Commands are copied to clipboard)
4. Incremental Rendering - (Rendering can be done incrementally for efficiency)

7.3.6 Schema overview

The following image is the basic schema design.

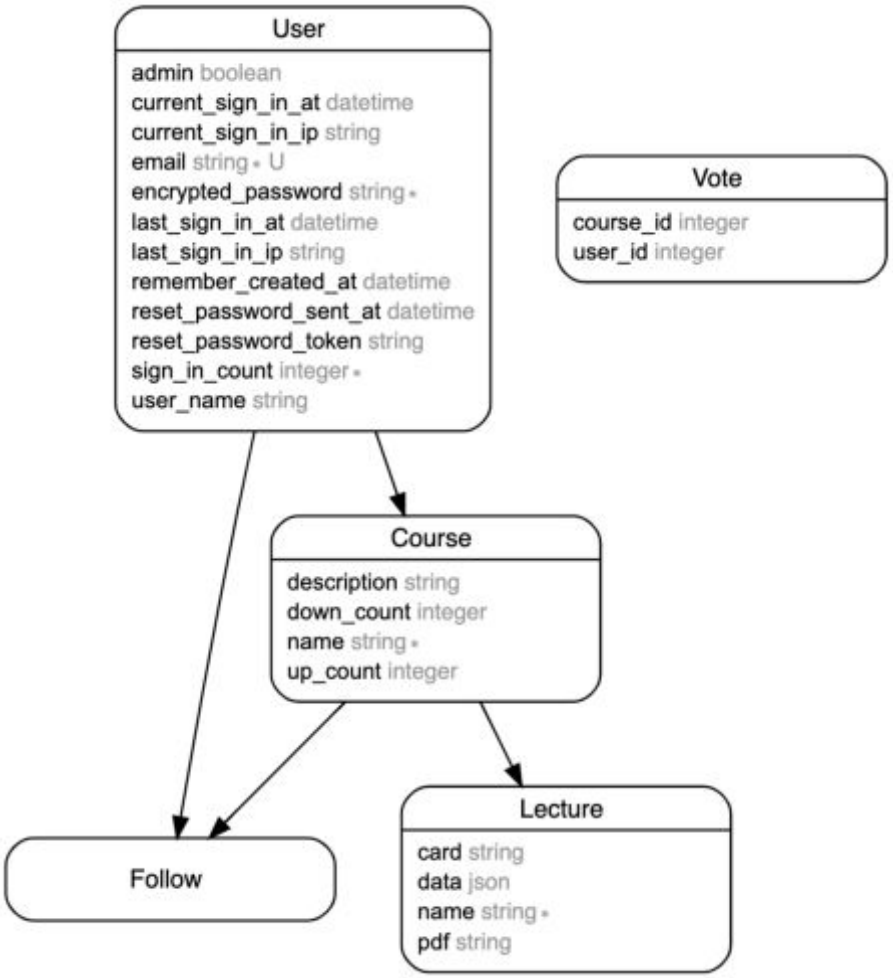

Fig 49: WebBoard Database schema

7.3.7 User flow overview:

The following diagram describes the user flow.

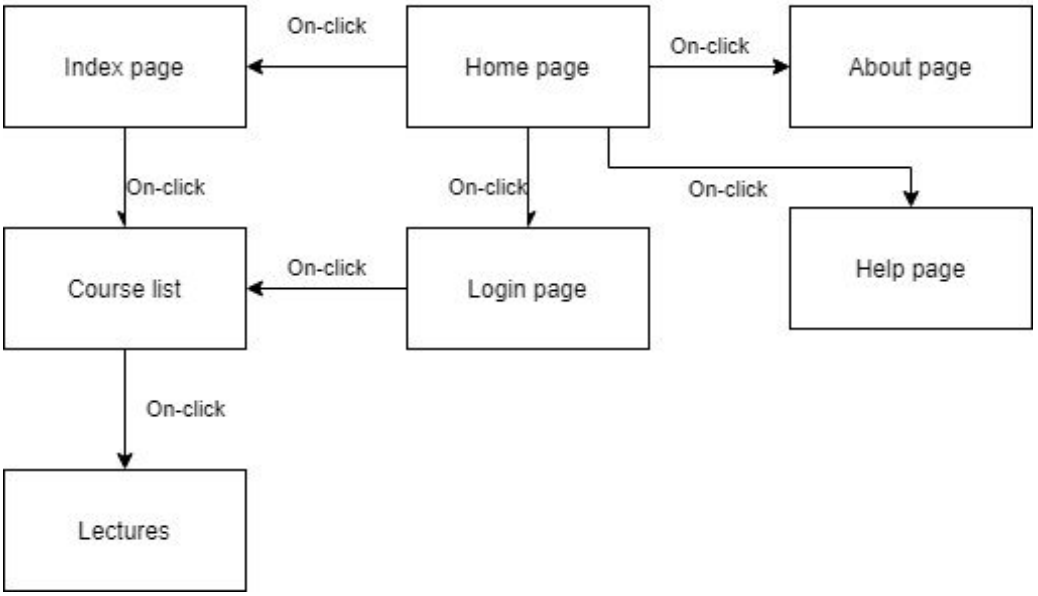

Fig 50: WebBoard architecture and workflow

8 Features Implemented

8.1 Feature - Undo/Redo

A highly requested feature was the ability to perform undo operation. This was essentially implemented by the direct usage of command pattern.

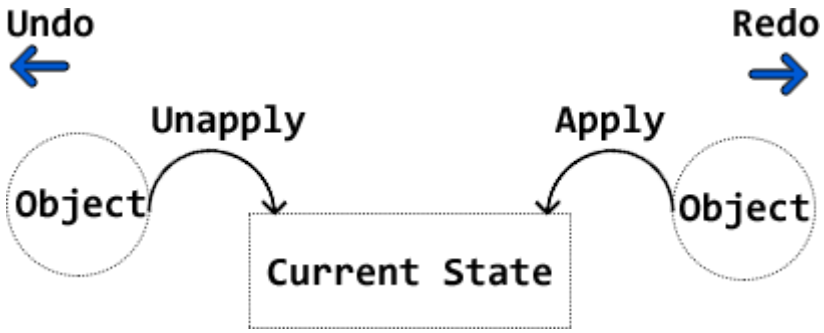

Fig 51: Undo Redo Stack (Command Pattern)

8.2 Feature: Lecture Playback

Very often, the professors don't write notes in a sequential manner. They often refer back to previous pages etc. Sometimes they even erase the page and start over it. These are common sources of frustration for the students. Therefore we implemented the Lecture Playback feature.

This was also done by the usage of command patterns. Each stroke object was a command. And each command was also annotated with a start time. This is incredibly useful as we can now order the command objects across pages with time. We can see how the lecture progressed exactly as it did with time.

Steps followed:

1. Each stroke command across pages were ordered
2. *renderHandler* handled the rendering of the page with time
3. The playback speed can easily be handled by tweaking the sleep time between frames.

```
1   function renderHandler(index) {
2       let strokeObject = animationObject.strokeList[index];
3       renderStateObject(strokeObject);
4       animationObject.renderHandler = undefined;
5       if (index + 1 < animationObject.strokeList.length) {
6           let handler = renderHandler.bind(null, index + 1);
7           let nextStrokeObject = animationObject.strokeList[index + 1];
8           let speed = parseInt(document.getElementById("Speed").value);
9           animationObject.renderHandler =
10              setTimeout(handler,
11                  Math.max(1, Math.min(1000, (nextStrokeObject.time - strokeObject.time) / speed)));
12      }
13      document.getElementById("animationSlider").value = index;
14  }
```

Fig 52: Render Handler Code

Fig 53: Lecture Playback Features

8.3 Feature: Selection and Copy/Cut/Paste

A much requested feature was the ability to select notes and the ability to move, cut, copy, paste etc. This is an essential feature because its available in all CAD tools. This feature was tricky to implement.

This feature also uses command pattern. When the area is selected, we go through every stroke object and check if the stroke falls in that area. We mark that stroke object as selected. Then we can copy it to clipboard by copying the selected stroke command objects.

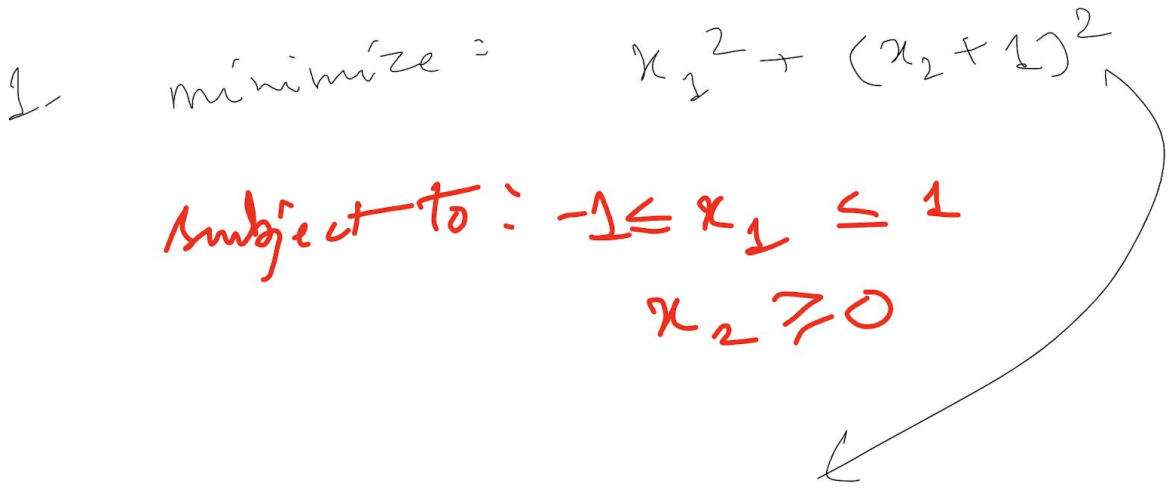

Fig 54: Red Text is selected

9 Results and Discussion

Home Page

- Users can create an account and login into the application to manage his lectures.
- Users can explore through the course contents.
- A new digital board can be launched where the user can record a lecture.

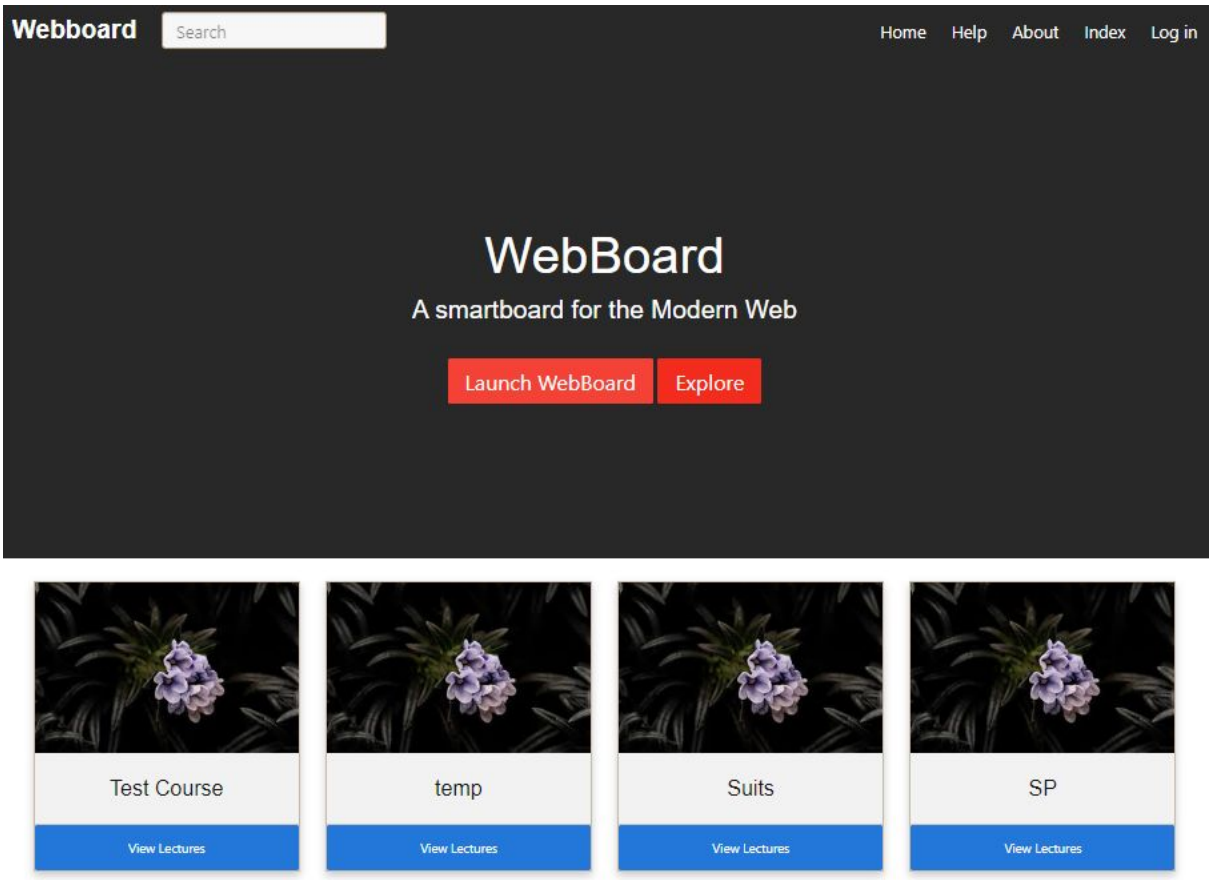

Fig 55: WebBoard homepage

WebBoard

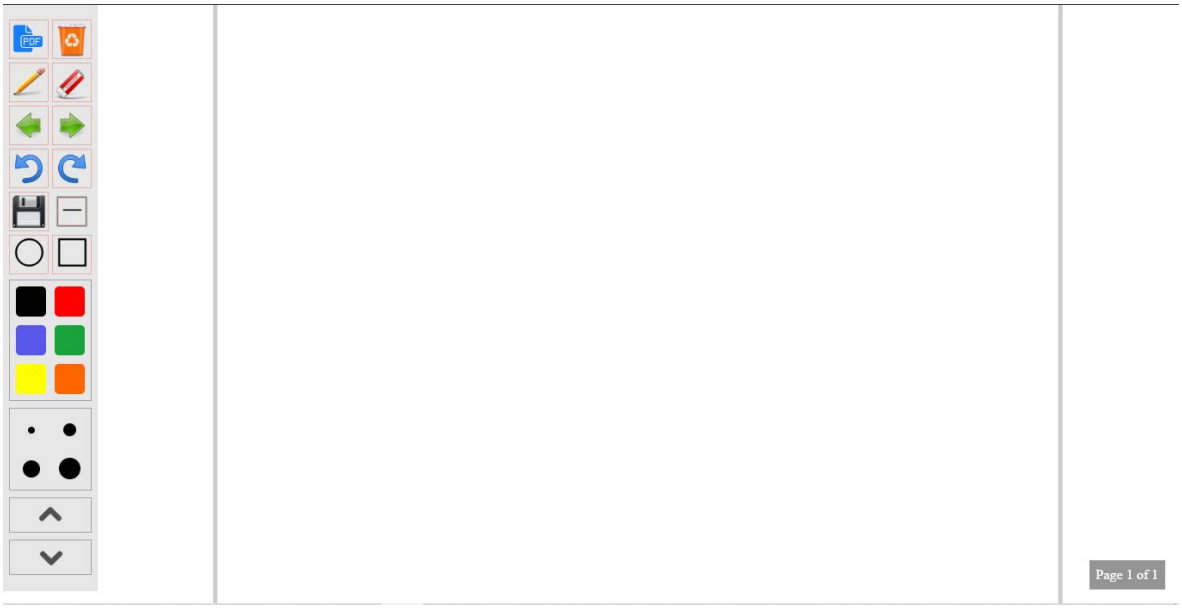

Fig 56: WebBoard writing interface

- Users can import an existing pdf into the lecture to work on.
- Pen - to write on the digital board, different colour and brush options are available for the pen.
- Users can undo and redo.
- The board can be extended to an infinite number of pages.
- A user can save the lecture to a course.
- Users can draw geometrical shapes easily.

The image below shows the result of using tools for drawing geometrical shapes and using different colours and brush sizes for the pen.

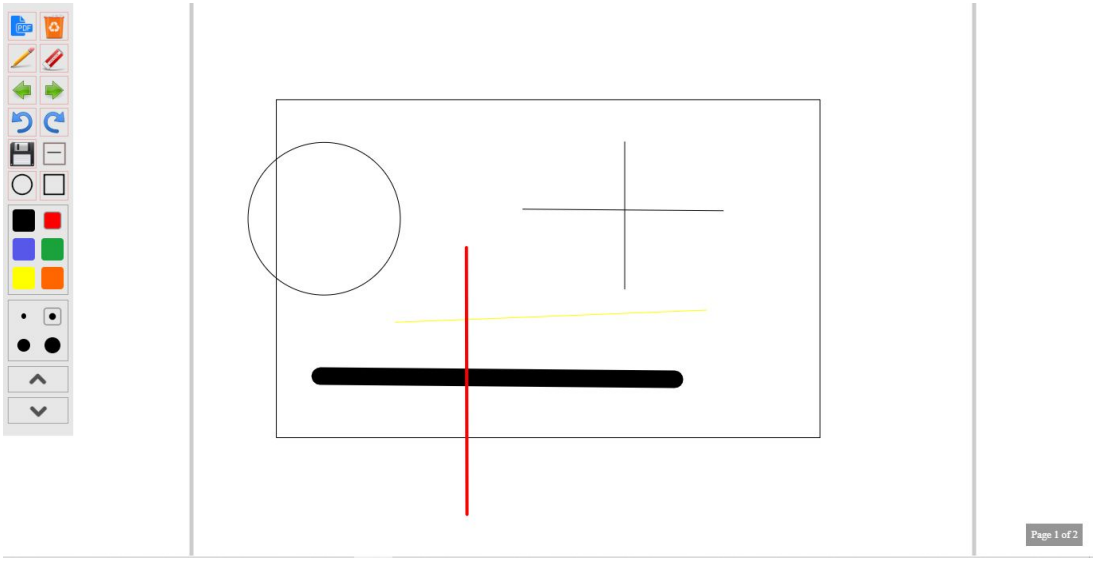

Fig 57: WebBoard allows drawing simple shapes

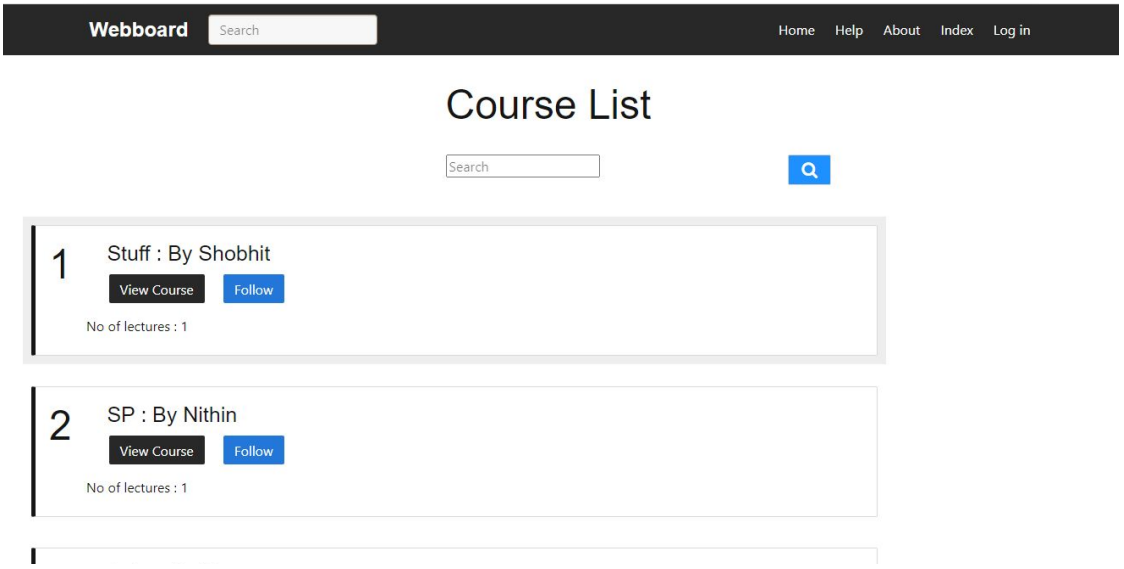

Fig 58: WebBoard index page showing course list

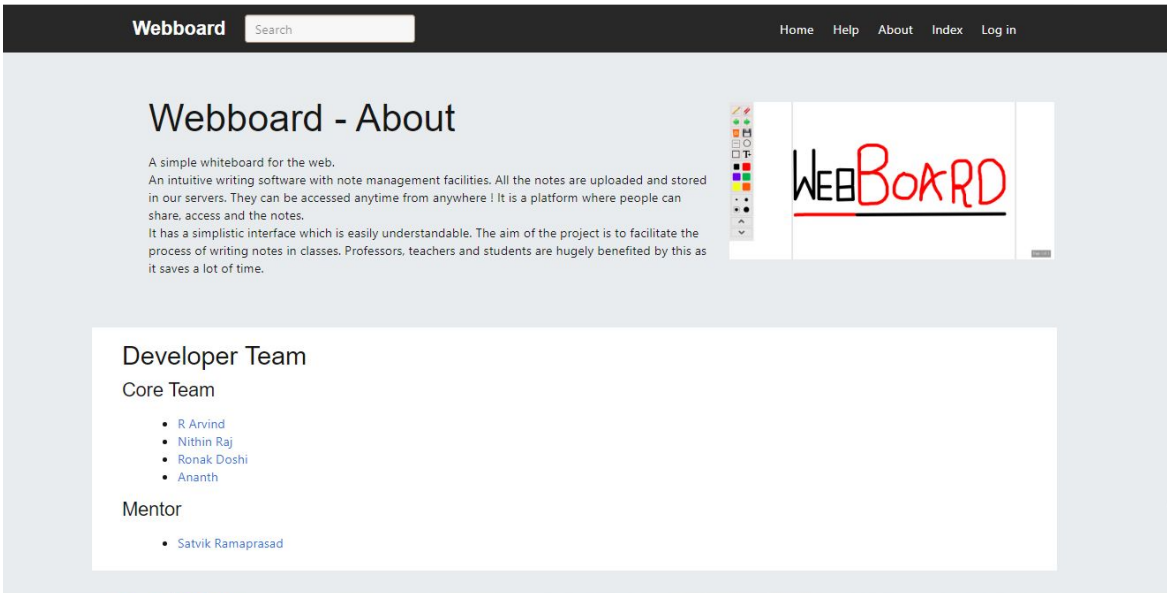

Fig 59: WebBoard About page

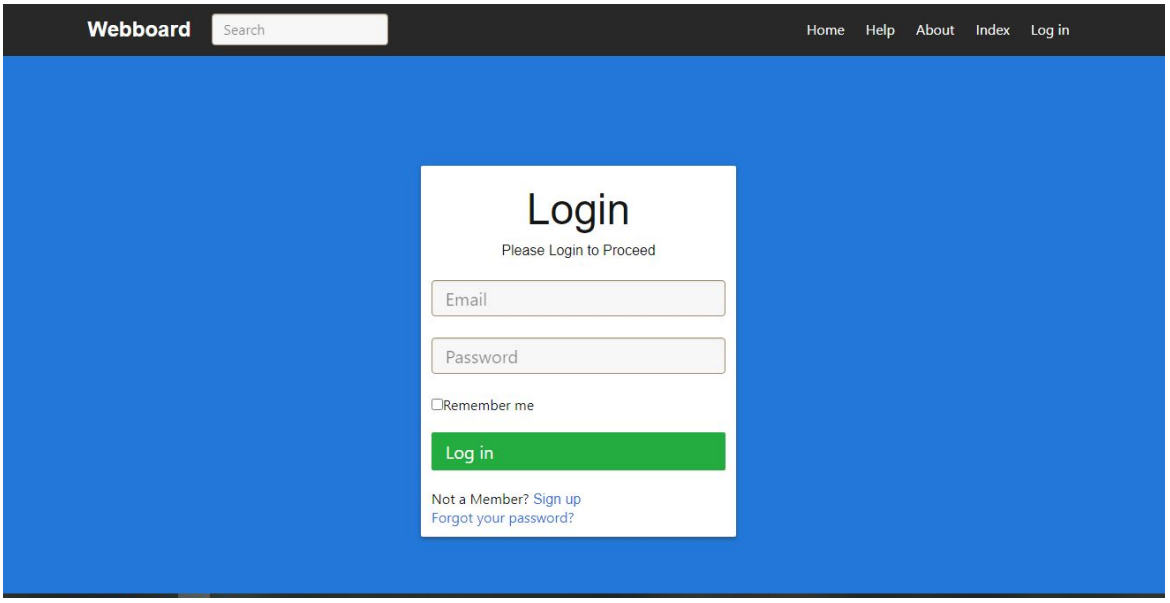

Fig 60: WebBoard login page for users

Lecture Recording

- Users can playback the lectures, exactly the way it was taught.
- Users can control the speed of playback.
- Users can scroll to the exact position in the lecture as their need.

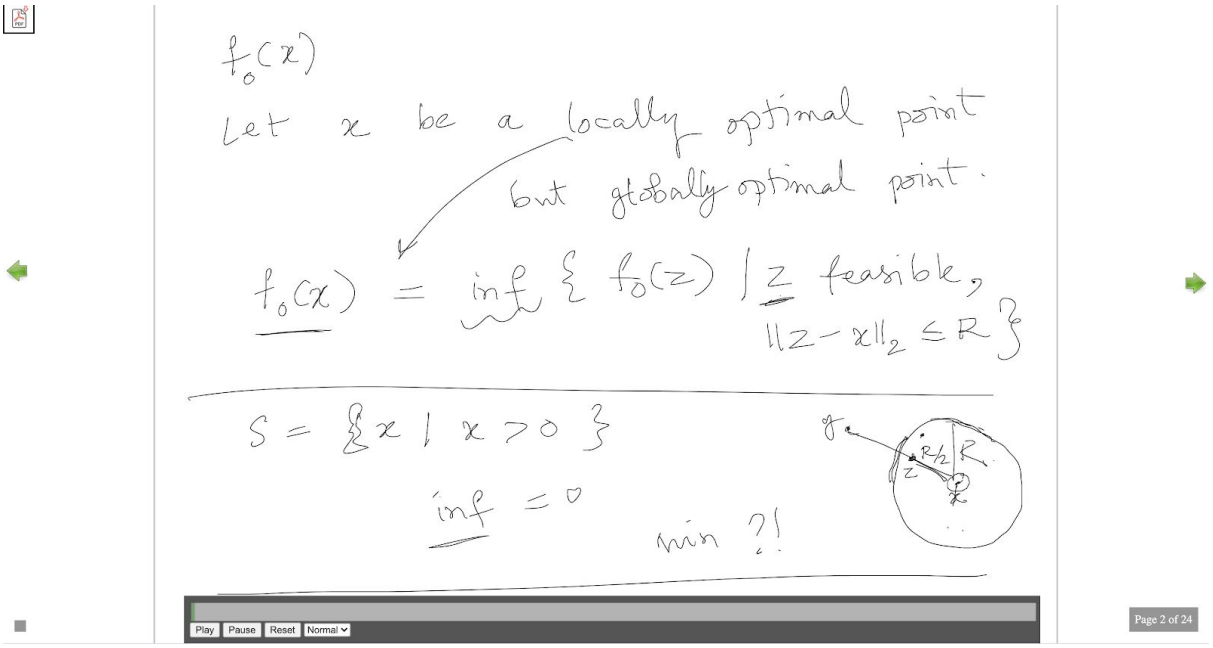

Fig 61: Playback of a lecture

10 Scope for future work

- Real time collaboration feature
- Introducing inbuilt Poll feature
- Introducing inbuilt Doubt asking feature
- Users can add tags to the "play-bar" denoting the topics being covered at a specific instance.

11 Conclusion

Webboard was successfully enhanced using several DevOps principles and multiple tools that increased efficiency while also allowing for easy monitoring of the product once deployed. We have managed to successfully deliver several highly requested features.

12 References

[1] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[2] S. Ruby, D. B. Copeland, and D. Thomas, *Agile Web Development with Rails 6*. Pragmatic bookshelf, 2020.

[3] J. F. Smart, *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. ” O’Reilly Media, Inc.”, 2011.

[4] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*. ” O’Reilly Media, Inc.”, 2012.

[5] B. Wilder, *Cloud architecture patterns: using microsoft azure*. ” O’Reilly Media, Inc.”, 2012.

[6] D. Spinellis, “Service orchestration with rundeck,” *IEEE Software*, vol. 31, no. 4, pp. 16–18, 2014.

[7] M. Taylor and S. Vargo, *Learning Chef: A Guide to Configuration Man- agement and Automation*. ” O’Reilly Media, Inc.”, 2014.

[8] C. Gormley and Z. Tong, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. ” O’Reilly Media, Inc.”, 2015.

[9] J. Turnbull, *The Logstash Book*. James Turnbull, 2013.

[10] Y. Gupta, *Kibana essentials*. Packt Publishing Ltd, 2015.

[11] A. Reelsen, “Using elasticsearch, logstash and kibana to create real- time dashboards,” *Dostupn´e z: https://secure. trifork. com/dl/goto-berlin- 2014/GOTO Night/logstash-kibanaintro. pdf*, 2014.

[12] B. Clifton, *Advanced web metrics with Google Analytics*. John Wiley & Sons, 2012.

[13] G. Smith, “Log analysis with the elk stack (elasticsearch, logstash and kibana),” 2015.

[14] M. Marschall, *Chef infrastructure automation cookbook*. Packt Publishing Ltd, 2015.

[15] W. Reese, “Nginx: the high-performance web server and reverse proxy,” *Linux Journal*, vol. 2008, no. 173, p. 2, 2008.

[16] C. Nedelcu, *Nginx HTTP Server: Adopt Nginx for Your Web Applications to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever*. Packt Publishing Ltd, 2010.

[17] M. Stonebraker and L. A. Rowe, “The design of postgres,” *ACM Sigmod Record*, vol. 15, no. 2, pp. 340–355, 1986.