Housekeepers

Final Project Report:

IIIT Bangalore
CS816-Software Production Engineering

Project By:
IMT2020129 Deep Shashank Pani

Under the guidance of
Prof. B. Thangaraju

Table of contents

Abstract

Due to the varying schedule of students, getting their room complaints resolved is one of the most important concerns in hostels. Students can now get their complaints noted down and resolved with Housekeepers.

It does this by letting the students add complaints in accordance to their need and priority, depending of the urgency of the complaint. The administrator looks at the complaint dashboard and lets the student know that their complaint is being resolved. Once the issue is resolved, the student then marks their complaint as resolved. This method is done for verifying whether the complaint actually got resolved or not. This reduces the overhead of the student and the hostel wardens.

Introduction:

1. Why DevOps?

DevOps is an organizational approach that facilitates faster application development and
easier maintenance of existing deployments by combining development (Dev) and
operations (Ops) teams.

DevOps encourages the use of best practices, automation, and new tools to create shorter,
more controllable iterations. It maximizes efficiency with automation and improves speed
and stability of software development lifecycle(SDLC).

DevOps is required in situations where there is high frequency of deployment. This project needed
DevOps because the pace of development was quite fast and a lot of new features were being added in short periods of time.

2. About the project

Registration

The first step of the application is user registration. Currently the admin registrations are blocked because if not, bad actors will signup as admins and cause major harm to the complaints database.
They are seeded as of now. Only student registrations are allowed as of now.

Students use their name, their email address, their room nos and a strong password to sign up for the application.

Login

Students login to the site using the email-id and password set during the registration.

Admins login to the site using the email-id and password provided to them.

Student Dashboard

The student dashboard appears after the student has logged in. It displays all the complaints they made so far, and a form that creates new complaints. The form has two fields: Complaint description and Priority(1-3).

The complaint list is sorted according to priority/urgency of the complaint, then by earliest created when both priorities are equal.

The student has the option to delete complaints which are NOT in the resolving stage. This restriction is added because the resolving stage means that the admin has looked at your complaint and has sent a personnel to your room.

Once the complaint is actually resolved, the student will mark the complaint as resolved and has the option to delete it, or keep it for archival purposes.

Admin Dashboard

The admin dashboard appears after the admin has logged in. It displays all the complaints of all the students made so far. This list is again sorted by priority, then by earliest created.

The admin has only one option, to mark complaints sent to the database as ‘Resolving’. This is done as an acknowledgement to the student that their complaint is being looked at.

Each complaint in the list contains the description, priority, and details of the student who made that complaint.

Tech Stack:

Coding:

1. Database: MongoDB Atlas

To ensure the scalability and availability of the project, MongoDB atlas cloud database has been used. This is done so that all admins can access this database from any device running the servers. The overhead with localDBs is that they have to synced, Atlas removes the synchronization overhead. The database is sharded for faster queries.

Other NoSQL systems could have been used, but MongoDB is one of the fewer wide known solutions to maintain ACID(atomicity, consistency, isolation,durability) properties.

2. Backend: Mongoose + Express + Nodejs

Mongoose is a library that helps you create and manage MongoDB objects in NodeJS applications.

Express.js, or simply Express, is a free and open source back end web application framework for building RESTful APIs with NodeJS.

Node.js(NodeJS) is a cross-platform, open-source JavaScript runtime environment. Node.js runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting.

3. Frontend: React + Nodejs

React is a free and open-source front-end JavaScript library for building user interfaces based on components.

Source Code Management: Git,Github

Source control Management is used for tracking the file changes history, source code etc. It helps us in many ways in keeping the running project in a structured and organized way. It allows multiple developers from the same team to work on a single project.
In this project, we'll be using Git and GitHub as SCM tools. Git is a version control tool that lets you manage different versions of the code inside a repository. Github is a host that stores git repositories.

Building: npm

npm(node package manager) is a packaging tool use to build and run NodeJS code, and to install NodeJS packages such as mongoose,express and react.

Testing: Jest,Supertest

Jest and Supertest are testing libraries that test JS code. They can test a wide variety of codes, from simple hello world functions to promises that return http responses. Here they are used to test backend API calls.

CI/CD pipeline: Jenkins

Continuous integration is a DevOps practice, where code changes are merged into a repository, after which automated builds and tests are run.
Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

Containerization: Docker,Dockerhub

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment supporting docker.

DockerHub is the Github of Docker. Instead of hosting source codes, it hosts docker images.

Deployment: Ansible

Ansible is an open source IT configuration management, deployment, and orchestration tool. It empowers DevOps teams to define their infrastructure as a code in a simple and declarative manner.
Ansible is an agentless tool, meaning that hosts who run the app need not install its dependencies.

Monitoring: ELK Stack

The ELK stack is a set of three logging tools: Logstash, Elasticsearch, and Kibana.
Logstash takes a wide variety of inputs (here it takes log files), filters them as per user's needs with the help of grok patterns, and outputs them to different types of destinations. In the ELK stack, logstash outputs to Elasticsearch.
Elasticsearch searches, stores, and analyses the database. It is a RESTful tool, and stores data in the form of JSON documents. It shards and then indexes the data for faster searching and scalability.
Kibana is a real-time dashboard for elasticsearch. It can perform visualizations on lots of data, including time-series data, geospatial data, relational graphs, etc.

Configuration

Coding:

sudo pacman -S nodejs
git clone https://github.com/Bimg-Brein42069/housekeepers-devops
cd housekeepers-devops

In server folder:

cd server
npm install
npm run dev (start the server)

In client folder:

cd client
npm install
npm start (start the client)

DevOps:

sudo pacman -S jenkins docker ansible sshpass

Configure jenkins, docker and ansible from their tutorials if not already done.

Add jenkins user to docker group

sudo usermod -aG docker jenkins

Reboot to save changes.

Then,

sudo systemctl enable --now jenkins
sudo systemctl enable --now docker

Monitoring

git clone https://github.com/deviantony/docker-elk
cd docker-elk
docker compose up setup (only once)
docker compose up

SDLC w/ DevOps

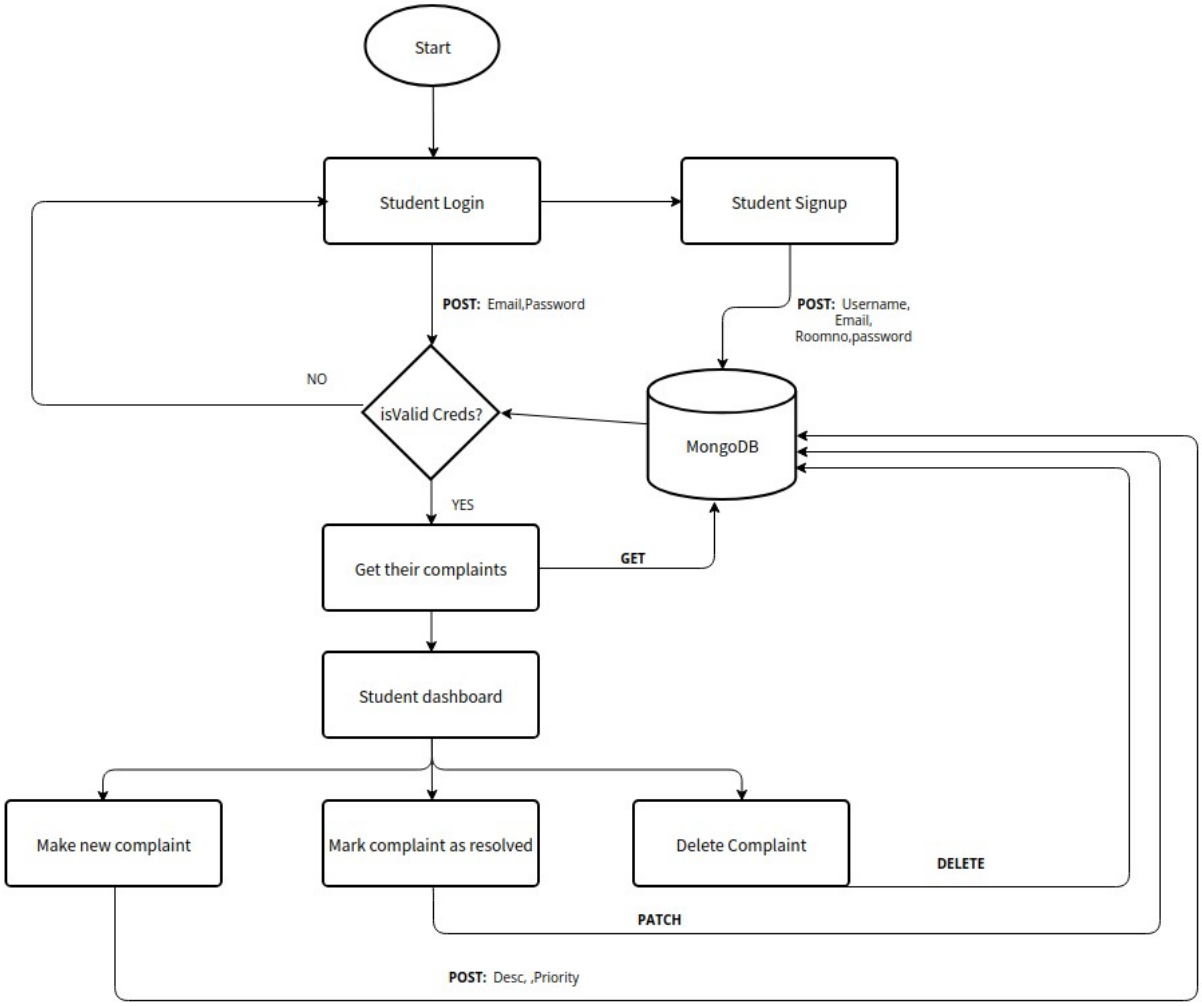
SDLC

Project Scope:

The scope of the application is to simplify the process of grievance redressal by making a website where complaints can be posted by students. Additionally, administrators can also view complaints, manage them and successfully resolve them. The objective of the project is to make the process of grievance redressal more transparent. Such transparency is beneficial to both students who post the complaints and administrators who resolve the complaints as it eliminates redundancy. The application allows students to post complaints in various domains. Then the concerned administrators can take actions to resolve.

Project Workflow:

1) **Students**

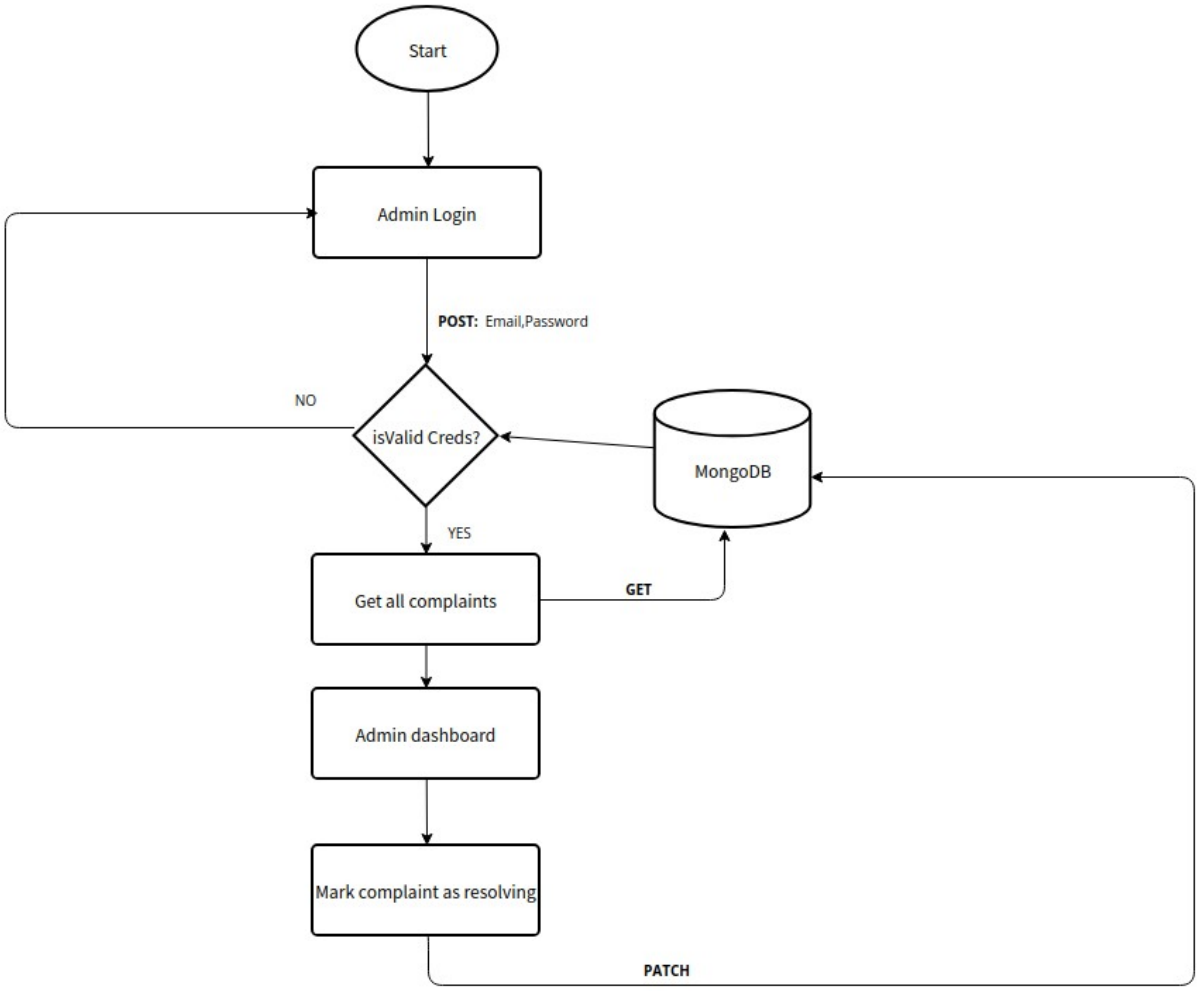


- **Registration:** Before using the app, the student first registers on the site. They provide their details, i.e. , name,email-id,roomno, password. This data is sent to the MongoDB database with a POST request.
- **Login:** After signing up and logging out, the student logins into the site. They provide their details, i.e. , email-id and password. The backend validates these credentials using a POST request, and if validated, then sends the student to the dashboard
- **Dashboard:**
  The student dashboard has a list of complaints they made, sorted in priority order, then by earliest created. It also has a form on the right hand side to create complaints
  - **New complaint:** The student can describe his complaint and give it a priority. If they want it to be resolved urgently, then they can give it a higher priority.
  - **Update complaint:** The student can click the update button on complaints marked as Resolving. Resolving status means that the

complaint has been acknowledged and personnel has been sent to resolve the problem.

- **Delete complaint:** The student can delete complaints with the statuses Sent and Resolved. Sent means that the complaint has been sent to the admins to look at. Resolved means that the complaint has been resolved. This feature is in case the student accidentally creates a complaint they didn't need any help for, or to delete the complaint in case it gets cluttered.

2) Admin



- **Registration:** The admins are manually added to the database. They are provided with a email-id and password to login. They do not have a signup page, because then anyone can join in as admin and potentially cause security issues.
- **Login:** Same deal as student login
- **Dashboard:**

The admin dashboard displays all complaints made by all the students, each complaint indicating the student name and roomno alongwith the complaint details. This list is sorted in the same manner as students dashboard complaints list. The admin can only do one thing: mark complaints with Sent status to Resolving status, which means the admin has looked at the complaint and has sent personnel accordingly.

API Documentation:

a) Student signup

- **Endpoint:** https://localhost:5000/api/user/signup
- **HTTP Method: POST**
- **Description:** Adds new student to the database
- Request body(req.body):

```
{
    name: “name”,
    email: “email@smtp.com”,
    roomno: “room000”,
    password: “<strongpassword>”
}
```

- Response body(res.body) if successful:

```
{
    isAdmin:false,
    email: “email@smtp.com”,
    token: jwt_token
}
```

- Response body(res.body) if failed:

```
{
    error: err.message
}
```

b) Admin signup

- **Endpoint:** https://localhost:5000/api/user/signupadmin
- **HTTP Method: POST**
- **Description:** Adds new admin to the database
- Request body(req.body):

```
{
    name: “name”,
```

email: “email@smtp.com”,
roomno: “room000”,
password: “<verystrongpassword>”
}

- Response body(res.body) if successful:

{
isAdmin:true,
email: “email@smtp.com”,
token: jwt_token
}

- Response body(res.body) if failed:

{
error: err.message
}

c) Login

- **Endpoint:** https://localhost:5000/api/user/login
- **HTTP Method: POST**
- **Description:** Logs in the user if valid credentials provided.
- Request body(req.body):

{
email: “email@smtp.com”,
password: “<theirpassword>”
}

- Response body(res.body) if successful:

{
isAdmin:<isuseradmin>,
email: “email@smtp.com”,
token: jwt_token
}

- Response body(res.body) if failed:

{
error: err.message
}

d) Get complaint list

- **Endpoint:** https://localhost:5000/api/complaints/
- **HTTP Method: GET**
- **Description:** Gets complaints according to user perms. If admin then gets all the complaints, else gets only their complaints.

- Request body(req.body):
  {
    _id: <user_id>
  }
- Response body(res.body) if successful:
  {
    ...complaint
  }

e) Get single complaint

- **Endpoint:** https://localhost:5000/api/complaints/:id
- **HTTP Method: GET**
- **Description:** Gets complaint with the id.
- Request body(req.body):
  {
    _id: <complaint_id>
  }
- Response body(res.body) if successful:
  {
    ...complaint
  }
- Response body(res.body) if failure:
  {
    error: ‘No such complaint’
  }

f) Create new complaint

- **Endpoint:** https://localhost:5000/api/complaints/
- **HTTP Method: POST**
- **Description:** Creates a new complaint
- Request body(req.body):
  {
    desc: “desc”
    priority: <number>
  }
- Response body(res.body) if successful:
  {
    ...complaint
  }
- Response body(res.body) if failure:

{
error: err.message
}

g) Delete complaint

- **Endpoint:** https://localhost:5000/api/complaints/:id
- **HTTP Method: DELETE**
- **Description:** Deletes complaint with the id.
- Request body(req.body):
  {
  _id: <complaint_id>
  }
- Response body(res.body) if successful:
  {
  ...complaint
  }
- Response body(res.body) if failure:
  {
  error: ‘No such complaint’
  }

h) Update complaint

- **Endpoint:** https://localhost:5000/api/complaints/:id
- **HTTP Method: PATCH**
- **Description:** Updates complaint with the id in this manner: A sent complaint updates to resolving, a resolving complaint updates to resolved.

- Request body(req.body):
  {
  _id: <complaint_id>
  }
- Response body(res.body) if successful:
  {
  ...complaint
  }
- Response body(res.body) if failure:
  {
  error: ‘No such complaint’
  }

DevOps

SCM:

Source Code Management (SCM) is a software tool used by programmers to manage the source codes and their versions. SCM is crucial when working on a project in teams and during iterative and incremental development.

Git/Github has been used to achieve SCM.We don't need seperate backend and frontend repositories as the build process integrates both and results in an integrated website.

Repo link: https://github.com/Bimg-Brein42069/housekeepers-devops

CI/CD Pipeline:

Jenkins is used as the CI/CD pipeline tool. First the backend is built, then tested using jest, then the frontend is built.
Pipeline script is stored in github:
Link: https://github.com/Bimg-Brein42069/housekeepers-devops/Jenkinsfile

Containerization:

Docker is used to store the project build files into a container. Jenkins uses docker and docker-compose to build the images out of the docker files.

docker-compose.yaml

```yaml
services:
  webserver:
    image: "dspani/housekeepserver"
    build: ./server
    ports:
      - "5000:5000"
  webclient:
    image: "dspani/housekeepclient"
    build: ./client
    ports:
      - "3000:3000"
    network_mode: host
    depends_on:
      - webserver
  redis:
    image: "redis:alpine"
```

Once the images are created, Jenkins pushes them into dockerhub repo and prunes the controller of unneccsary images to save space.
Dockerhub repo links:
https://hub.docker.com/repository/docker/dspani/housekeepserver
https://hub.docker.com/repository/docker/dspani/housekeepclient

Deployment:

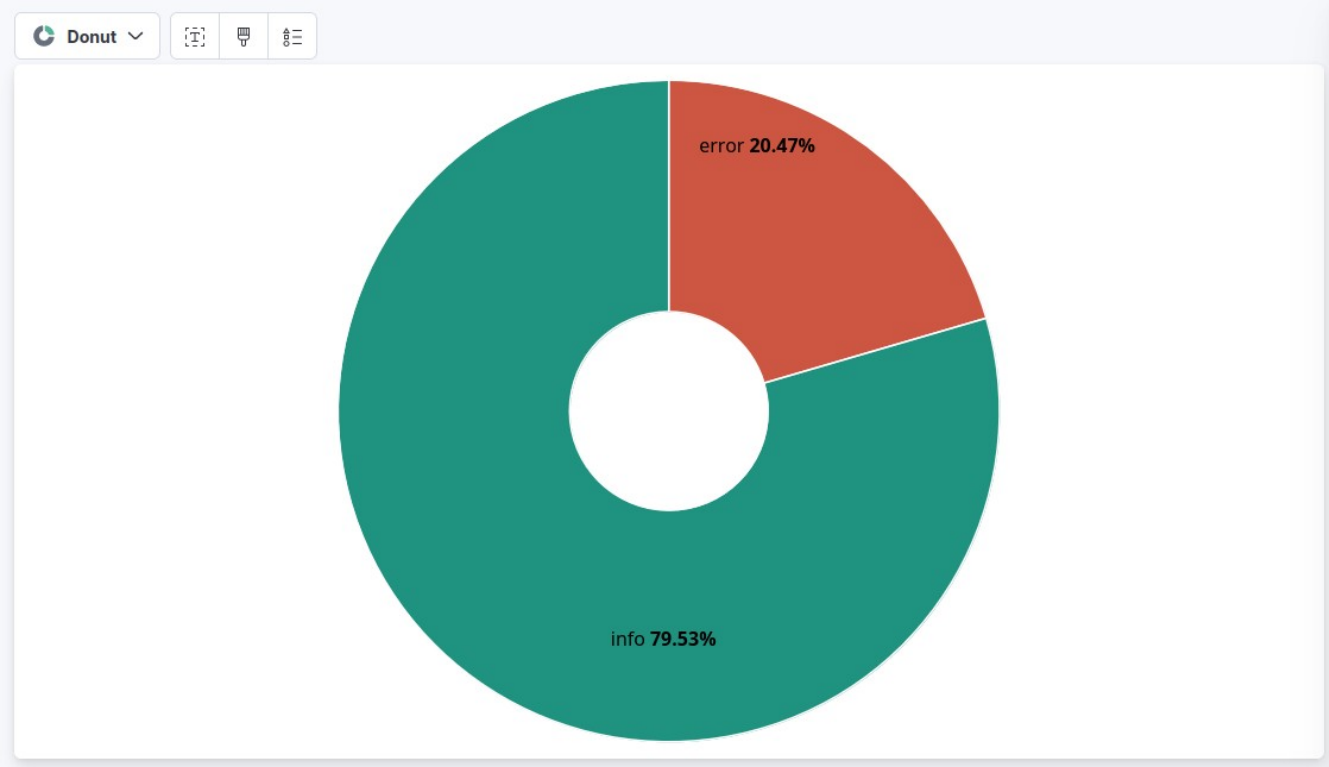
Ansible is used to deploy the docker images into the target systems. In this case, we deploy both to the localhost, then run them using docker compose (another file- runner.yaml) since both of them depend on each other, and docker does not allow localhost access by default.

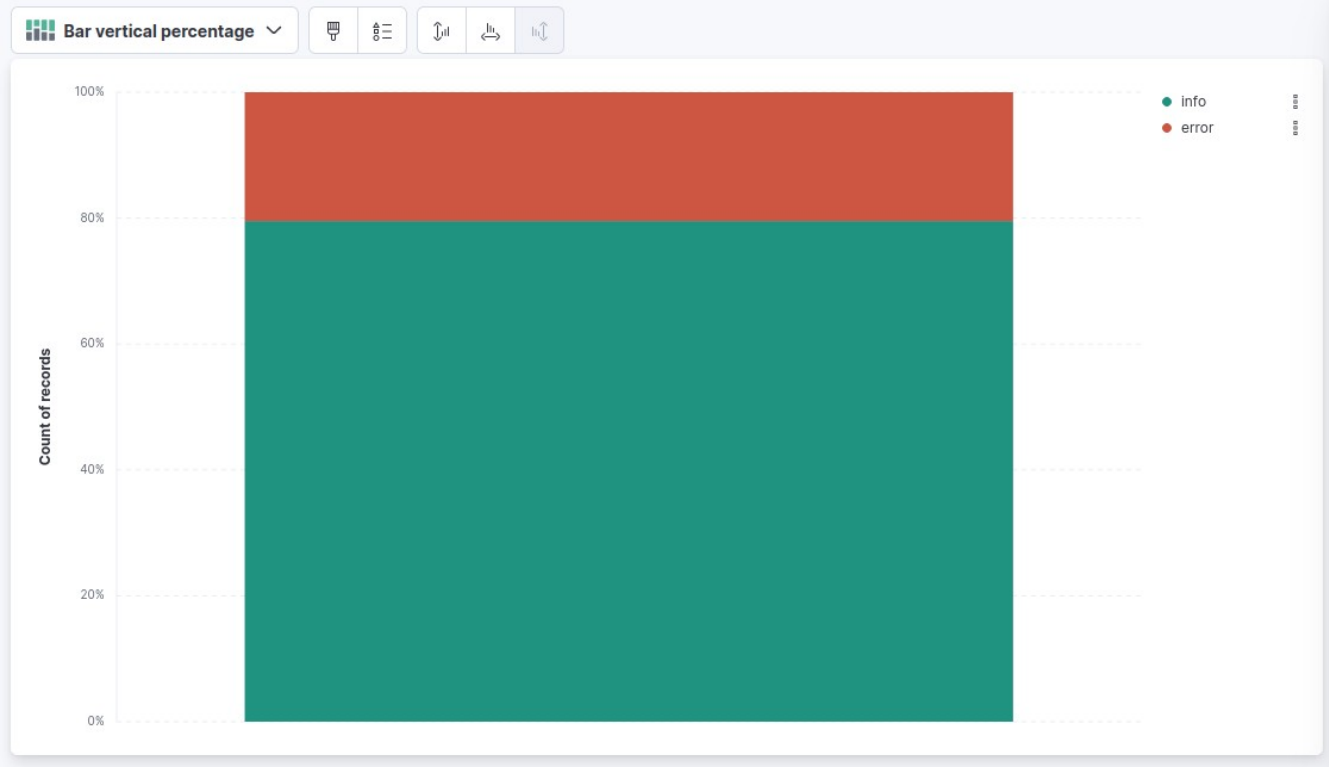
Monitoring:

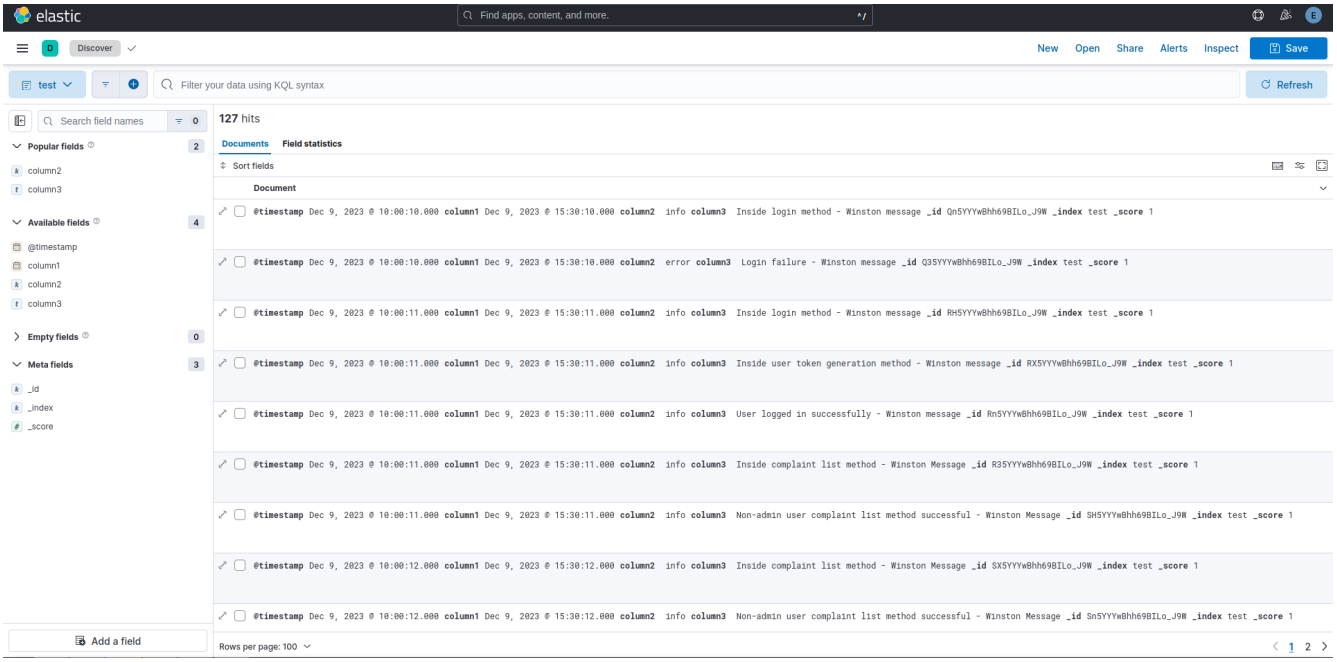
ELK stack is used to monitor the log files generated by the project.

Logstash will accept the log file as input and send the output to Elasticsearch, Kibana will then visualise the output stored in Elasticsearch.

Log Outputs:


Donut
error 20.47%
info 79.53%

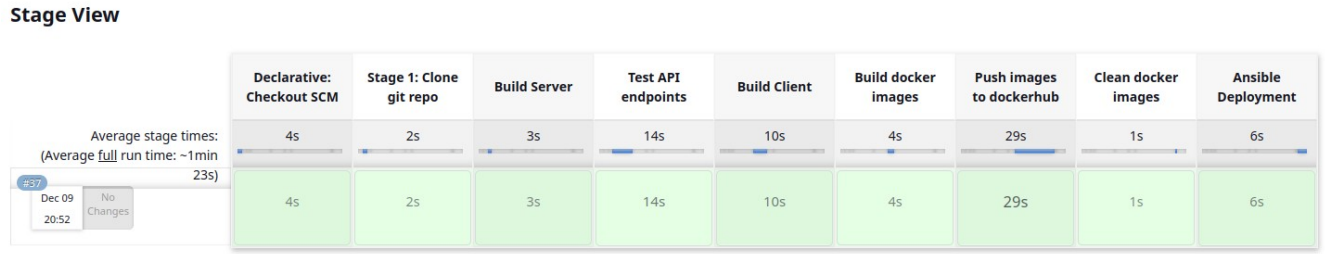

Bar vertical percentage
100%
80%
60%
40%
20%
0%
Count of records
info
error


elastic
Find apps, content, and more.
Discover
New
Open
Share
Alerts
Inspect
Save
test
Filter your data using KQL syntax
Refresh
Search field names
Popular fields
column2
column3
Available fields
@timestamp
column1
column2
column3
Empty fields
Meta fields
_id
_index
_score
Add a field
127 hits
Documents
Field statistics
Sort fields
Document
@timestamp Dec 9, 2023 @ 10:00:10.000 column1 Dec 9, 2023 @ 15:30:10.000 column2 info column3 Inside login method - Winston message _id Qn5YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:10.000 column1 Dec 9, 2023 @ 15:30:10.000 column2 error column3 Login failure - Winston message _id Q35YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:11.000 column1 Dec 9, 2023 @ 15:30:11.000 column2 info column3 Inside login method - Winston message _id RH5YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:11.000 column1 Dec 9, 2023 @ 15:30:11.000 column2 info column3 Inside user token generation method - Winston message _id RX5YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:11.000 column1 Dec 9, 2023 @ 15:30:11.000 column2 info column3 User logged in successfully - Winston message _id Rn5YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:11.000 column1 Dec 9, 2023 @ 15:30:11.000 column2 info column3 Inside complaint list method - Winston Message _id R35YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:11.000 column1 Dec 9, 2023 @ 15:30:11.000 column2 info column3 Non-admin user complaint list method successful - Winston Message _id SH5YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:12.000 column1 Dec 9, 2023 @ 15:30:12.000 column2 info column3 Inside complaint list method - Winston Message _id SX5YYYwBhh69BILo_J9W _index test _score 1
@timestamp Dec 9, 2023 @ 10:00:12.000 column1 Dec 9, 2023 @ 15:30:12.000 column2 info column3 Non-admin user complaint list method successful - Winston Message _id Sn5YYYwBhh69BILo_J9W _index test _score 1
Rows per page: 100
1
2

Results

In this project we were able to achieve Continuous Integration and Continuous Delivery and successfully apply all DevOps principles. Jenkins was used to trigger automatic builds, perform tests, build docker images and use Ansible to finally run the docker images.

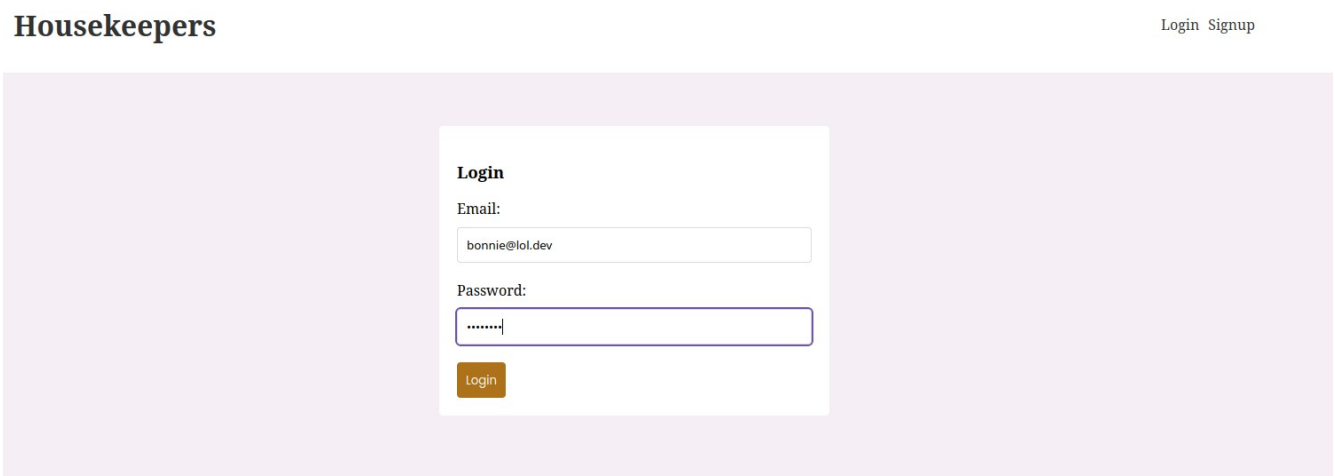
Jenkins pipeline script:



As you can see, it takes 1m23s on average to deploy the app. The majority of time taken is from testing, building the react client and pushing the images to dockerhub.

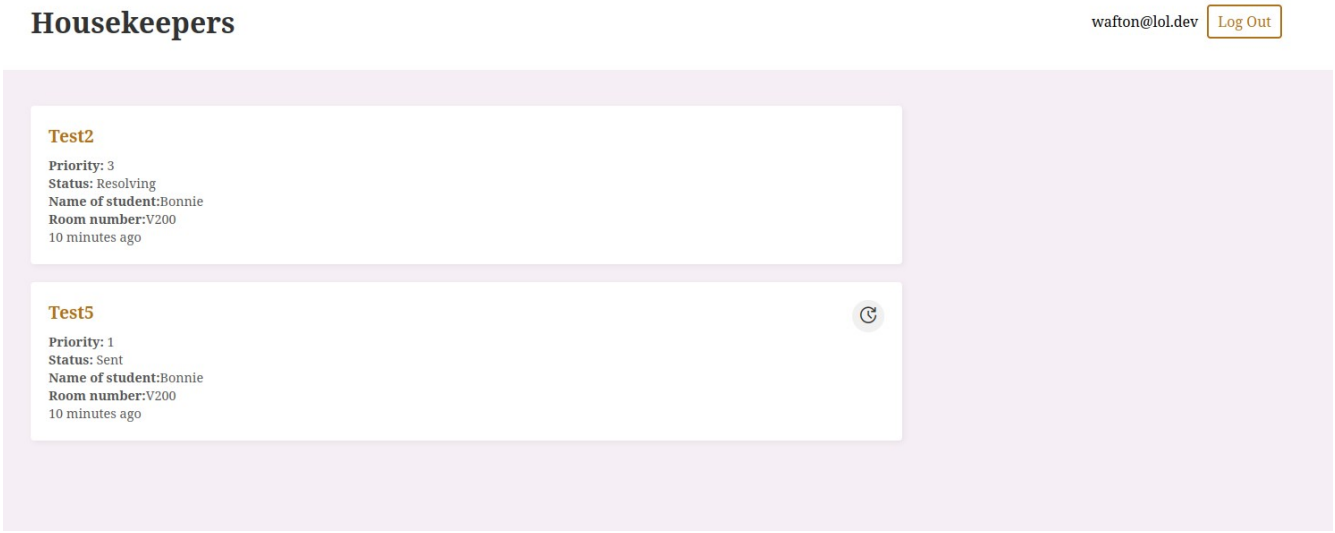
Screenshots of running the code:

User login:



Student dashboard:

Admin dashboard:



Future Scope:

The project can be enhanced with the following

- **NLP Model:** Students can abuse this system and create high urgency complaints everytime, the admin can only provide a limited number of resources, thus causing the students actually needing an urgent resolution unneccesary delays. Instead of allowing the student to set priority, we can create a NLP model to intake the description of the complaint, then use it to assign the appropiate prioirity to the complaint.
- **Dynamic Priority:** Suppose the student's low priority complaint is unresolved due to higher priority complaints, and the student needs it to be resolved suddenly, then the student gets a option to change the priority after a day.
- **Notifier**: A notifier can be configured such that when a priority 3 complaint is created, the system will immediately notify the admins of the complaint with a high priority one. Priority 2 complaints will be notified after a short amount of time(~5 mins) and the admin can choose whether he wants a priority 1 complaint to be notified or not.

Challenges faced:

- Jest has a timeout problem wherein if a test exceeds 5 seconds before completion, then it gets timed out. Such problems occurred in the project when testing POST requests. One successful workaround around that was to increase the timer to 30 seconds in the config. This problem is caused due to an unfixed regression since jest 27 https://github.com/jestjs/jest/issues/11500

```
FAIL ./server.test.js (9.283 s)
  post request /signup
    ✓ status code must be 400, type must be json list (57 ms)
    × status code must be 400, type must be json list (5001 ms)
  post request /login
    ✓ status code must be 400, type must be json list (2143 ms)
    ✓ status code must be 200, type must be json list (508 ms)
  post request /signupadmin
    ✓ status code must be 400, type must be json list (10 ms)
    ✓ status code must be 400, type must be json list (469 ms)

  ● post request /signup › status code must be 400, type must be json list

    thrown: "Exceeded timeout of 5000 ms for a test.
    Add a timeout value to this test to increase the timeout, if this is a long-running test. See https://jestjs.io/docs/api#testname-fn-timeout."

      23 |         expect(response.type).toBe("application/json")
      24 |     })*/
    > 25 |     test("status code must be 400, type must be json list", async () => {
         |     ^
      26 |         const response=await request(userr).post("/api/user/signup").send({
      27 |             name: "testestest",
      28 |             email: "testest@lol.dev",

      at test (server.test.js:25:5)
      at Object.describe (server.test.js:4:1)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 5 passed, 6 total
Snapshots:   0 total
Time:        9.325 s
Ran all test suites.
Force exiting Jest: Have you considered using `--detectOpenHandles` to detect async operations that kept running after all tests finished?
[Pipeline] }
```

- Since docker files have their own localhost, starting both server and client docker files separately caused the client to be unable to connect to the server. Thus docker-compose was needed to be able to link the two.

docker-runner.yaml

```yaml
services:
  webserver:
    image: "dspani/housekeepserver"
    ports:
      - "5000:5000"
  webclient:
    image: "dspani/housekeepclient"
    ports:
      - "3000:3000"
    network_mode: host
    depends_on:
      - webserver
```

- Ansible failed on running, returning an unreachable host. Turns out, ansible does not support ssh authentication by password natively. To

authenticate passwords, sshpass was required on managed hosts, and we needed to add correct credentials into jenkins.

```
[housekeeper-pipeline] $ sshpass ******** ansible-playbook ./ansible-playbook.yml -i ./inventoryFile -u dspani -k

PLAY [Pull and Run docker image] ***********************************************

TASK [Gathering Facts] *********************************************************
[1;31mfatal: [127.0.0.1]: UNREACHABLE! => {"changed": false, "msg": "Invalid/incorrect password: Permission denied, please try again.", "unreachable": true}[0m

PLAY RECAP *********************************************************************
[0;31m127.0.0.1[0m                 : ok=0    changed=0    [1;31munreachable=1   [0m failed=0    skipped=0    rescued=0    ignored=0

FATAL: command execution failed
hudson.AbortException: Ansible playbook execution failed
        at org.jenkinsci.plugins.ansible.AnsiblePlaybookBuilder.perform(AnsiblePlaybookBuilder.java:271)
        at org.jenkinsci.plugins.ansible.workflow.AnsiblePlaybookStep$AnsiblePlaybookExecution.run(AnsiblePlaybookStep.java:468)
        at org.jenkinsci.plugins.ansible.workflow.AnsiblePlaybookStep$AnsiblePlaybookExecution.run(AnsiblePlaybookStep.java:362)
        at org.jenkinsci.plugins.workflow.steps.AbstractSynchronousNonBlockingStepExecution$1$1.call(AbstractSynchronousNonBlockingStepExecution.java:47)
        at hudson.security.ACL.impersonate2(ACL.java:451)
        at hudson.security.ACL.impersonate(ACL.java:463)
        at org.jenkinsci.plugins.workflow.steps.AbstractSynchronousNonBlockingStepExecution$1.run(AbstractSynchronousNonBlockingStepExecution.java:44)
        at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:539)
        at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
        at java.base/java.lang.Thread.run(Thread.java:840)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: Ansible playbook execution failed
Finished: FAILURE
```

Conclusion:

Ultimately we were able to completely go through the Automated SDLC of a web application “Housekeepers”.
We learned and worked with React framework, NodeJS libraries like Express for backend and Mongoose for MongoDB database, SCM like GitHub, CI/CD tools like Jenkins and Docker, Deployment tools like Ansible, and monitoring tools like ELK.
We were able to appreciate the importance of automated tools. It really helped us to simplify our work.