

# **SOFTWARE ENGINEERING**

## **PROJECT REPORT**

---

**Project Title**  
**Online LHC room booking system.**

**Team Members**  
Manav Gopal(B19CSE112)  
Kartik Narayan(B19CSE110)

**Document Type**  
Test Case Design and Automation Script



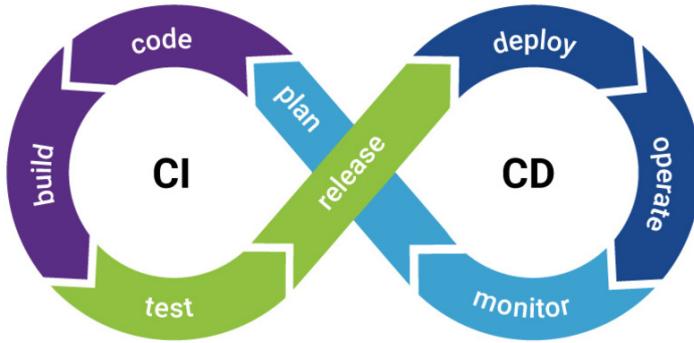
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
IIT JODHPUR**

## AUTOMATION SCRIPT

For the automation script we had decided to run a CI/CD pipeline which fully automates all the processes.

### **What is CI/CD pipeline and how we used it.**

In software engineering, CI/CD or CICD generally refers to the combined practices of continuous integration and either continuous delivery or continuous deployment. CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications.



## PIPELINE BUILD

Chosen Software - LHC Room Booking System

Github Link - [SE Project](#)

We had used two automation script methods to test our software which are:

- Fake data generation
- Github actions

### **Method 1 - Generating fake data to test the software**

We have written a script in javascript which was used in the starting phase of development to run the software. The data generated was used to display the rooms along with the capacity. The room numbers were labelled numerically and randomly. The capacity of the room was also assigned randomly from within the range 0 to 8. We named

3 random locations which were allocated randomly to the rooms. The script generated a json file which contains a javascript object with all the information about the rooms.

```
// Randomly generate a fake allTables JSON file

const fs = require('fs');
const numRooms = Math.floor(Math.random() * 10) + 16; // 16 - 26 (exclusive)

const fakeRooms = [];
for (i = 1; i < numRooms; i++) {
    const chairs = Math.floor(Math.random() * 6) + 2; // 2-8 (exclusive)
    const name = `Room ${i}`;
    // const availability = [true, false][Math.round(Math.random())];
    const location = ['Patio', 'Inside', 'Bar'][Math.floor(Math.random() * 3)]; // 0-3
    (exclusive)
    fakeRooms.push({
        name: name,
        capacity: chairs,
        // isAvailable: availability,
        isAvailable: true,
        location: location,
    });
}

const data = JSON.stringify({
    rooms: fakeRooms,
});
fs.writeFileSync(__dirname + '/allRooms.json', data);
```

## Method 2 - CI/CD pipeline using github actions

CI-CD pipeline:

- Commit-New coders are integrated to the base code.
- Build-Source code is converted into executable form.
- Test-Check the interaction between builds and if the app is working or not.
- Deploy-Deploys the application to the production environment.

### Steps

## Step 1-

Clone the project from GitHub which is used to make the CI/CD pipeline.

`git clone https://github.com/Kartik-3004/SE_Project.git`

The screenshot shows the GitHub repository page for 'Kartik-3004/SE\_Project'. The repository has 1 branch and 0 tags. The code tab is selected, showing a list of files and their last commit times. The 'Clone' button is highlighted, providing links via HTTPS, SSH, or GitHub CLI. The repository has no description, website, or topics provided. It includes a Readme, MIT License, and no releases published. Contributors listed are Kartik-3004 and manavgopal. The environments section shows 'se-project-heroku' as active. The languages section indicates a mix of JavaScript and Node.js.

Setup the project in your local environment and add your origin repository.

`git remote add origin https://github.com/Kartik-3004/SE_Project.git`

## Step 2 -

Open the project in your local computer in an editor Now create a .yml file.

`mkdir .github/`

`mkdir .github/workflows/`

`touch .github/workflows/node.js.yml`

## Step 3 -

Write the code of the pipeline

```
# This workflow will do a clean install of node dependencies, build the source code  
and run tests across different versions of node
```

```
# For more information see:  
https://help.github.com/actions/language-and-framework-guides/using-nodejs-with-github  
-actions  
  
name: Node.js CI  
  
on:  
  push:  
    branches: [ main ]  
  pull_request:  
    branches: [ main ]  
  
jobs:  
  build:  
  
    runs-on: ubuntu-latest  
  
    strategy:  
      matrix:  
        node-version: [10.x, 12.x, 14.x, 15.x]  
        # See supported Node.js release schedule at  
        https://nodejs.org/en/about/releases/  
  
      steps:  
        - uses: actions/checkout@v2  
        - name: Use Node.js ${matrix.node-version}  
          uses: actions/setup-node@v2  
          with:  
            node-version: ${matrix.node-version}  
        - run: npm ci  
        - run: npm run build --if-present  
        - run: npm test
```

## Step 4 -

Create an app on heroku for deployment

Add environment variables, HEROKU\_API\_KEY and HEROKU\_APP\_NAME to the project and set up the CD.

git push heroku master

The screenshot shows the Heroku dashboard for the application 'se-project-heroku'. The dashboard includes sections for Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. It displays information about installed add-ons (\$0.00/month), dyno formation (\$0.00/month), and collaborator activity (narayan.2@iitj.ac.in). The 'Latest activity' section lists several build logs, with the first entry showing a successful deployment on May 15 at 3:36 PM.

Activity	Date	Status
narayan.2@iitj.ac.in: Deployed a6603ac7	May 15	Succeeded
narayan.2@iitj.ac.in: Build succeeded	May 15	Succeeded
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed
narayan.2@iitj.ac.in: Build failed	May 15	Failed

The screenshot shows “build succeed” which shows a CD made is working.

## Step 6 -

All the changes are added and committed.

git add .

git commit -m “<name>” git push origin -u master

## Step 7 -

Github provides features like “Github actions”. Use that to make your CI/CD pipeline. Go to the action sections of the GitHub repository.

The screenshot shows the GitHub Workflows interface. At the top, there's a blue header bar with the text "Workflows" and a "New workflow" button. Below this, a blue navigation bar highlights "All workflows". A sidebar on the left shows a "Node.js CI" icon with the number "0". The main area is titled "All workflows" and says "Showing runs from all workflows". It includes a search bar labeled "Filter workflow runs". A table lists one workflow run:

1 workflow run	Event ▾	Status ▾	Branch ▾	Actor ▾
<b>Update node.js.yml</b> Node.js CI #8: Commit f902510 pushed by Kartik-3004	master	25 days ago	2m 11s	...

## Step 8 -

Build pipeline can be seen on clicking the jobs and if there is tick on every stage of the pipeline there we have obtained our pipeline.

The screenshot shows the GitHub Actions build log for the job "build (15.x)". The log is displayed in a dark-themed interface. On the left, a sidebar lists other build jobs: "build (10.x)", "build (12.x)", "build (14.x)", and "build (15.x)", with "build (15.x)" currently selected. The main pane shows the build steps:

- Set up job (7s)
- Run actions/checkout@v2 (6s)
- Use Node.js 15.x (4s)
- Run npm ci (43s)
- Run npm install (5s)
- Run CI=false npm run build --if-present (45s)
- Run npm test (5s)
- Post Run actions/checkout@v2 (0s)
- Complete job (0s)

At the bottom right of the log pane, there is a "Search logs" input field and a gear icon.

## TEST CASE DESIGN

**Software Testing** is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

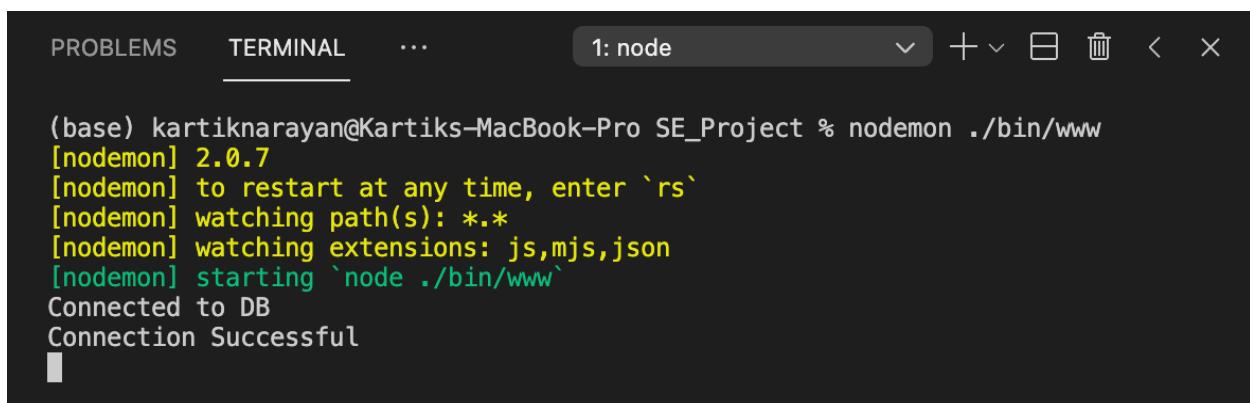
We are testing our software in four different areas. These include backend requests, React frontend, Functional testing, Database connection. We have also integrated the software with CI/CD which tests our software after integration of each functionality. It provides information about any merge conflicts and code errors which disturbs the code base

### 1. Backend Requests

We have used Node.js along with express.js for our backend. In this testing criteria we are going to test GET and POST requests to the server. We have used Postman to test these which interprets the requests and presents in a very user friendly way. **Postman** is a popular API client that makes it easy for developers to create, share, test and document APIs. This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses.

#### Steps

Run the backend server and connect it to the database



```
(base) kartiknarayan@Kartiks-MacBook-Pro SE_Project % nodemon ./bin/www
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/www`
Connected to DB
Connection Successful
```

Write a simple code and make a post request to send some data.

1) Checking availability route

```
const express = require('express');
const router = express.Router();

// Parameters:
// {
//   "date": String ("Apr 30 2021 06:00")
// }

router.post('/', function (req, res, next) {
  res.status(200).send('Availability Route working');
});

module.exports = router;
```

2) Checking reservation route

```
const express = require('express');
const router = express.Router();

// Parameters:
// {
//   "date": String ("Apr 30 2021 06:00")
// }

router.post('/', function (req, res, next) {
  res.status(200).send(['Reservation| Route working']);
});

module.exports = router;
```

We first need to open the postman application and type the route where the data needs to be altered and set the type of request to POST.

We then use Postman to check whether the get and post request is working or not.

The screenshot shows the Postman application window. The left sidebar lists 'My Workspace' with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. A central message says 'You don't have any collections' with a 'Create Collection' button. The main workspace shows a POST request to 'http://localhost:1853/availability'. The 'Params' tab is selected, showing a single parameter 'Key' with value 'Value'. Below the request details, the 'Body' tab is selected, showing the response: '1 Availability Route working'. The status bar at the bottom indicates '200 OK 31 ms 286 B'.

We then send the POST request and check the test results.

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area displays a POST request to `http://localhost:1853/availability`. The 'Params' tab is selected, showing a single query parameter 'Key' with value 'Value'. Below the request, the response status is 200 OK, 31ms, 286 B. The response body contains the text 'Availability Route working'. At the bottom, there are tabs for Body, Cookies, Headers (8), and Test Results, along with a 'Pretty' button.

The same is to be done for the reservation route.

We fill in the reservation route and select the request type to be POST

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area has a title bar "Postman" with tabs for Home, Workspaces, Reports, and Explore. Below the title bar, there's a search bar and some global settings like "Invite", "Upgrade", and "Save". The main workspace is titled "My Workspace" and shows a message "You don't have any collections". In the center, there's a POST request setup for "http://localhost:1853/reserve". The "Params" tab is selected. The "Query Params" table has one row with "Key" and "Value" columns. A large button labeled "Send" is visible. Below the request area, there's a "Response" section with a placeholder message "Hit Send to get a response". At the bottom, there are buttons for "Find and Replace", "Console", and links to Bootcamp, Runner, Trash, and Help.

We then send the POST request and check the test results.

This screenshot shows the same Postman interface after the POST request has been sent. The "Body" tab is now active in the response pane. It displays the text "Reservation is working" in a monospaced font. Above the body, the response status is shown as "200 OK 19 ms 282 B". There are also "Save Response" and "Copy" buttons. The rest of the interface remains the same, including the sidebar and the "Send" button.

## Conclusion

We can see that the GET and POST requests are working and the server health is fine and there are no bugs. The availability and reservation route both are in use by the server and both are working.

## 2. React Frontend

We are going to use the inbuilt "`react-scripts test`" provided by React.js to check the frontend of the application. The script will check the react buttons, routing and ensure the correct development practices are used.

```
(base) kartiknarayan@Kartiks-MacBook-Pro SE_Project % npm test
```

## Output

```
PASS  src/App.test.js
  ✓ renders learn react link (37 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.589 s
Ran all test suites related to changed files.

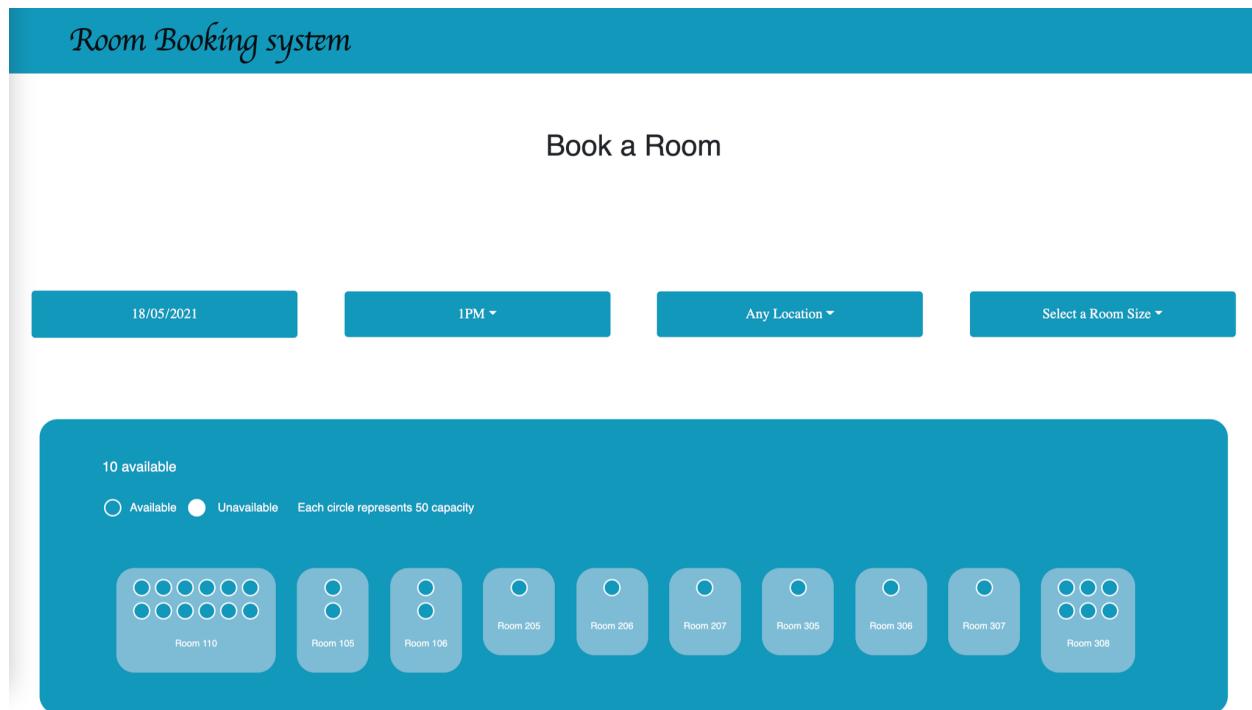
Watch Usage
  > Press a to run all tests.
  > Press f to run only failed tests.
  > Press q to quit watch mode.
  > Press p to filter by a filename regex pattern.
  > Press t to filter by a test name regex pattern.
  > Press Enter to trigger a test run.
```

## Conclusion

Our application has passed the frontend test using the inbuilt function and there are no errors in the frontend of the application.

### 3. Functionality Testing

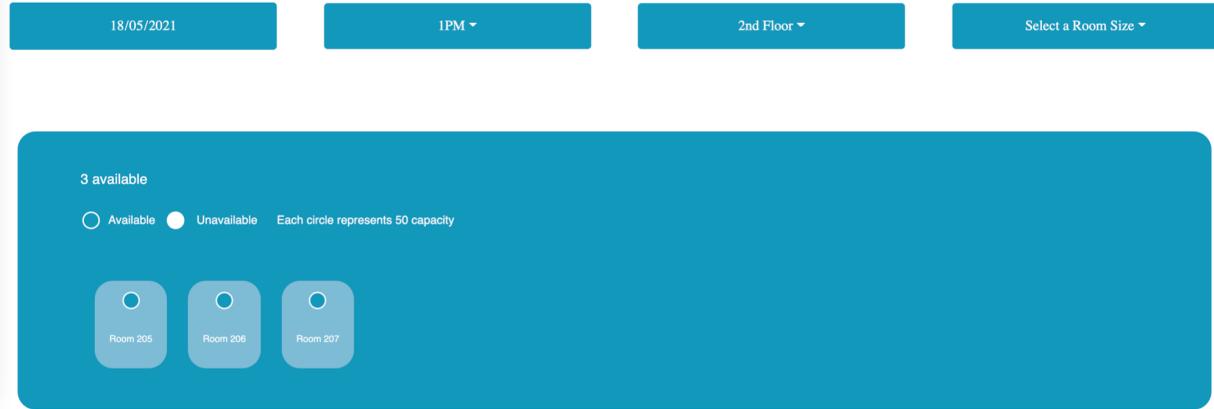
- 1) Filter of rooms based on location and parameters.



These are the available rooms without any filter applied.

## Room Booking system

### Book a Room



Here we can see the availability of rooms after applying the filter of location where we selected it to be 2nd floor.

- 2) Same room cannot be booked twice on the same date and time.

Steps -

Go to the main page of the application and press on book a room

## *Room Booking system*

If you're looking to book a room

[Book a Room](#)



Fill in the details and select the available room.

## *Room Booking system*

[Book a Room](#)

18/05/2021

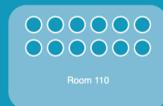
1PM ▾

Any Location ▾

Select a Room Size ▾

10 available

Available    Unavailable   Each circle represents 50 capacity



For instance, we can choose Room. 110 and book it for 18/05/2021 at 1PM.  
After selecting the room we need to fill in the details

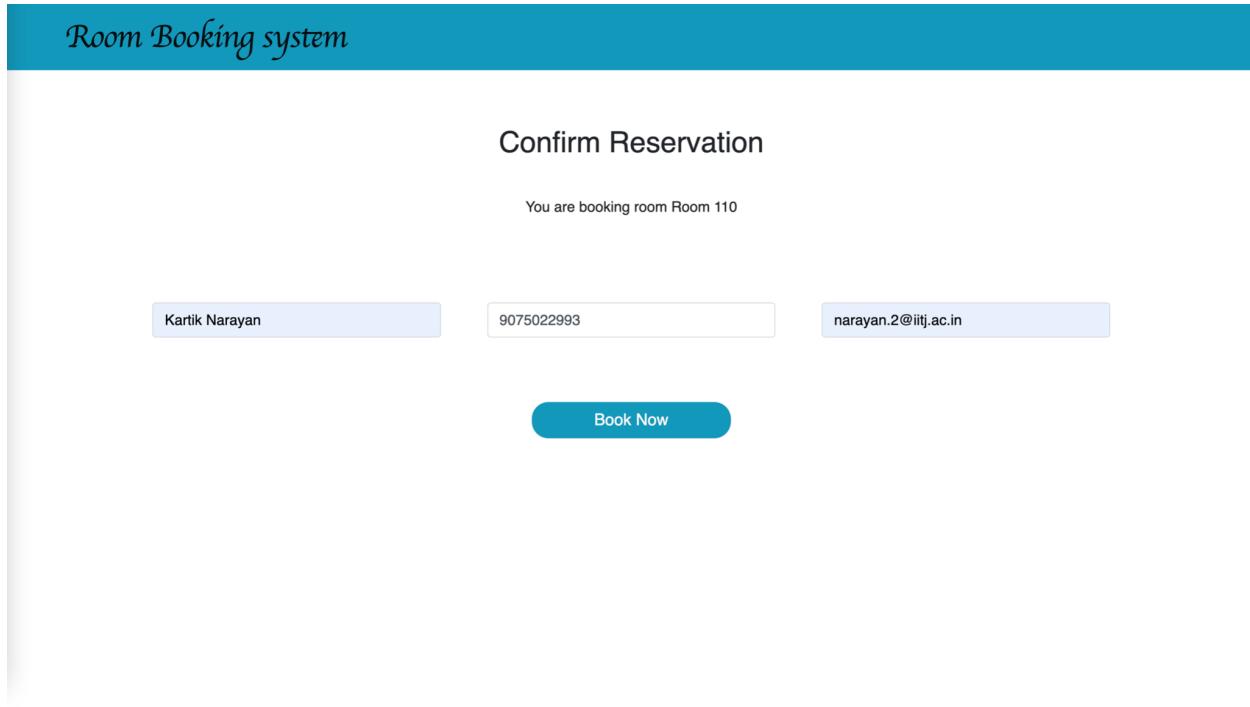
*Room Booking system*

Confirm Reservation

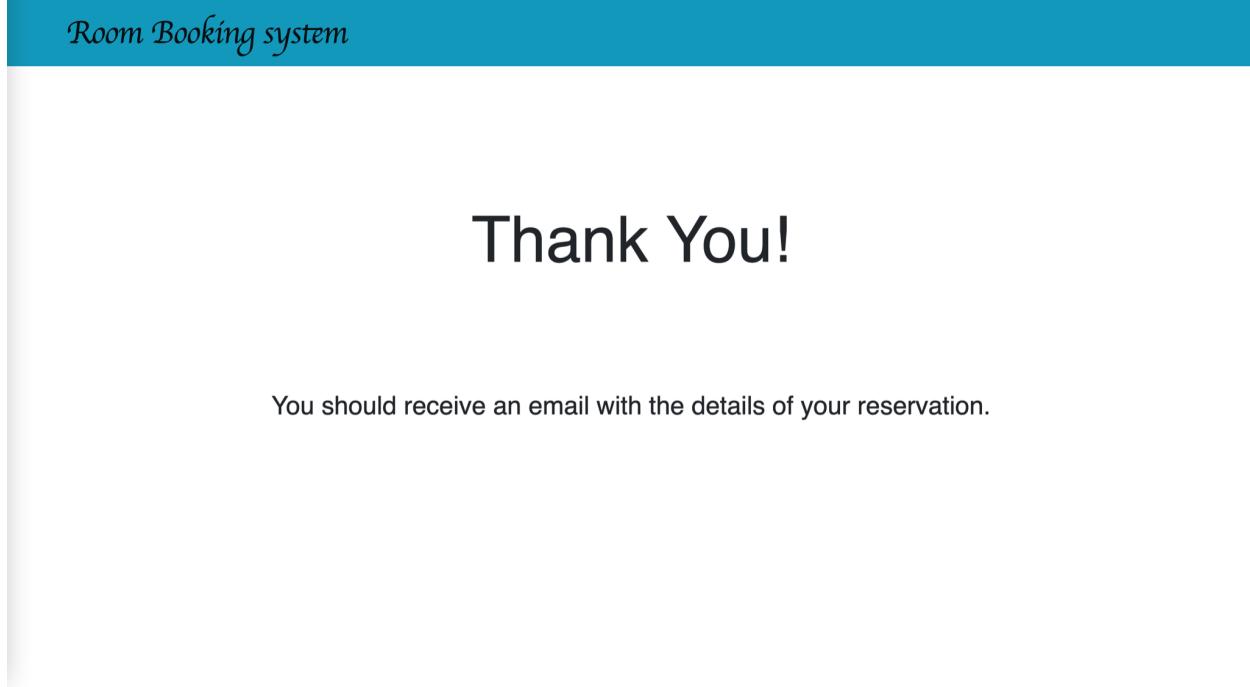
You are booking room Room 110

Kartik Narayan      9075022993      narayan.2@iitj.ac.in

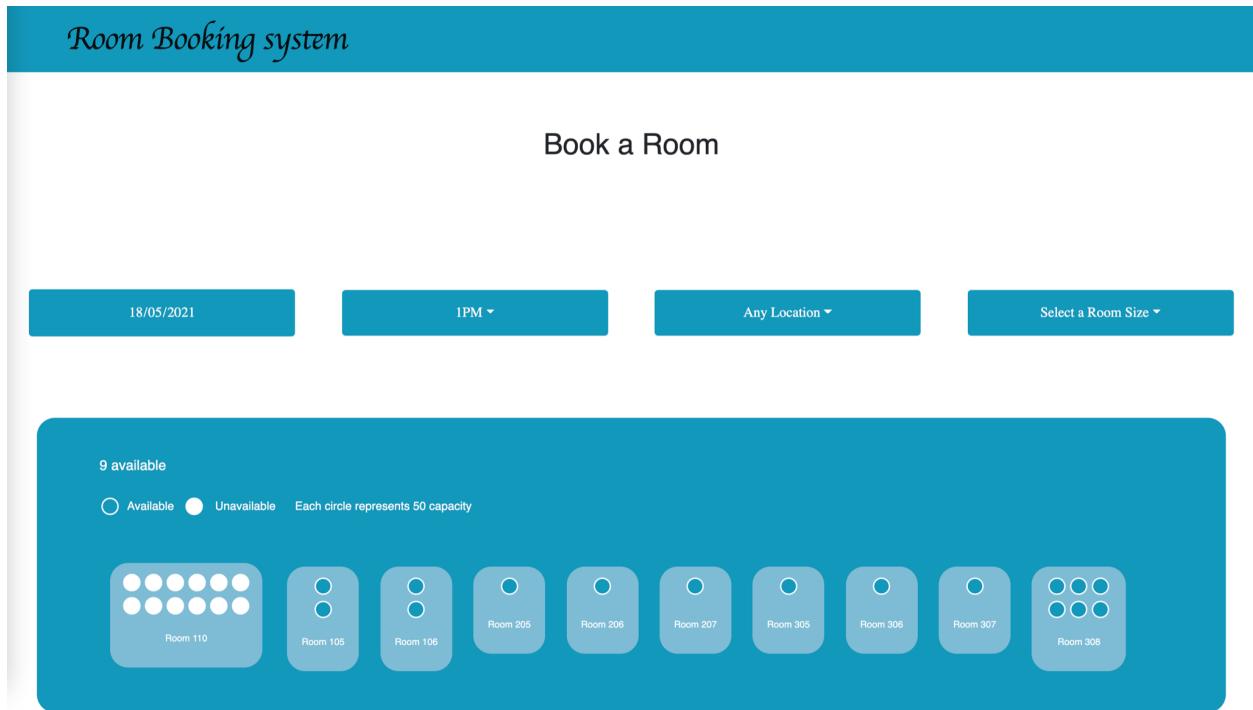
**Book Now**



You can then press the book now button and book the room for the filled in date and time.



Now to check whether the application is working we will again open the application and see whether the already booked room is available or it shows unavailable and you cannot book it.



We can see that the already booked room for the same date and time cannot be booked again. The same is shown in the figure above where the unavailable room is displayed by white circles.

#### Conclusion-

We have tested the room booking functionality of our software and tested some of the features like filtering the room based on parameters like location and capacity. We also checked that the same room cannot be booked twice for the same time

## 4. Database Testing

In the previous testing scenario where we tested the functionality of the software we filled in some details and booked the room. We will now see whether the data has been uploaded to the database or not.

### Steps

Open the database in an interactive UI and go to the required cluster where the data is being stored.

The screenshot shows the MongoDB Atlas interface. The top navigation bar includes 'Kartik's Org', 'Access Manager', 'Billing', 'All Clusters', 'Get Help', and 'Kartik'. Below the navigation is a secondary header with 'SE\_project', 'Atlas', 'Realm', and 'Charts'. The main area is titled 'DATA STORAGE' with 'Clusters' selected. It shows 'DATABSES: 1' and 'COLLECTIONS: 1'. A sub-section for 'database.days' is displayed, showing 'COLLECTION SIZE: 36.17KB', 'TOTAL DOCUMENTS: 33', and 'INDEXES TOTAL SIZE: 36KB'. A 'Find' button is present. The 'QUERY RESULTS 1-20 OF MANY' section lists four documents:

- `_id: ObjectId("609e1a9b59f6ac07e498959")  
date: 2021-05-17T04:30:00.000+00:00  
> rooms: Array  
__v: 0`
- `_id: ObjectId("609e1c05af1ec13a3a499")  
date: 2021-05-15T19:30:00.000+00:00  
> rooms: Array  
__v: 0`
- `_id: ObjectId("609e1d34ab401ac157ac1360")  
date: 2021-05-19T06:30:00.000+00:00`

Open the stored data object and check the availability of the rooms.

```
_id: ObjectId("60a21028068fbc0cfa13e832")
date: 2021-05-17T19:30:00.000+00:00
rooms: Array
  0: Object
    _id: ObjectId("60a21028068fbc0cfa13e828")
    name: "Room 110"
    capacity: 12
    isAvailable: false
    location: "1st Floor"
    > reservation: Object
  1: Object
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object
  8: Object
  9: Object
__v: 0
```

From the image above we can cross check that the booking status of the Room 110 which we booked has been updated and the availability of the room with the time and date is updated in the database.

## Conclusion

The connections to the database are healthy and there are no leaks. The booking details of every booking is updated in the database along with other details of the booking like location and capacity of the room.

---

**Thanking You**  
**Course instructor - DR. Sumit Kalra**