



Department of Computer Application

University of Petroleum & Energy Studies, Dehradun

MINOR-I PROJECT

REPORT

ON

Submitted By

Rahul Sharma	Rajat Tak	Kunal Gaurav Srivastava	Vasu Tayal
500066535	500070511	500068583	500068847

Under the guidance of

DR. Vinay Awasthi

INDEX

● Acknowledgement.....	4
● Abstract.....	5
● Introduction.....	6
● Problem Statements.....	7
● Solutions to problem statements.....	8
● Literature Review.....	9
● Objectives.....	10
● Methodology.....	11
● System Requirements(Hardware/Software).....	12
● Schedule(Pert Chart).....	13
● ER Diagram.....	14
● Data Analysis.....	15
● Database.....	17
● Conclusions.....	19
● Screenshot.....	20

ACKNOWLEDMENT

I would like to express my special thanks of gratitude to my teacher (DR VINAY AWASTHI) as well as our course coordinator (SONALI MA'AM) who gave me the golden opportunity to do this wonderful project on the topic "HOSTEL MANAGEMENT SYSTEM", working on this project has helped us to put our ideas to more realistic and challenging. We actually pushed us to the point where we implemented the problems which we faced and actually come up with something that solved the issues.

Secondly we would also like to thank our parents and friends who helped us a lot in finalizing this project within the limited time frame. At last we want to thank our Project reviewer Dr.Kaushik Ghosh who gave us real time solutions and made our project much better.

Abstract

Necessity is the mother of invention, we at our website we tend to make the user experience for our users and students more interactive and lay down more information to them. Here we compare different hostels depending on the type of service they provide so that students can find home according to their choice. We can also make the payment done for the hostel and grab our seats for it at earliest. We also bring a featured website which raises ticket for students facing problem in their respective rooms with day out and night out pass facility. This website is something students were looking forward from a long time, it serves as one place all solution. It also helps the warden or the hostel management to have hassle free service and information about the students where about, it promotes paper free work which tends to promote the SDG and new India.

Introduction

As student face a lot of difficulty in finding a true place for themselves, so thought for it and tend to make it free for the students, respective we will charge the hostels for getting featured on our website, as students need a platform for finding place of their requirement. Looking at the student and hostel requirement we brought things that can pile up the entire work and comes as a solution for all. As the demand for this kind of work is high but there is no platform for the same, so we insisted on solving a problem that was there from before and that can really help the students of UPES. At first we try our beta project with the areas around Bidholi, Kandoli, Pondha, Nanda Ki Chowki and Dunga and then we go live with Dehradun and boundaries ahead.

Problem Statement:

Provide hassle free platform for the students and help them finding their homes and a well featured platform for hostel management.

Solutions to problem statement

- We made a platform where students can and compare the listed hostels in the area.
- the management system enables the student to login and report their problems as well as their movement in / out of the hostel campus
- the management system enables the warden to keep track of student movement as well as his / her whereabouts
- there is a portal for guard and all the people associated to the work that makes it hassle free
- by doing this we solve the paper free and provide student and hostel a user friendly experience which helps them in daily hostel life

Literature Review

We as students faced a lot of difficulties while finding a place for ourselves and when we come to the campus the first thing is the hostel and hence the campus hostel cannot provide accommodation for all so we have to get us a desired place to live , so looking out on that issue we came up with this idea of hostel management system which not only provide us to find the hostel of our choice but also helps us in providing a platform for the hostel so that they can have a hassle free work system . as day out and night out pass is required , so we focus on email and message generated work which saves paper but also keeps us in loop with the hostel on current basis. Students face problems in their rooms so they can raise a ticket on the website and get the hostel notified about their problem and get things done at real time basis, while this shortens the time and energy. So we actually focused on the requirement of a student and tried to work for their help.

Objective :

- ❖ Website to find hostel of their of own choice
- ❖ Well established hostel management system
- ❖ Paper free work

Methodology:

In this project we are using, **Feature Driven Development (FDD)**, methodology as our project is full of features that will provide a unique taste for our customers.

It's an iterative and incremental approach for development. It's an older methodology but it still has a lot to deliver. Features are a foundational piece of FDD and our whole project revolves around a bunch of user friendly features for betterment of the hostels around our campus, UPES.

Our portal would have a different view along with different authority for its users.

Example:

- Student
- Security officer
- Warden
- Developer and many more.

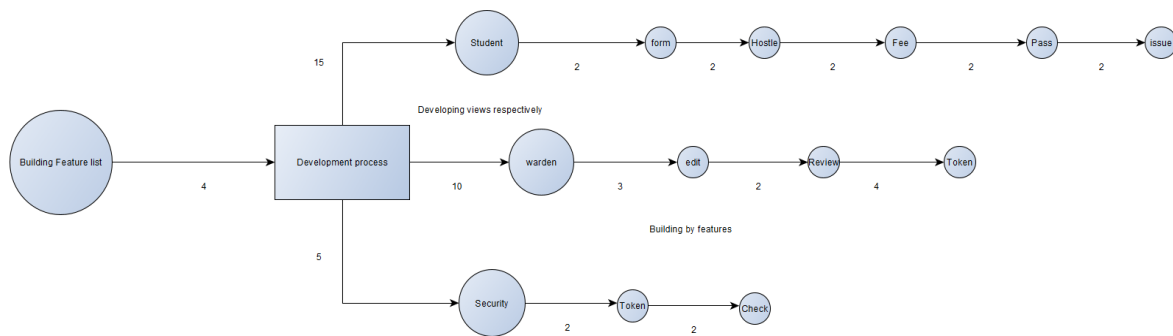
System Requirements: (Software/Hardware)

- ❖ Good internet connection.
- ❖ Web browser.
- ❖ Smartphone/ computer to access the internet.

Schedule: (PERT Chart)

Program Evaluation Review Technique is a project management tool that provides a graphical representation of a project's timeline.

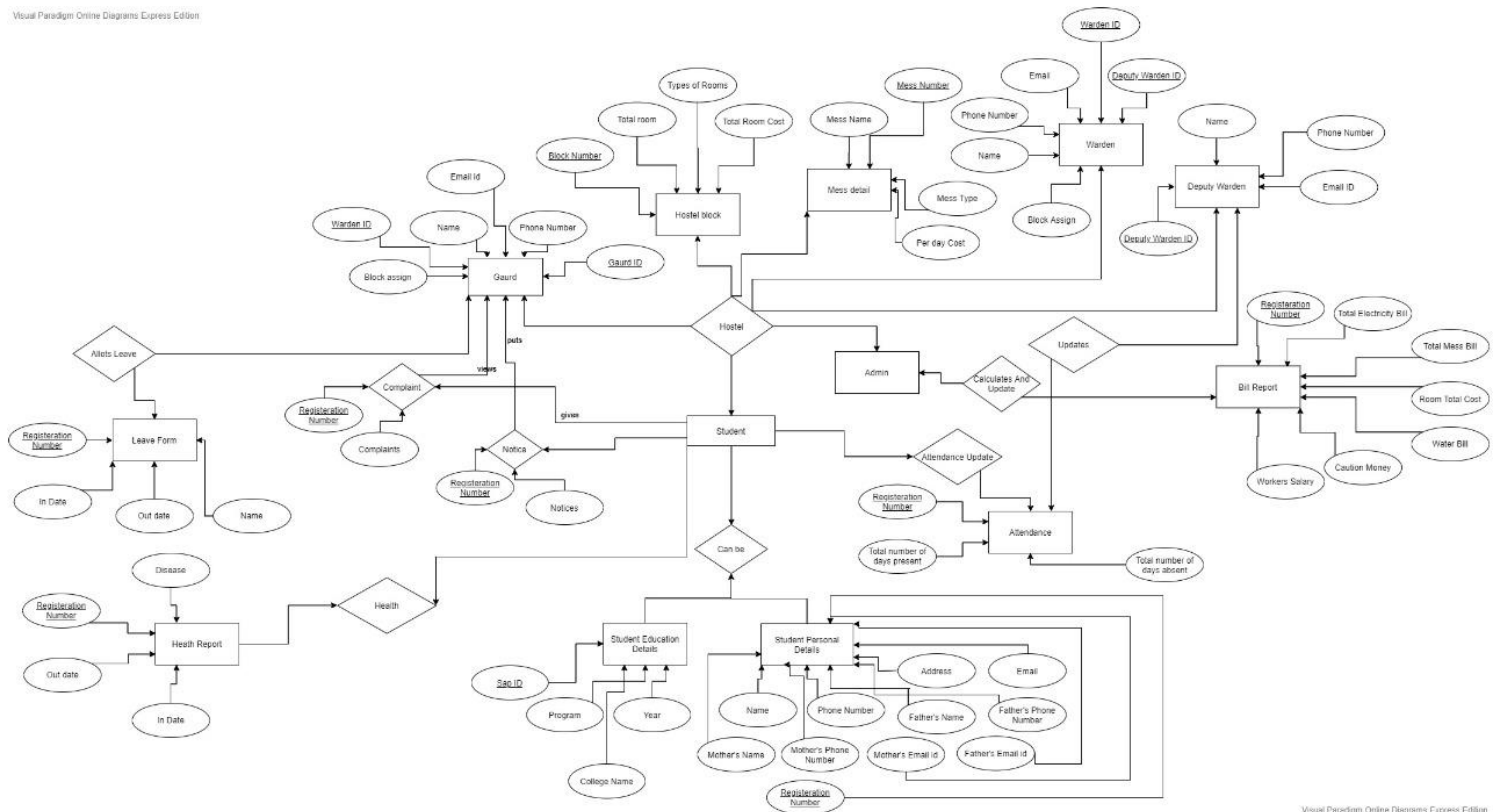
We have laid down the milestones necessary to complete this project timely.



The figure below provides the activity and the time required to complete this project.

* Based on the methodology this chart has been created. High priority has been given to our features.

ER DIAGRAM

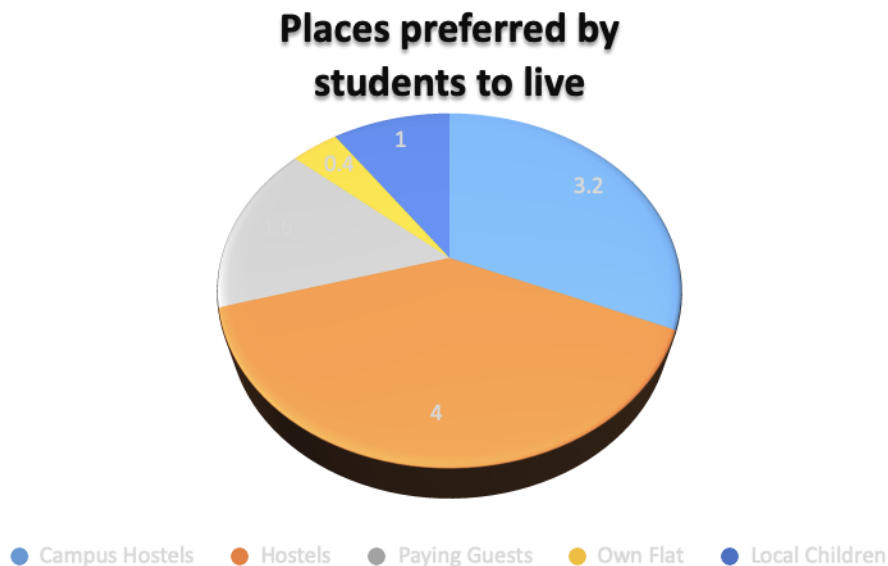


DATA ANALYSIS

We have done quite a research on internet about the information of hostels and their procedures. 95% of the hostels depend on the paper work for small things like day out pass and night out pass. 40-50% of the students are just doing sign the passes themselves. Even their parents don't know about it. So that's why we figure out to make a platform where we can make an easy way for warden, guards and parents to know about the children where they are and when they go out and when they come in for whole year. Our platform also has the data of each child of that particular hostel of each day so that warden and parents have absolute information. It will reduce the paper work and will save paper also so, it will be environment friendly too. And now a days all country is progressing on "Digital India" hence our project is one step towards it. It will not create mess for any person, it will be easy to access and use.

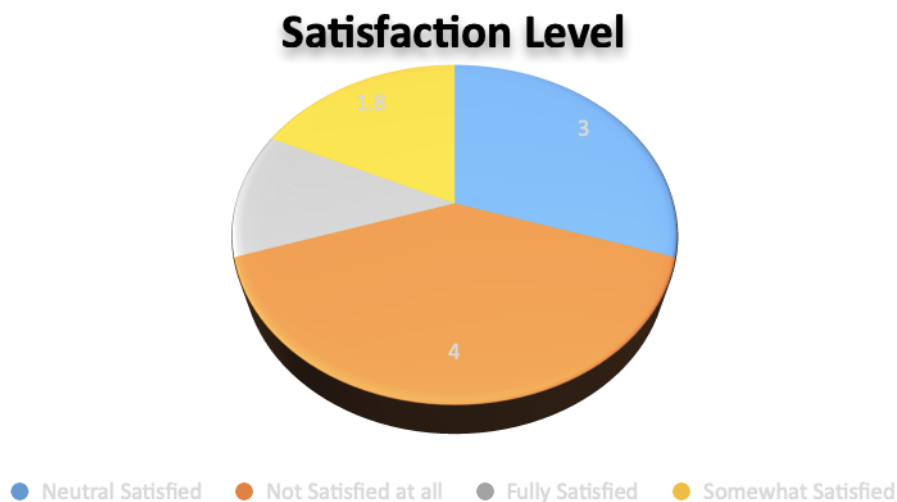
According to our research, we find out students are preferring which kind of place to stay. So there are five different places students are going. Firstly it is campus hostels, secondly it is hostels near their college, thirdly they are living as paying guests, fourthly own their flats and last but not least, there are local children.

We created a pie chart according to our research, we cannot say it's perfect but it is approximated idea of the students where they are preferring to live and it has all five different places preferred by the students.



Being students, we have faced many problems finding good hostel for that too we have created a platform that would have hostels around that particular area you have searched and what facility they are providing. If you search on Google, no one provides you the information hostels available in that area with their facilities and fee structures. There are some companies like oxfordcaps, stanzaliving which are providing different hostels but they are not providing best of all them. So our platform will provide that information, which will create easy platform to find a best hostel.

We have created another pie chart which will show the satisfaction of students with their stay in hostels. It will show that they are satisfy with the facilities provided by the hostels like food, games, rooms size, their small problems like problem of leakage and electricity many others. That's why we will have option of raising a query of problems faced by children.



DATABASE

We are using Firebase database service from Google. It's a fully-managed no SQL document database for mobile and web app development. It's designed to easily store and sync app data at global scale, and it's now available in beta. Firstly we used mongodb database but then we shifted to firebase database because mongodb is not real time and it stores the data in tabular form whereas firebase is real time and no server less document database that effortlessly scales to meet any demand, with no maintenance.

Why we opt for Cloud Firestore as our database:

- Cloud Firestore is an economical database solution that charges fees only for the operations performed, network bandwidth, and storage. The usage tab helps you keep track of your expenses at any given period.
- Cloud Firestore follows the ACID (atomicity, consistency, isolation, and durability) properties for transactions. Another attractive property of Google Firestore is its ability to perform indexed queries by combining filtering and sorting in a single query.
- Since Cloud Firestore is a Google product, it combines the prowess of the scalable Google Cloud and Firebase Realtime Database.
- It offers live synchronization and support and is quite beneficial in prototyping, and iterating easily due to its serverless properties.
- It provides excellent security for both mobile and web-based platforms. It also offers multi-region replication that ensures that your data is secure even in the event of a disaster.

Some of the key features of Firebase database-

- Documents and collections with powerful querying
- iOS, Android, and Web SDKs with offline data access
- Real-time data synchronization
- Automatic, multi-region data replication with strong consistency
- Node, Python, Go, and Java server SDKs

Managing data is still hard, you have to scale servers, handle intermittent connectivity, and deliver data with low latency.

- **Synchronizes data between devices in real-time.** Android, iOS, and Javascript SDKs sync your app data almost instantly. This makes it incredibly easy to build reactive apps, automatically sync data across devices, and build powerful collaborative features -- and if you don't need real-time sync, one-time reads are a first-class feature.
- **Uses collections and documents to structure and query data.** This data model is familiar and intuitive for many developers. It also allows for expressive queries. Queries scale with the size of your result set, not the size of your data set, so you'll get the same performance fetching 1 result from a set of 100, or 100,000,000.
- **Enables offline data access via a powerful, on-device database.** This local database means your app will function smoothly, even when your users lose connectivity. This offline mode is available on Web, iOS and Android.
- **Enables server-less development.** Cloud Firestore's client-side SDKs take care of the complex authentication and networking code you'd normally need to write yourself. Then, on the backend, we provide a powerful set of security rules so you can control access to your data. Security rules let you control which users can access which documents, and let you apply complex validation logic to your data as well. Combined, these features allow your mobile app to connect directly to your database.
- **Integrates with the rest of the Firebase platform.** You can easily configure Cloud Functions to run custom code whenever data is written, and our SDKs automatically integrate with Firebase Authentication, to help you get started quickly.

The Firebase Real-time Database, with its client SDKs and real-time capabilities, is all about making app development faster and easier. Since its launch, it has been adopted by hundreds of thousands of developers, and as its adoption grew, so did usage patterns. Developers began using the Real-time Database for more complex data and to build bigger apps, pushing the limits of the JSON data model and the performance of the database at scale. Cloud Firestore is inspired by what developers love most about the Firebase Real-time Database while also addressing its key limitations like data structuring, querying, and scaling.

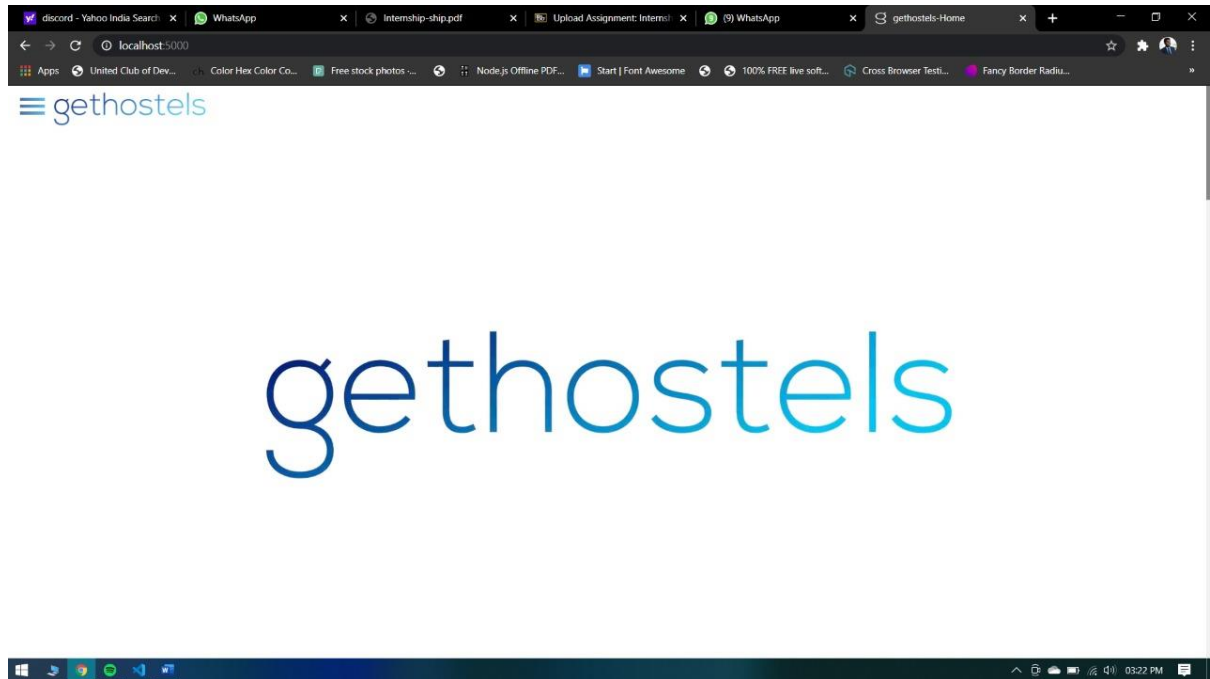
CONCLUSION

Our intension is to provide hassle free service to the students, as there has been no platform for the students to search hostels and now a days things are going online, we believe to provide management system to the hostels so that they can be benefitted with the project, we are starting from Bidholi, but in coming time we will try to implement it in Dehradun and further!

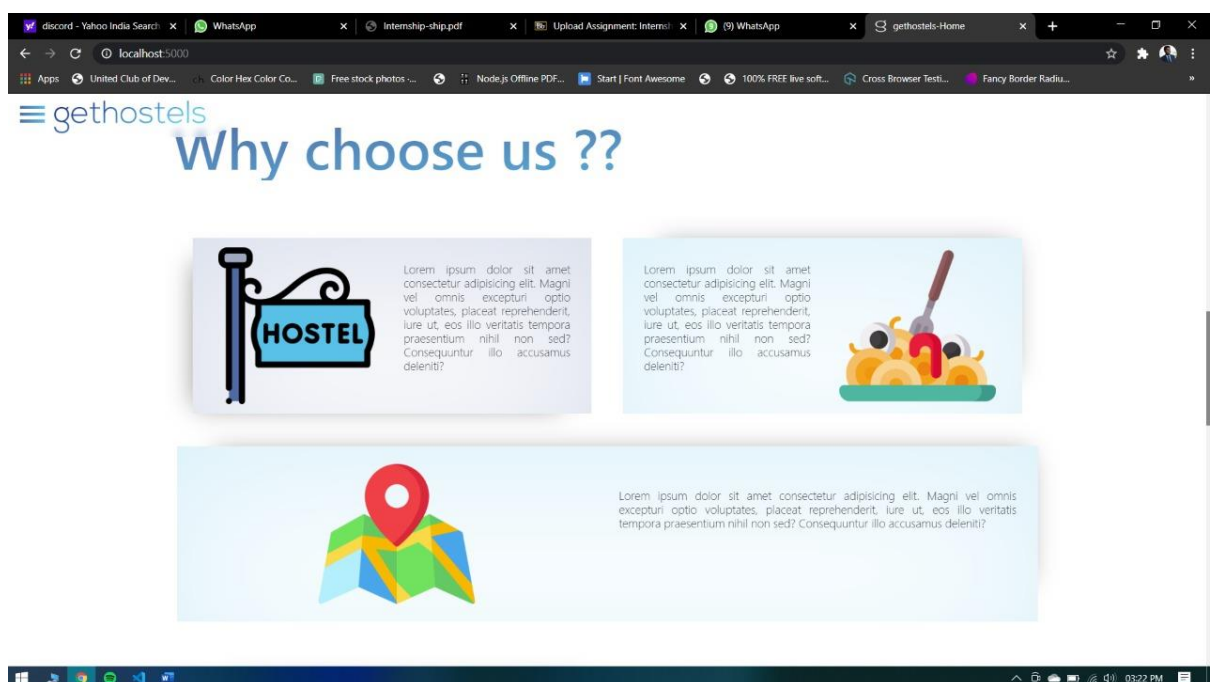
While working on this project we not all utilised our skills but also developed with all the efforts we put into it. The idea to work together and distribute the work among each other has let us to know the value of team work.

SCREENSHOTS

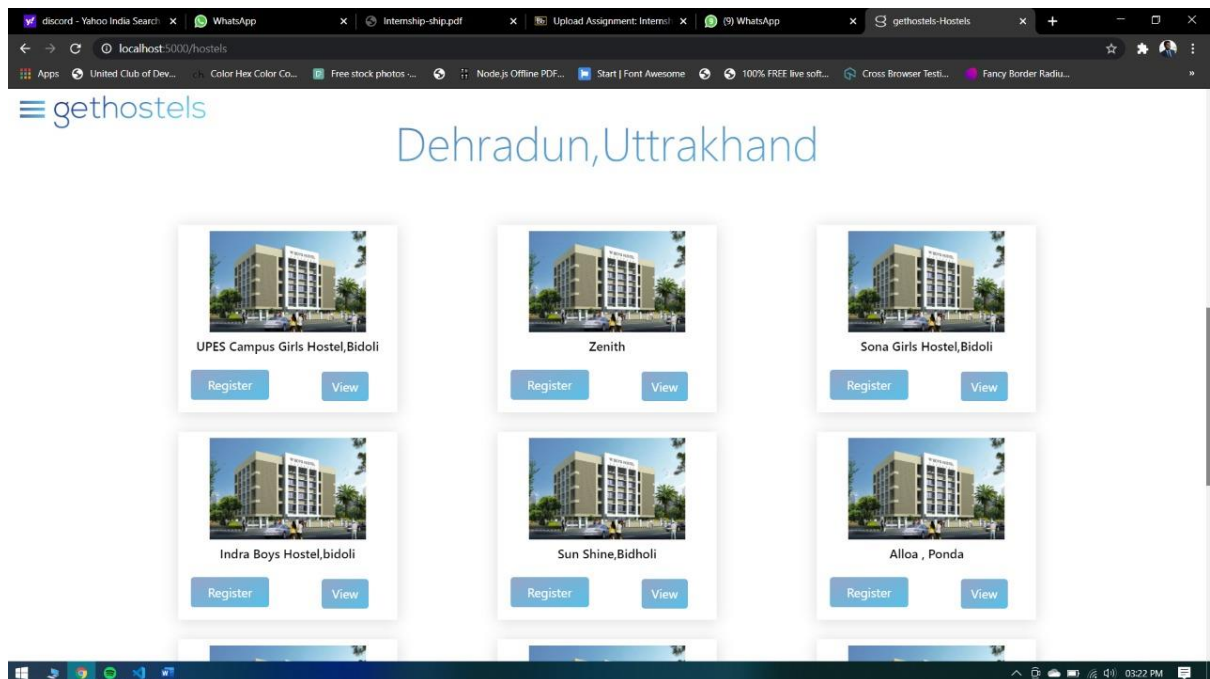
This is the front page of our website



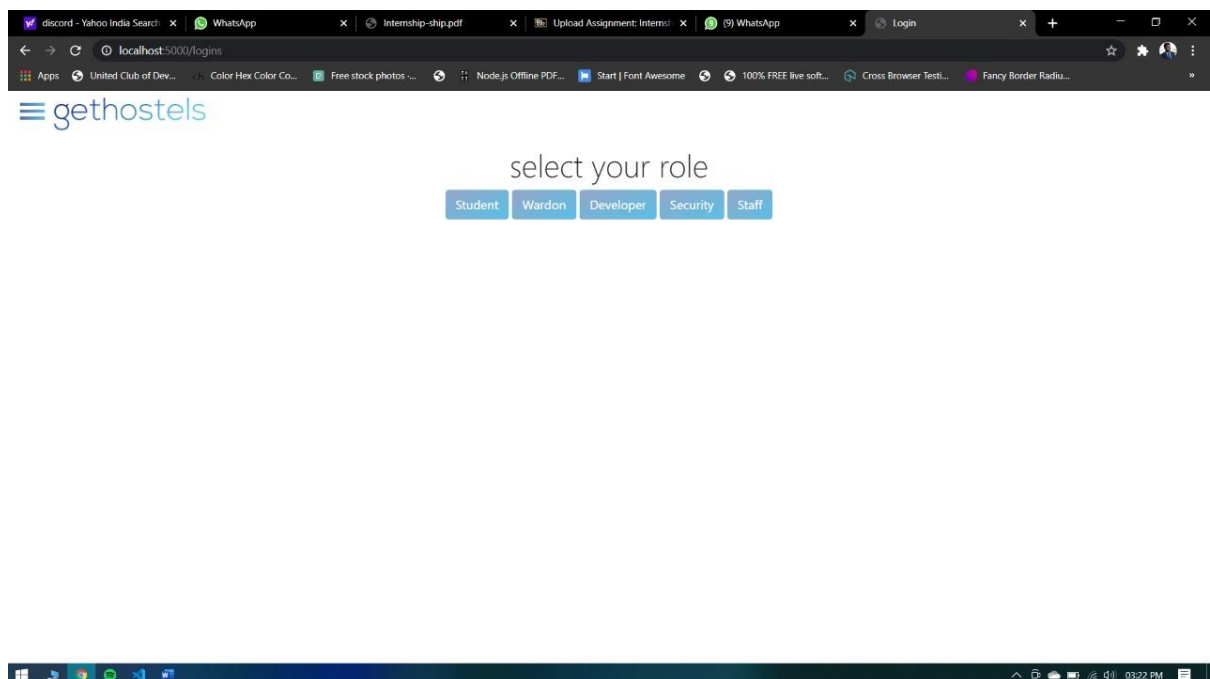
This is the home page where we have tell “why choose us”



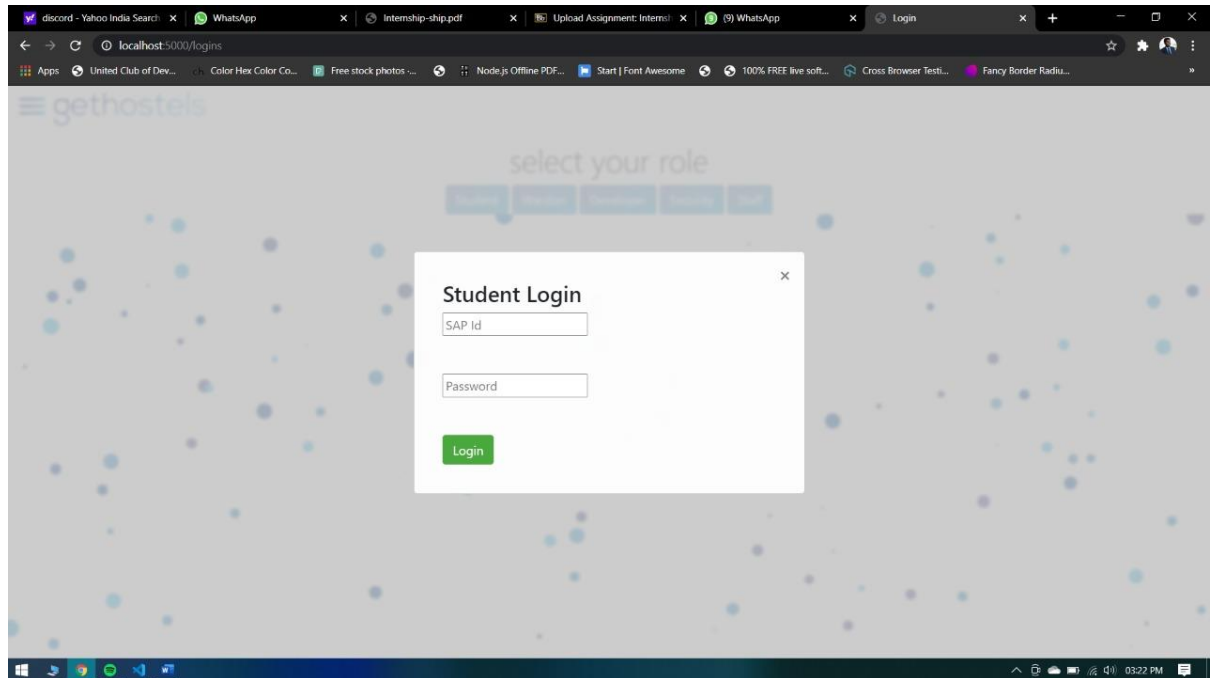
This is where our hostels list available



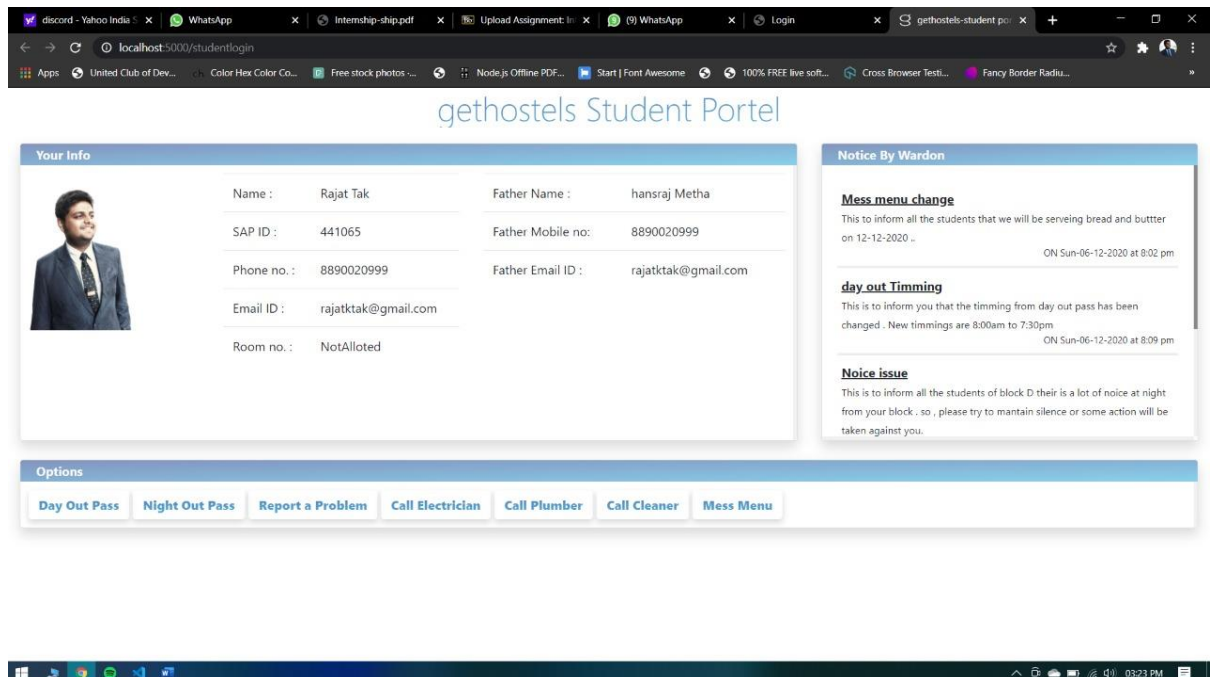
This is the main portal page with respective roles given



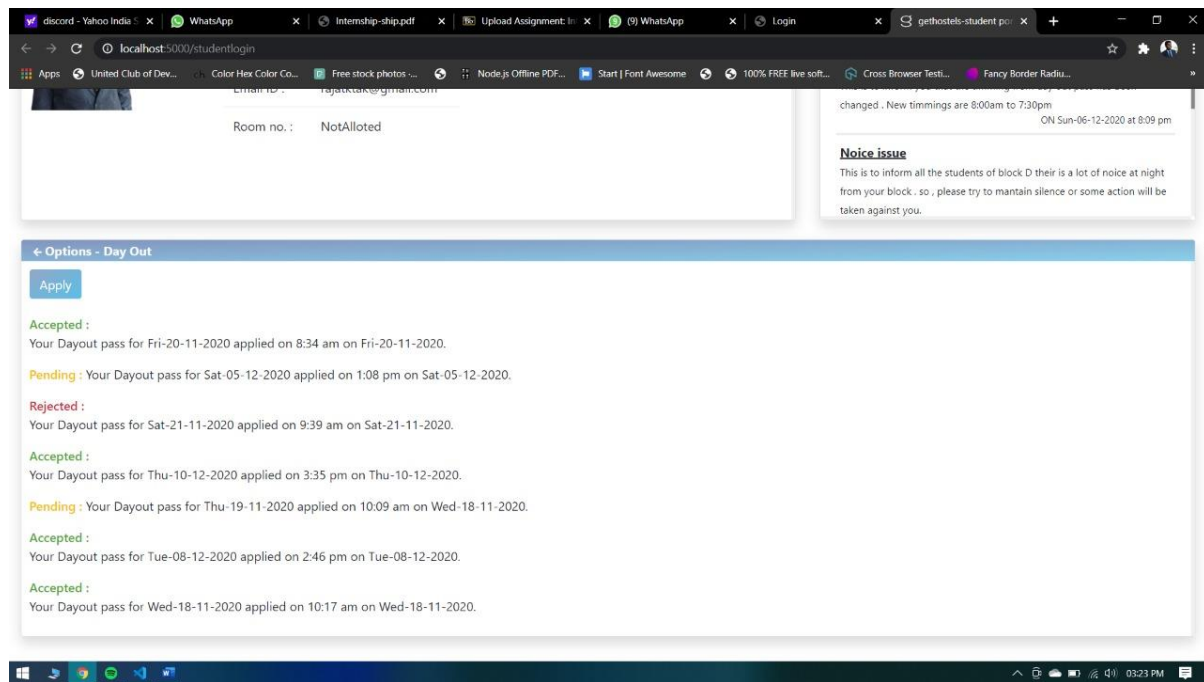
This is the login page for students



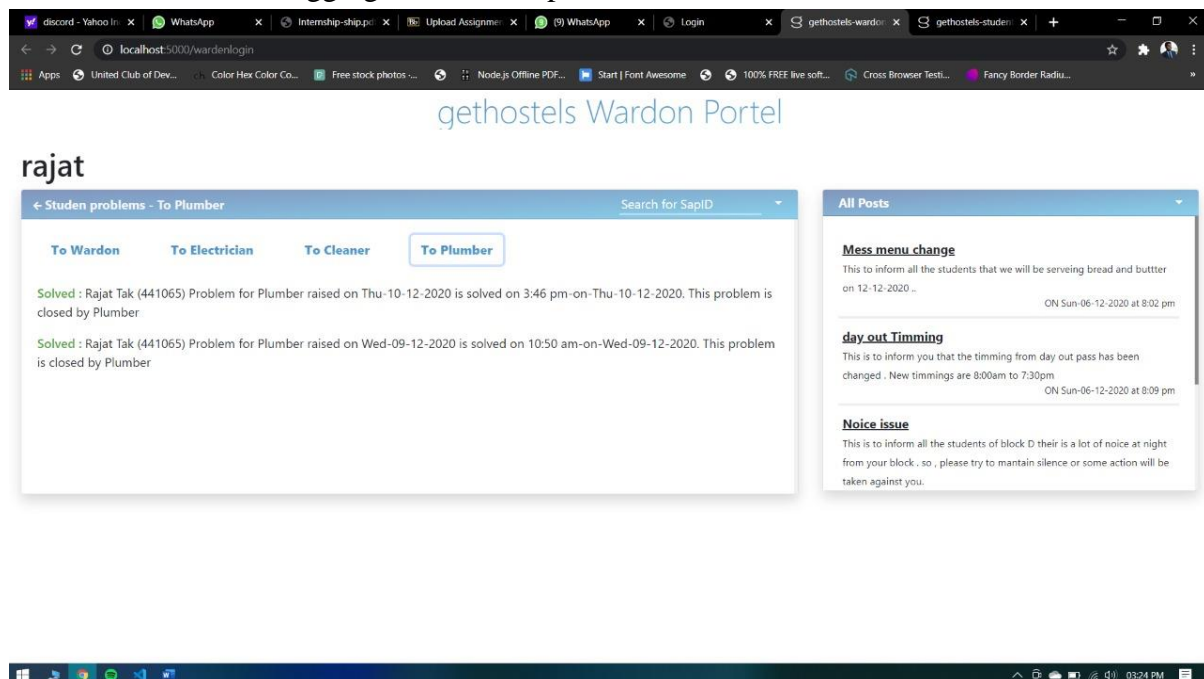
This is the page after logging in student portal



This is the where all the day-out night-out pass records and any query records are available



This is the look after logging into warden portal



This is some of the backend code-:

The screenshot displays a VS Code workspace for a Node.js application. The Explorer sidebar on the left shows a file structure with folders like 'routes', 'controllers', and 'static', and various files including 'server.js', 'index.js', 'login.js', 'student.js', 'wardon.js', 'security.js', 'style.css', 'uploads', 'alluserinfo.js', 'contact.js', 'cover.png', 'develop.js', 'error.js', 'home.js', 'hostels.js', 'index.js', 'infohostels.js', 'NewaDemo-LightLoaf', 'pay.js', 'register.js', 'student.js', 'style.css', 'up.js', 'upimg.js', 'upload.js', 'video.mp4', 'wlogin.js', 'app.js', 'cover.png', 'firebaseConnect.js', and 'gethostels740ae-firebase-admin'. The main editor shows the 'server.js' file with the following code:

```

1 // server.js
2 const express = require('express');
3 const mongoose = require('mongoose');
4 const passport = require('passport');
5 const session = require('express-session');
6 const bodyParser = require('body-parser');
7 const app = express();
8 const db = mongoose.connection;
9 const PORT = 3000;
10
11 // Middleware
12 app.use(bodyParser.json());
13 app.use(bodyParser.urlencoded({ extended: false,
14   }));
15 app.use(session({ secret: "ssshhhhh" }));
16 app.use(passport.initialize());
17 app.use(passport.session());
18
19 // Routes
20 app.get('/', async function (req, res) {
21   const hos = await hostels.find();
22   res.render("index", {
23     hos: hos,
24   });
25 });
26 app.get("/contact", async function (req, res) {
27   res.render("contact", {});
28 });
29 app.get("/logins", function (req, res) {
30   console.log(moment().tz("Asia/kolkata"));
31   res.render("wlogin");
32 });
33 app.get("/viewhostel", async function (req, res) {
34   const id = req.query.hid;
35   const hids = await hostels.findOne({
36     _id: id,
37   });
38   res.render("viewhostel", {
39     hids: hids,
40   });
41 });
42
43 // Start the server
44 app.listen(PORT, () => {
45   console.log(`Server started on port ${PORT}`);
46 });
47
48 // Export the app
49 module.exports = app;
50
51 // Test the server
52 if (require.main === module) {
53   app.listen(PORT, () => {
54     console.log(`Server started on port ${PORT}`);
55   });
56 }
57
58 // Export the app
59 module.exports = app;
60
61 // Test the server
62 if (require.main === module) {
63   app.listen(PORT, () => {
64     console.log(`Server started on port ${PORT}`);
65   });
66 }
67
68 // Export the app
69 module.exports = app;
70
71 // Test the server
72 if (require.main === module) {
73   app.listen(PORT, () => {
74     console.log(`Server started on port ${PORT}`);
75   });
76 }
77
78 // Export the app
79 module.exports = app;
80
81 // Test the server
82 if (require.main === module) {
83   app.listen(PORT, () => {
84     console.log(`Server started on port ${PORT}`);
85   });
86 }
87
88 // Export the app
89 module.exports = app;
90
91 // Test the server
92 if (require.main === module) {
93   app.listen(PORT, () => {
94     console.log(`Server started on port ${PORT}`);
95   });
96 }
97
98 // Export the app
99 module.exports = app;
100
101 // Test the server
102 if (require.main === module) {
103   app.listen(PORT, () => {
104     console.log(`Server started on port ${PORT}`);
105   });
106 }
107
108 // Export the app
109 module.exports = app;
110
111 // Test the server
112 if (require.main === module) {
113   app.listen(PORT, () => {
114     console.log(`Server started on port ${PORT}`);
115   });
116 }
117
118 // Export the app
119 module.exports = app;
120
121 // Test the server
122 if (require.main === module) {
123   app.listen(PORT, () => {
124     console.log(`Server started on port ${PORT}`);
125   });
126 }
127
128 // Export the app
129 module.exports = app;
130
131 // Test the server
132 if (require.main === module) {
133   app.listen(PORT, () => {
134     console.log(`Server started on port ${PORT}`);
135   });
136 }
137
138 // Export the app
139 module.exports = app;
140
141 // Test the server
142 if (require.main === module) {
143   app.listen(PORT, () => {
144     console.log(`Server started on port ${PORT}`);
145   });
146 }
147
148 // Export the app
149 module.exports = app;
150
151 // Test the server
152 if (require.main === module) {
153   app.listen(PORT, () => {
154     console.log(`Server started on port ${PORT}`);
155   });
156 }
157
158 // Export the app
159 module.exports = app;
160
161 // Test the server
162 if (require.main === module) {
163   app.listen(PORT, () => {
164     console.log(`Server started on port ${PORT}`);
165   });
166 }
167
168 // Export the app
169 module.exports = app;
170
171 // Test the server
172 if (require.main === module) {
173   app.listen(PORT, () => {
174     console.log(`Server started on port ${PORT}`);
175   });
176 }
177
178 // Export the app
179 module.exports = app;
180
181 // Test the server
182 if (require.main === module) {
183   app.listen(PORT, () => {
184     console.log(`Server started on port ${PORT}`);
185   });
186 }
187
188 // Export the app
189 module.exports = app;
190
191 // Test the server
192 if (require.main === module) {
193   app.listen(PORT, () => {
194     console.log(`Server started on port ${PORT}`);
195   });
196 }
197
198 // Export the app
199 module.exports = app;
200
201 // Test the server
202 if (require.main === module) {
203   app.listen(PORT, () => {
204     console.log(`Server started on port ${PORT}`);
205   });
206 }
207
208 // Export the app
209 module.exports = app;
210
211 // Test the server
212 if (require.main === module) {
213   app.listen(PORT, () => {
214     console.log(`Server started on port ${PORT}`);
215   });
216 }
217
218 // Export the app
219 module.exports = app;
220
221 // Test the server
222 if (require.main === module) {
223   app.listen(PORT, () => {
224     console.log(`Server started on port ${PORT}`);
225   });
226 }
227
228 // Export the app
229 module.exports = app;
230
231 // Test the server
232 if (require.main === module) {
233   app.listen(PORT, () => {
234     console.log(`Server started on port ${PORT}`);
235   });
236 }
237
238 // Export the app
239 module.exports = app;
240
241 // Test the server
242 if (require.main === module) {
243   app.listen(PORT, () => {
244     console.log(`Server started on port ${PORT}`);
245   });
246 }
247
248 // Export the app
249 module.exports = app;
250
251 // Test the server
252 if (require.main === module) {
253   app.listen(PORT, () => {
254     console.log(`Server started on port ${PORT}`);
255   });
256 }
257
258 // Export the app
259 module.exports = app;
260
261 // Test the server
262 if (require.main === module) {
263   app.listen(PORT, () => {
264     console.log(`Server started on port ${PORT}`);
265   });
266 }
267
268 // Export the app
269 module.exports = app;
270
271 // Test the server
272 if (require.main === module) {
273   app.listen(PORT, () => {
274     console.log(`Server started on port ${PORT}`);
275   });
276 }
277
278 // Export the app
279 module.exports = app;
280
281 // Test the server
282 if (require.main === module) {
283   app.listen(PORT, () => {
284     console.log(`Server started on port ${PORT}`);
285   });
286 }
287
288 // Export the app
289 module.exports = app;
290
291 // Test the server
292 if (require.main === module) {
293   app.listen(PORT, () => {
294     console.log(`Server started on port ${PORT}`);
295   });
296 }
297
298 // Export the app
299 module.exports = app;
300
301 // Test the server
302 if (require.main === module) {
303   app.listen(PORT, () => {
304     console.log(`Server started on port ${PORT}`);
305   });
306 }
307
308 // Export the app
309 module.exports = app;
310
311 // Test the server
312 if (require.main === module) {
313   app.listen(PORT, () => {
314     console.log(`Server started on port ${PORT}`);
315   });
316 }
317
318 // Export the app
319 module.exports = app;
320
321 // Test the server
322 if (require.main === module) {
323   app.listen(PORT, () => {
324     console.log(`Server started on port ${PORT}`);
325   });
326 }
327
328 // Export the app
329 module.exports = app;
330
331 // Test the server
332 if (require.main === module) {
333   app.listen(PORT, () => {
334     console.log(`Server started on port ${PORT}`);
335   });
336 }
337
338 // Export the app
339 module.exports = app;
340
341 // Test the server
342 if (require.main === module) {
343   app.listen(PORT, () => {
344     console.log(`Server started on port ${PORT}`);
345   });
346 }
347
348 // Export the app
349 module.exports = app;
350
351 // Test the server
352 if (require.main === module) {
353   app.listen(PORT, () => {
354     console.log(`Server started on port ${PORT}`);
355   });
356 }
357
358 // Export the app
359 module.exports = app;
360
361 // Test the server
362 if (require.main === module) {
363   app.listen(PORT, () => {
364     console.log(`Server started on port ${PORT}`);
365   });
366 }
367
368 // Export the app
369 module.exports = app;
370
371 // Test the server
372 if (require.main === module) {
373   app.listen(PORT, () => {
374     console.log(`Server started on port ${PORT}`);
375   });
376 }
377
378 // Export the app
379 module.exports = app;
380
381 // Test the server
382 if (require.main === module) {
383   app.listen(PORT, () => {
384     console.log(`Server started on port ${PORT}`);
385   });
386 }
387
388 // Export the app
389 module.exports = app;
390
391 // Test the server
392 if (require.main === module) {
393   app.listen(PORT, () => {
394     console.log(`Server started on port ${PORT}`);
395   });
396 }
397
398 // Export the app
399 module.exports = app;
400
401 // Test the server
402 if (require.main === module) {
403   app.listen(PORT, () => {
404     console.log(`Server started on port ${PORT}`);
405   });
406 }
407
408 // Export the app
409 module.exports = app;
410
411 // Test the server
412 if (require.main === module) {
413   app.listen(PORT, () => {
414     console.log(`Server started on port ${PORT}`);
415   });
416 }
417
418 // Export the app
419 module.exports = app;
420
421 // Test the server
422 if (require.main === module) {
423   app.listen(PORT, () => {
424     console.log(`Server started on port ${PORT}`);
425   });
426 }
427
428 // Export the app
429 module.exports = app;
430
431 // Test the server
432 if (require.main === module) {
433   app.listen(PORT, () => {
434     console.log(`Server started on port ${PORT}`);
435   });
436 }
437
438 // Export the app
439 module.exports = app;
440
441 // Test the server
442 if (require.main === module) {
443   app.listen(PORT, () => {
444     console.log(`Server started on port ${PORT}`);
445   });
446 }
447
448 // Export the app
449
```

The screenshot displays the Visual Studio Code interface with the following details:

- Explorer (Left):** Shows a file tree with folders like `views` and `uploads`. The file `index.ejs` is selected and highlighted in blue.
- Editor (Center):** Displays the content of `index.ejs`. The code is a JavaScript snippet for a hamburger menu toggle, using jQuery and DOM manipulation. It includes comments like `// document.getElementById` and `const canvas = document.getElementById("canvas")`.
- Terminal (Bottom):** Shows the command `PS > node server.js` and the output `server started on port 5000`.
- Status Bar (Bottom):** Indicates the current file is `index.ejs`, line 618, column 1, using UTF-8 encoding.

