**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJS22ITL604 DATE:**

**COURSE NAME: Full Stack Web Development Laboratory**  **CLASS: TYBTech**

**Name: Anish Sharma Roll no:I011**

**EXPERIMENT NO. 08**

**CO/LO:** CO1-Develop a full stack web application.

**AIM / OBJECTIVE:** Deploy the completed MERN stack project to a chosen hosting service (e.g., Heroku, Netlify, Render, Vercel).

**THEORY**:

Deploying a full-stack web application involves making the application accessible on the internet. The MERN stack (MongoDB, Express.js, React.js, Node.js) requires deployment of both backend and frontend, along with database configuration. Proper deployment ensures scalability, accessibility, and continuous availability of applications.

**Technologies/Platforms Used:**

* Frontend Hosting: Netlify / Vercel
* Backend Hosting: Render / Railway / Heroku
* Database: MongoDB Atlas
* Version Control: Git & GitHub

# Step 1: Prepare the MERN Application for Deployment

1. Ensure both frontend and backend are fully functional locally.
2. Connect backend to MongoDB Atlas (cloud-based database).
3. Add the production build script in frontend package.json:

**"scripts": {**

**"start": "react-scripts start",**

**"build": "react-scripts build"**

**}**

## Step 2: Deploy Backend (Express.js) to Render (or Heroku)

1. **Create GitHub Repository and push backend code.**

**COURSE CODE: DJS22ITL604 DATE:**

1. **Create Procfile (for Heroku users)** web: node server.js

**COURSE NAME: Full Stack Web Development Laboratory**  **CLASS: TYBTech**

1. **Environment Variables**:
   * Create .env file for sensitive data like database URL and port.
   * Example

MONGODB\_URI=your\_mongodb\_atlas\_url PORT=5000

1. **Update server.js to use environment variables.**  const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

1. **Render Deployment:** Go to https://render.com

Create a new Web Service > Connect GitHub Repo > Select branch

Add environment variables

Deploy

## Step 3: Deploy Frontend (React.js) to Netlify/Vercel

1. **Build React App** cd frontend npm run build
2. **Netlify Deployment:** Visit https://netlify.com

Import GitHub Repo or drag and drop /build folder

Set build directory

Set environment variables if API URLs are dynamic

**COURSE NAME: Full Stack Web Development Laboratory**  **CLASS: TYBTech**  **COURSE CODE: DJS22ITL604 DATE:**

1. **Vercel Deployment (Alternative):**

Visit https://vercel.com

Connect to GitHub

Set Root Directory as frontend

Set environment variables

**Step 4: Connect Frontend to Deployed Backend** • In frontend, change API URLs to deployed backend URL.

* Use .env in frontend for dynamic URLs:

REACT\_APP\_API\_URL=https://your-backend-url.onrender.com/api

* Access API in code: axios.post(`${process.env.REACT\_APP\_API\_URL}/users/register`, data);

## Step 5: Test Deployed App

* Visit deployed frontend URL.
* Register user via form.
* Check backend receives data and database stores it.

**Code:**

**Server.js**

const express=require('express') const cors=require('cors') const colors=require('colors') const dotenv=require('dotenv').config() const connectDB=require('./config/db') **COURSE CODE: DJS22ITL604 DATE:**

const {errorHandler}=require('./middleware/errorMiddleware') const PORT =process.env.PORT ||8000 connectDB() const app=express() app.use(cors({

origin: 'https://fs2010.netlify.app', // Replace with your frontend's origin methods: 'GET,POST,PUT,DELETE', allowedHeaders: 'Content-Type,Authorization', // Include 'Authorization' here

}));

app.use(function(req, res, next) { res.header("Access-Control-Allow-Origin", "https://fs2010.netlify.app");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept"); next(); });

app.use(express.json())

app.use(express.urlencoded({ extended: false })) app.get('/',(req,res)=>{

res.status(200).json({message:'Welcome to the Support Desk Api'})

})

app.use('/api/users',require('./routes/userRoutes')) app.use('/api/tickets',require('./routes/ticketRoutes')) app.use(errorHandler)

app.listen(PORT,()=>{console.log(`Server Started on port ${PORT}`)})

**COURSE CODE: DJS22ITL604 DATE:**  **frontend**

**import axios from 'axios' const API\_URL='https://ticket-backend-8.onrender.com/api/users'**

**//Register user**

**const register=async(userData)=>{ const response = await axios.post(API\_URL,userData) if(response.data)**

**{**

**localStorage.setItem('user',JSON.stringify(response.data))**

**}**

**return response.data**

**}**

**const login=async(userData)=>{**

**const response = await axios.post('https://ticket-backend-**

**8.onrender.com/api/users/login',userData) console.log(response.data) if(response.data)**

**{**

**localStorage.setItem('user',JSON.stringify(response.data))**

**}**

**return response.data**

**}**

**//Logout user**

**const logout=()=>localStorage.removeItem('user') const authService={**

**COURSE CODE: DJS22ITL604 DATE:**   **register, logout,login**

**}**

**export default authService**

**import axios from 'axios' const API\_URL='https://ticket-backend-7.onrender.com/api/tickets'**

**//Create new Ticket**

**const createTicket = async (ticketData,token) => { const config = { headers: {**

**Authorization: `Bearer ${token}`,**

**},**

**};**

**const response = await axios.post(API\_URL,ticketData,config); return response.data;**

**}**

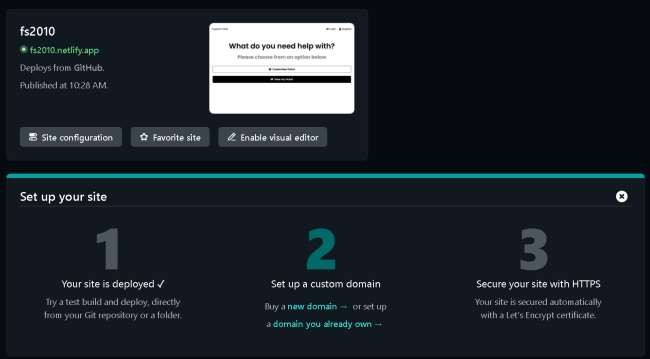
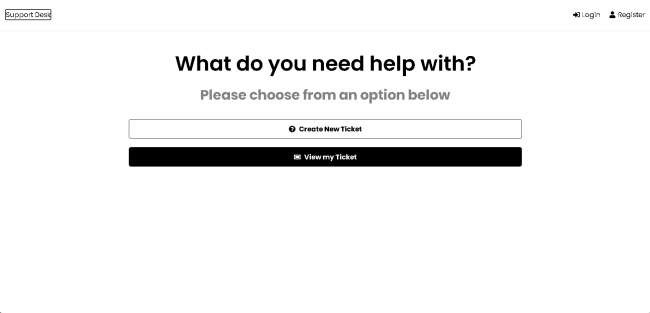
**const ticketService={ createTicket**

**}**

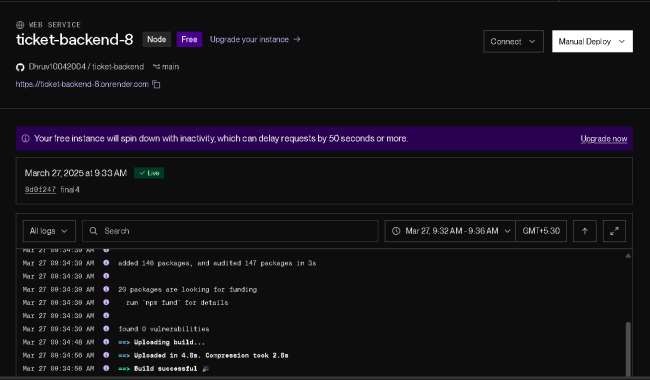
**export default ticketService**

**COURSE CODE: DJS22ITL604 DATE:**

**Output:**



**COURSE CODE: DJS22ITL604 DATE:**



Conclusion: Successfully deployed the site.

**COURSE NAME: Full Stack Web Development Laboratory**  **CLASS: TYBTech**

**BOOKS AND WEB RESOURCES:**

* React Documentation
* Express.js Guide
* Render Hosting Guide
* Netlify Deployment

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJS22ITL604 DATE:**

* MongoDB Atlas
* YouTube Guide: Deploy MERN App

**WRITE-UP QUESTIONS:**

1. Why is input validation necessary in both frontend and backend?
2. What is the purpose of error handling middleware in Express.js?
3. How can security be improved further in this application?