dAssist – Internal Ticketing Web App

Awesome, I’ve got you. Below is a chapter-wise, copy-paste, click-by-click guide that your developer can follow exactly. It builds a Node.js + Express + SQLite app with Tailwind + vanilla JS on the frontend, Quill editor for rich text, Nodemailer via Gmail SMTP (sending from info@dolluzcorp.com), an Admin panel with EJS, exports (PDF and Excel), and deployment on your DigitalOcean droplet with Nginx + Certbot for dassist.dolluzcorp.in and admin.dolluzcorp.in.

If you see placeholders like <<<REPLACE\_ME>>>, do not leave them as is. Replace exactly as instructed.

Chapter 1 - Environment setup (Windows + VS Code)

Your dev mentioned “Visual Basics”. I will assume Visual Studio Code on Windows. If they truly use Visual Basic IDE, ask me and I will adapt.

1.1 Install prerequisites

1. Install Node.js 20 LTS

* Open Chrome.
* Go to nodejs.org.
* Click LTS Windows Installer.
* Run installer, keep defaults, check “Add to PATH”.
* After install, open Command Prompt and run:

node -v

npm -v

Both should print versions.

1. Install Git for Windows

* Download from git-scm.com.
* Install with defaults.
* Open Git Bash and run:

git --version

1. Install VS Code

* Download from code.visualstudio.com.
* Install with defaults.
* Open VS Code, install Extensions:
  + “ESLint” by Microsoft
  + “Prettier” by Prettier
  + “EditorConfig for VS Code”

1. Optional for node-gyp builds

* From an elevated PowerShell:

npm install --global --production windows-build-tools

If it errors, skip. We do not rely on native modules.

1.2 Create project folder

1. Open Command Prompt.
2. Run:

cd %USERPROFILE%

mkdir dassist

cd dassist

1.3 Initialize Node project and folders

npm init -y

npm pkg set type="module"

mkdir src public public/assets public/js public/css src/config src/models src/routes src/utils src/middleware src/views uploads exports

1.4 Install dependencies

npm install express express-session connect-session-sequelize sequelize sqlite3 bcryptjs csurf helmet compression morgan cors multer uuid validator nodemailer exceljs pdfkit sanitize-html dayjs

npm install quill --save

npm install --save-dev nodemon dotenv

1.5 Create EditorConfig and Prettier

Create .editorconfig at project root:

root = true

[\*]

charset = utf-8

end\_of\_line = lf

insert\_final\_newline = true

indent\_style = space

indent\_size = 2

Create .prettierrc:

{

"singleQuote": true,

"semi": true,

"printWidth": 100

}

1.6 Git ignore

Create .gitignore:

node\_modules

.env

uploads

exports

1.7 Environment variables

Create .env in project root. Replace values carefully.

# General

PORT=3000

SESSION\_SECRET=<<<REPLACE\_ME\_WITH\_A\_LONG\_RANDOM\_STRING>>>

# Database

DB\_STORAGE=./dassist.sqlite

# Mail

SMTP\_HOST=smtp.gmail.com

SMTP\_PORT=465

SMTP\_SECURE=true

SMTP\_USER=info@dolluzcorp.com

SMTP\_PASS=<<<REPLACE\_ME\_WITH\_GOOGLE\_WORKSPACE\_APP\_PASSWORD>>>

MAIL\_FROM="Dolluz Support <info@dolluzcorp.com>"

MAIL\_TO\_SUPPORT=info@dolluzcorp.com

# App

APP\_PUBLIC\_URL=https://dassist.dolluzcorp.in

APP\_ADMIN\_URL=https://admin.dolluzcorp.in

# File uploads

MAX\_FILE\_MB=10

For Gmail Workspace, create an App Password and use it for SMTP\_PASS. Do not use your normal password.

1.8 Package scripts

Update package.json "scripts" section:

"scripts": {

"dev": "nodemon src/server.js",

"start": "node src/server.js",

"db:sync": "node src/utils/sync.js",

"seed:employees": "node src/utils/seedEmployees.js",

"seed:admin": "node src/utils/seedAdmin.js"

}

Chapter 2 - Development code (copy-paste)

2.1 Server entry - src/server.js

import 'dotenv/config';

import express from 'express';

import path from 'path';

import helmet from 'helmet';

import compression from 'compression';

import morgan from 'morgan';

import session from 'express-session';

import csrf from 'csurf';

import { fileURLToPath } from 'url';

import { sequelize, SessionStore } from './config/db.js';

import publicRoutes from './routes/public.js';

import adminRoutes from './routes/admin.js';

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

const app = express();

// Security and basics

app.use(helmet({ contentSecurityPolicy: false }));

app.use(compression());

app.use(morgan('combined'));

app.use(express.json({ limit: '2mb' }));

app.use(express.urlencoded({ extended: true }));

// Sessions

app.use(

session({

secret: process.env.SESSION\_SECRET,

resave: false,

saveUninitialized: false,

store: new SessionStore({ db: sequelize }),

cookie: { httpOnly: true, sameSite: 'lax', maxAge: 1000 \* 60 \* 60 \* 8 }

})

);

// CSRF only on admin panel pages and forms

const csrfProtection = csrf();

// Static front site

app.use(express.static(path.join(\_\_dirname, '../public')));

// Views for admin

app.set('views', path.join(\_\_dirname, './views'));

app.set('view engine', 'ejs');

// Routes

app.use('/api', publicRoutes);

app.use('/admin', csrfProtection, adminRoutes);

// Health check

app.get('/healthz', (req, res) => res.json({ ok: true }));

const port = process.env.PORT || 3000;

app.listen(port, async () => {

await sequelize.authenticate();

console.log(`dAssist running on http://localhost:${port}`);

});

2.2 Database config - src/config/db.js

import 'dotenv/config';

import { Sequelize } from 'sequelize';

import connectSessionSequelize from 'connect-session-sequelize';

import session from 'express-session';

export const sequelize = new Sequelize({

dialect: 'sqlite',

storage: process.env.DB\_STORAGE,

logging: false

});

export const SessionStore = connectSessionSequelize(session.Store);

const init = async () => {

await sequelize.authenticate();

};

export default init;

2.3 Models - src/models/\*

src/models/Employee.js

import { DataTypes } from 'sequelize';

import { sequelize } from '../config/db.js';

const Employee = sequelize.define('Employee', {

id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },

name: { type: DataTypes.STRING, allowNull: false },

email: { type: DataTypes.STRING, allowNull: false, unique: true },

mobile: { type: DataTypes.STRING, allowNull: true },

department: { type: DataTypes.STRING, allowNull: true },

employeeType: { type: DataTypes.STRING, allowNull: true }

});

export default Employee;

src/models/Ticket.js

import { DataTypes } from 'sequelize';

import { sequelize } from '../config/db.js';

const Ticket = sequelize.define('Ticket', {

id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },

ticketId: { type: DataTypes.STRING, unique: true, allowNull: false },

email: { type: DataTypes.STRING, allowNull: false },

name: { type: DataTypes.STRING, allowNull: false },

mobile: { type: DataTypes.STRING },

altMobile: { type: DataTypes.STRING, allowNull: false },

department: { type: DataTypes.STRING },

employeeType: { type: DataTypes.STRING },

location: { type: DataTypes.STRING, allowNull: false },

category: { type: DataTypes.STRING, allowNull: false },

priority: { type: DataTypes.STRING, allowNull: false },

preferredContact: { type: DataTypes.STRING, allowNull: false },

subject: { type: DataTypes.STRING(250), allowNull: false },

descriptionHtml: { type: DataTypes.TEXT, allowNull: false },

status: {

type: DataTypes.ENUM(

'Open',

'In Review',

'In Progress',

'Action Pending',

'Cancelled',

'Resolved'

),

defaultValue: 'Open'

}

});

export default Ticket;

src/models/Comment.js

import { DataTypes } from 'sequelize';

import { sequelize } from '../config/db.js';

const Comment = sequelize.define('Comment', {

id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },

ticketId: { type: DataTypes.INTEGER, allowNull: false },

text: { type: DataTypes.TEXT, allowNull: false },

addedBy: { type: DataTypes.STRING, allowNull: false } // 'admin:<email>' or 'system'

});

export default Comment;

src/models/Attachment.js

import { DataTypes } from 'sequelize';

import { sequelize } from '../config/db.js';

const Attachment = sequelize.define('Attachment', {

id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },

ticketId: { type: DataTypes.INTEGER, allowNull: false },

filename: { type: DataTypes.STRING, allowNull: false },

storedPath: { type: DataTypes.STRING, allowNull: false },

size: { type: DataTypes.INTEGER, allowNull: false },

mimetype: { type: DataTypes.STRING, allowNull: false }

});

export default Attachment;

src/models/AdminUser.js

import { DataTypes } from 'sequelize';

import { sequelize } from '../config/db.js';

const AdminUser = sequelize.define('AdminUser', {

id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },

email: { type: DataTypes.STRING, allowNull: false, unique: true },

passwordHash: { type: DataTypes.STRING, allowNull: false },

name: { type: DataTypes.STRING, allowNull: false }

});

export default AdminUser;

src/models/index.js

import Employee from './Employee.js';

import Ticket from './Ticket.js';

import Comment from './Comment.js';

import Attachment from './Attachment.js';

import AdminUser from './AdminUser.js';

// Relations

Ticket.hasMany(Comment, { foreignKey: 'ticketId' });

Comment.belongsTo(Ticket, { foreignKey: 'ticketId' });

Ticket.hasMany(Attachment, { foreignKey: 'ticketId' });

Attachment.belongsTo(Ticket, { foreignKey: 'ticketId' });

export { Employee, Ticket, Comment, Attachment, AdminUser };

2.4 Utils

src/utils/generateId.js

export function generateTicketId() {

const now = new Date();

const year = now.getFullYear();

const rand = Math.floor(10000 + Math.random() \* 89999);

return `TCK-${year}-${rand}`;

}

src/utils/mailer.js

import nodemailer from 'nodemailer';

export const transporter = nodemailer.createTransport({

host: process.env.SMTP\_HOST,

port: Number(process.env.SMTP\_PORT),

secure: String(process.env.SMTP\_SECURE) === 'true',

auth: {

user: process.env.SMTP\_USER,

pass: process.env.SMTP\_PASS

}

});

export async function sendMail({ to, subject, html }) {

return transporter.sendMail({

from: process.env.MAIL\_FROM,

to,

subject,

html

});

}

src/utils/exporters.js

import ExcelJS from 'exceljs';

import PDFDocument from 'pdfkit';

import fs from 'fs';

import path from 'path';

export async function exportTicketsExcel(tickets, outPath) {

const wb = new ExcelJS.Workbook();

const ws = wb.addWorksheet('Tickets');

ws.columns = [

{ header: 'Ticket ID', key: 'ticketId', width: 18 },

{ header: 'Name', key: 'name', width: 20 },

{ header: 'Email', key: 'email', width: 28 },

{ header: 'Department', key: 'department', width: 16 },

{ header: 'Employee Type', key: 'employeeType', width: 16 },

{ header: 'Location', key: 'location', width: 12 },

{ header: 'Category', key: 'category', width: 14 },

{ header: 'Priority', key: 'priority', width: 10 },

{ header: 'Preferred Contact', key: 'preferredContact', width: 16 },

{ header: 'Subject', key: 'subject', width: 40 },

{ header: 'Status', key: 'status', width: 14 },

{ header: 'Created At', key: 'createdAt', width: 20 }

];

tickets.forEach(t => ws.addRow(t));

await wb.xlsx.writeFile(outPath);

}

export function exportTicketPDF(ticket, outPath) {

const doc = new PDFDocument({ margin: 40 });

const stream = fs.createWriteStream(outPath);

doc.pipe(stream);

doc.fontSize(18).text(`Ticket ${ticket.ticketId}`, { underline: true });

doc.moveDown();

doc.fontSize(12);

const lines = [

`Name: ${ticket.name}`,

`Email: ${ticket.email}`,

`Mobile: ${ticket.mobile || ''}`,

`Alt Mobile: ${ticket.altMobile}`,

`Dept: ${ticket.department} | Type: ${ticket.employeeType}`,

`Location: ${ticket.location}`,

`Category: ${ticket.category} | Priority: ${ticket.priority}`,

`Preferred Contact: ${ticket.preferredContact}`,

`Status: ${ticket.status}`,

`Subject: ${ticket.subject}`

];

lines.forEach(l => doc.text(l));

doc.moveDown();

doc.text('Description (HTML stripped):');

const stripped = ticket.descriptionHtml.replace(/<[^>]\*>/g, ' ');

doc.text(stripped);

doc.end();

return new Promise(resolve => stream.on('finish', resolve));

}

src/utils/sync.js

import { sequelize } from '../config/db.js';

import { Employee, Ticket, Comment, Attachment, AdminUser } from '../models/index.js';

(async () => {

await sequelize.sync({ alter: true });

console.log('DB synced');

process.exit(0);

})();

src/utils/seedEmployees.js

import fs from 'fs';

import path from 'path';

import { Employee } from '../models/index.js';

import { fileURLToPath } from 'url';

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

(async () => {

// Example static seed. Replace or extend as needed.

const list = [

{ name: 'Anand T', email: 'anand@dolluzcorp.com', mobile: '9999999999', department: 'IT', employeeType: 'Full Time' },

{ name: 'Nathan', email: 'nathan@dolluzcorp.com', mobile: '8888888888', department: 'IT', employeeType: 'Full Time' }

];

for (const e of list) {

await Employee.upsert(e);

}

console.log('Seeded Employees:', list.length);

process.exit(0);

})();

src/utils/seedAdmin.js

import bcrypt from 'bcryptjs';

import { AdminUser } from '../models/index.js';

(async () => {

const email = 'admin@dolluzcorp.in';

const name = 'Portal Admin';

const passwordPlain = '<<<REPLACE\_ME\_STRONG\_PASSWORD>>>';

const passwordHash = await bcrypt.hash(passwordPlain, 10);

await AdminUser.findOrCreate({

where: { email },

defaults: { name, passwordHash }

});

console.log('Admin seeded. Email:', email);

console.log('Set password now in DB if needed.');

process.exit(0);

})();

2.5 Middleware - src/middleware/auth.js

export function requireAdmin(req, res, next) {

if (!req.session?.admin) {

return res.redirect('/admin/login');

}

next();

}

2.6 Public routes (APIs for frontend) - src/routes/public.js

import express from 'express';

import multer from 'multer';

import path from 'path';

import validator from 'validator';

import sanitizeHtml from 'sanitize-html';

import { Employee, Ticket, Attachment } from '../models/index.js';

import { generateTicketId } from '../utils/generateId.js';

import { sendMail } from '../utils/mailer.js';

const router = express.Router();

// Multer for attachments

const maxFileBytes = Number(process.env.MAX\_FILE\_MB || 10) \* 1024 \* 1024;

const upload = multer({

dest: 'uploads/',

limits: { fileSize: maxFileBytes }

});

// Validate email and get profile

router.post('/validate-email', async (req, res) => {

const { email } = req.body;

if (!email || !validator.isEmail(email) || !email.endsWith('@dolluzcorp.com')) {

return res.status(400).json({ ok: false, message: 'Invalid email format or domain.' });

}

const emp = await Employee.findOne({ where: { email } });

if (!emp) return res.status(404).json({ ok: false, message: 'Email not registered.' });

return res.json({

ok: true,

data: {

name: emp.name,

email: emp.email,

mobile: emp.mobile || '',

department: emp.department || '',

employeeType: emp.employeeType || ''

}

});

});

// Create ticket

router.post('/tickets', upload.array('attachments', 5), async (req, res) => {

try {

const {

name,

email,

mobile,

altMobile,

department,

employeeType,

location,

category,

priority,

preferredContact,

subject,

descriptionHtml

} = req.body;

// Basic validations

const required = {

email,

name,

altMobile,

location,

category,

priority,

preferredContact,

subject,

descriptionHtml

};

for (const [k, v] of Object.entries(required)) {

if (!v || String(v).trim() === '') {

return res.status(400).json({ ok: false, message: `Missing field: ${k}` });

}

}

if (!email.endsWith('@dolluzcorp.com')) {

return res.status(400).json({ ok: false, message: 'Email must be @dolluzcorp.com' });

}

if (String(subject).length > 250) {

return res.status(400).json({ ok: false, message: 'Subject must be 250 characters or less.' });

}

// Sanitize HTML

const safeHtml = sanitizeHtml(descriptionHtml, {

allowedTags: ['b', 'i', 'em', 'strong', 'ul', 'ol', 'li', 'p', 'br'],

allowedAttributes: {}

});

const ticketId = generateTicketId();

const ticket = await Ticket.create({

ticketId,

email,

name,

mobile,

altMobile,

department,

employeeType,

location,

category,

priority,

preferredContact,

subject,

descriptionHtml: safeHtml

});

// Save attachments

if (req.files?.length) {

for (const f of req.files) {

await Attachment.create({

ticketId: ticket.id,

filename: f.originalname,

storedPath: f.path,

size: f.size,

mimetype: f.mimetype

});

}

}

// Emails

const baseFooter =

'For any concerns, amendments, or notes, please write to info@dolluzcorp.com with the Ticket ID in the subject line.<br><br>This is an auto-generated email. Please do not reply directly.';

const submitHtmlEmp = `

Hello ${name},<br><br>

Your ticket <b>${ticketId}</b> regarding "<b>${subject}</b>" has been <b>Submitted</b>.<br>

Current Status: <b>Open</b><br><br>${baseFooter}`;

const submitHtmlOps = `

New Ticket Submitted: <b>${ticketId}</b><br>

From: ${name} &lt;${email}&gt;<br>

Subject: ${subject}<br>

Priority: ${priority}<br>

Category: ${category}<br>

Location: ${location}<br><br>

Please log in to Admin to triage.`;

await sendMail({

to: email,

subject: `[Ticket ID: ${ticketId}] - Submitted`,

html: submitHtmlEmp

});

await sendMail({

to: process.env.MAIL\_TO\_SUPPORT,

subject: `[Ticket ID: ${ticketId}] - New Ticket Submitted`,

html: submitHtmlOps

});

return res.json({ ok: true, ticketId });

} catch (err) {

console.error(err);

return res.status(500).json({ ok: false, message: 'Server error.' });

}

});

export default router;

2.7 Admin routes and views

src/routes/admin.js

import express from 'express';

import bcrypt from 'bcryptjs';

import { requireAdmin } from '../middleware/auth.js';

import { AdminUser, Ticket, Comment, Employee } from '../models/index.js';

import { exportTicketsExcel, exportTicketPDF } from '../utils/exporters.js';

import path from 'path';

const router = express.Router();

// Login

router.get('/login', (req, res) => {

return res.render('login', { csrfToken: req.csrfToken(), error: null });

});

router.post('/login', async (req, res) => {

const { email, password } = req.body;

const admin = await AdminUser.findOne({ where: { email } });

if (!admin) return res.render('login', { csrfToken: req.csrfToken(), error: 'Invalid login' });

const ok = await bcrypt.compare(password, admin.passwordHash);

if (!ok) return res.render('login', { csrfToken: req.csrfToken(), error: 'Invalid login' });

req.session.admin = { id: admin.id, email: admin.email, name: admin.name };

return res.redirect('/admin/tickets');

});

router.get('/logout', (req, res) => {

req.session.destroy(() => res.redirect('/admin/login'));

});

// Dashboard list

router.get('/tickets', requireAdmin, async (req, res) => {

const { status, category, priority, q } = req.query;

const where = {};

if (status) where.status = status;

if (category) where.category = category;

if (priority) where.priority = priority;

if (q) where.subject = { $like: `%${q}%` };

const rows = await Ticket.findAll({ where, order: [['createdAt', 'DESC']] });

res.render('tickets', { csrfToken: req.csrfToken(), rows });

});

// View single

router.get('/tickets/:id', requireAdmin, async (req, res) => {

const row = await Ticket.findByPk(req.params.id, { include: ['Comments', 'Attachments'] });

if (!row) return res.redirect('/admin/tickets');

res.render('ticket', { csrfToken: req.csrfToken(), row });

});

// Status update

router.post('/tickets/:id/status', requireAdmin, async (req, res) => {

const { status } = req.body;

const row = await Ticket.findByPk(req.params.id);

if (!row) return res.redirect('/admin/tickets');

row.status = status;

await row.save();

res.redirect(`/admin/tickets/${row.id}`);

});

// Comment add

router.post('/tickets/:id/comments', requireAdmin, async (req, res) => {

const { text } = req.body;

await Comment.create({

ticketId: req.params.id,

text,

addedBy: `admin:${req.session.admin.email}`

});

res.redirect(`/admin/tickets/${req.params.id}`);

});

// Export all or filtered

router.get('/export/excel', requireAdmin, async (req, res) => {

const rows = await Ticket.findAll({ order: [['createdAt', 'DESC']] });

const outPath = path.join('exports', `tickets\_${Date.now()}.xlsx`);

await exportTicketsExcel(rows.map(r => r.toJSON()), outPath);

res.download(outPath);

});

// Export one as PDF

router.get('/tickets/:id/export/pdf', requireAdmin, async (req, res) => {

const row = await Ticket.findByPk(req.params.id);

const outPath = path.join('exports', `ticket\_${row.ticketId}.pdf`);

await exportTicketPDF(row.toJSON(), outPath);

res.download(outPath);

});

// Employees basic CRUD - list and add

router.get('/employees', requireAdmin, async (req, res) => {

const rows = await Employee.findAll({ order: [['name', 'ASC']] });

res.render('employees', { csrfToken: req.csrfToken(), rows });

});

router.post('/employees', requireAdmin, async (req, res) => {

const { name, email, mobile, department, employeeType } = req.body;

await Employee.upsert({ name, email, mobile, department, employeeType });

res.redirect('/admin/employees');

});

router.post('/employees/delete', requireAdmin, async (req, res) => {

const { email } = req.body;

await Employee.destroy({ where: { email } });

res.redirect('/admin/employees');

});

export default router;

2.7.1 Views - EJS

Create src/views/layout.ejs

<!doctype html>

<html>

<head>

<meta charset="utf-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1"/>

<title><%= title || 'dAssist Admin' %></title>

<link href="https://cdn.tailwindcss.com" rel="stylesheet" />

</head>

<body class="bg-gray-50">

<nav class="bg-white border-b shadow-sm">

<div class="max-w-6xl mx-auto px-4 py-3 flex justify-between">

<div class="font-semibold">dAssist Admin</div>

<div>

<a class="text-sm text-blue-600" href="/admin/tickets">Tickets</a>

<span class="mx-2">|</span>

<a class="text-sm text-blue-600" href="/admin/employees">Employees</a>

<span class="mx-2">|</span>

<a class="text-sm text-red-600" href="/admin/logout">Logout</a>

</div>

</div>

</nav>

<main class="max-w-6xl mx-auto p-6">

<%- body %>

</main>

</body>

</html>

Create src/views/login.ejs

<!doctype html>

<html>

<head>

<meta charset="utf-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1"/>

<title>Admin Login</title>

<script src="https://cdn.tailwindcss.com"></script>

</head>

<body class="bg-gray-50 flex items-center justify-center h-screen">

<form method="post" class="bg-white p-6 rounded-xl shadow w-full max-w-sm">

<input type="hidden" name="\_csrf" value="<%= csrfToken %>"/>

<h1 class="text-xl font-semibold mb-4">Admin Login</h1>

<% if (error) { %><p class="text-red-600 text-sm mb-2"><%= error %></p><% } %>

<label class="text-sm">Email</label>

<input name="email" class="border rounded w-full p-2 mb-3" type="email" required/>

<label class="text-sm">Password</label>

<input name="password" class="border rounded w-full p-2 mb-4" type="password" required/>

<button class="w-full bg-blue-600 text-white rounded p-2">Login</button>

</form>

</body>

</html>

Create src/views/tickets.ejs

<% layout('layout') -%>

<section>

<div class="flex items-center justify-between mb-4">

<h2 class="text-xl font-semibold">Tickets</h2>

<a class="text-sm text-blue-600 border px-3 py-1 rounded" href="/admin/export/excel">Export Excel</a>

</div>

<table class="w-full bg-white rounded shadow text-sm">

<thead>

<tr class="bg-gray-100">

<th class="p-2 text-left">Ticket ID</th>

<th class="p-2">Subject</th>

<th class="p-2">Email</th>

<th class="p-2">Priority</th>

<th class="p-2">Status</th>

<th class="p-2">Action</th>

</tr>

</thead>

<tbody>

<% rows.forEach(r => { %>

<tr class="border-t">

<td class="p-2"><%= r.ticketId %></td>

<td class="p-2"><%= r.subject %></td>

<td class="p-2"><%= r.email %></td>

<td class="p-2"><%= r.priority %></td>

<td class="p-2"><%= r.status %></td>

<td class="p-2">

<a class="text-blue-600" href="/admin/tickets/<%= r.id %>">Open</a>

</td>

</tr>

<% }) %>

</tbody>

</table>

</section>

Create src/views/ticket.ejs

<% layout('layout') -%>

<h2 class="text-xl font-semibold mb-2">Ticket <%= row.ticketId %></h2>

<p class="text-sm text-gray-600 mb-4"><%= row.subject %></p>

<div class="grid grid-cols-1 md:grid-cols-2 gap-4 bg-white p-4 rounded shadow mb-6">

<div><b>Name</b>: <%= row.name %></div>

<div><b>Email</b>: <%= row.email %></div>

<div><b>Dept</b>: <%= row.department %></div>

<div><b>Type</b>: <%= row.employeeType %></div>

<div><b>Location</b>: <%= row.location %></div>

<div><b>Category</b>: <%= row.category %></div>

<div><b>Priority</b>: <%= row.priority %></div>

<div><b>Preferred Contact</b>: <%= row.preferredContact %></div>

</div>

<div class="bg-white p-4 rounded shadow mb-6">

<h3 class="font-semibold mb-2">Description</h3>

<div class="prose max-w-none"><%- row.descriptionHtml %></div>

</div>

<div class="flex items-center gap-3 mb-4">

<form method="post" action="/admin/tickets/<%= row.id %>/status">

<input type="hidden" name="\_csrf" value="<%= csrfToken %>"/>

<select name="status" class="border rounded p-2">

<%- ['Open','In Review','In Progress','Action Pending','Cancelled','Resolved'].map(s => `<option ${row.status===s?'selected':''}>${s}</option>`).join('') %>

</select>

<button class="ml-2 bg-blue-600 text-white px-3 py-2 rounded">Update</button>

</form>

<a class="text-sm text-blue-700" href="/admin/tickets/<%= row.id %>/export/pdf">Export PDF</a>

</div>

<div class="bg-white p-4 rounded shadow">

<h3 class="font-semibold mb-2">Comments</h3>

<ul class="mb-3">

<% if (row.Comments && row.Comments.length) { row.Comments.forEach(c => { %>

<li class="border-b py-2"><%= c.text %> <span class="text-xs text-gray-400">- <%= c.addedBy %></span></li>

<% }) } else { %>

<li class="text-sm text-gray-600">No comments yet.</li>

<% } %>

</ul>

<form method="post" action="/admin/tickets/<%= row.id %>/comments">

<input type="hidden" name="\_csrf" value="<%= csrfToken %>"/>

<textarea name="text" rows="3" class="border rounded w-full p-2" required></textarea>

<button class="mt-2 bg-gray-800 text-white px-3 py-2 rounded">Add Comment</button>

</form>

</div>

Create src/views/employees.ejs

<% layout('layout') -%>

<h2 class="text-xl font-semibold mb-4">Employees</h2>

<form method="post" class="bg-white p-4 rounded shadow mb-6">

<input type="hidden" name="\_csrf" value="<%= csrfToken %>"/>

<div class="grid grid-cols-1 md:grid-cols-5 gap-2">

<input name="name" placeholder="Name" class="border p-2 rounded" required/>

<input name="email" placeholder="Email @dolluzcorp.com" class="border p-2 rounded" required/>

<input name="mobile" placeholder="Mobile" class="border p-2 rounded"/>

<input name="department" placeholder="Department" class="border p-2 rounded"/>

<input name="employeeType" placeholder="Employee Type" class="border p-2 rounded"/>

</div>

<button class="mt-3 bg-blue-600 text-white px-3 py-2 rounded">Add or Update</button>

</form>

<table class="w-full bg-white rounded shadow text-sm">

<thead><tr class="bg-gray-100">

<th class="p-2 text-left">Name</th><th class="p-2">Email</th><th class="p-2">Dept</th><th class="p-2">Type</th><th class="p-2">Action</th>

</tr></thead>

<tbody>

<% rows.forEach(r => { %>

<tr class="border-t">

<td class="p-2"><%= r.name %></td>

<td class="p-2"><%= r.email %></td>

<td class="p-2"><%= r.department %></td>

<td class="p-2"><%= r.employeeType %></td>

<td class="p-2">

<form method="post" action="/admin/employees/delete" onsubmit="return confirm('Delete employee?')">

<input type="hidden" name="\_csrf" value="<%= csrfToken %>"/>

<input type="hidden" name="email" value="<%= r.email %>"/>

<button class="text-red-600">Delete</button>

</form>

</td>

</tr>

<% }) %>

</tbody>

</table>

2.8 Frontend (public)

Place your logo at public/assets/logo.png (use the file you shared).

Create public/index.html

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1"/>

<title>dAssist - Raise a Ticket</title>

<script src="https://cdn.tailwindcss.com"></script>

<link href="https://cdn.quilljs.com/1.3.6/quill.snow.css" rel="stylesheet" />

</head>

<body class="bg-gray-50 min-h-screen">

<div class="max-w-3xl mx-auto p-6">

<div class="text-center mb-6">

<img src="/assets/logo.png" class="mx-auto w-28 h-28" alt="Dolluz Corp"/>

<h1 class="text-2xl font-semibold mt-3">Submit a Ticket</h1>

<p class="text-gray-500 text-sm">One Place, One Start, One Team</p>

</div>

<div class="bg-white p-6 rounded-xl shadow">

<!-- Stage 1 -->

<div id="stage-email">

<label class="block text-sm font-medium">Work Email <span class="text-red-600">\*</span></label>

<input id="email" type="email" class="border rounded w-full p-2 mt-1" placeholder="you@dolluzcorp.com"/>

<button id="btnValidate" class="mt-3 w-full bg-yellow-500 text-white rounded p-2">Validate Email</button>

<p id="emailError" class="text-red-600 text-sm mt-2 hidden"></p>

</div>

<!-- Stage 2 -->

<form id="ticketForm" class="hidden mt-4" enctype="multipart/form-data">

<div class="grid grid-cols-1 md:grid-cols-2 gap-3">

<div>

<label class="text-sm">Name</label>

<input id="name" name="name" class="border rounded w-full p-2 bg-gray-100" disabled/>

</div>

<div>

<label class="text-sm">Email</label>

<input id="email2" name="email" class="border rounded w-full p-2 bg-gray-100" disabled/>

</div>

<div>

<label class="text-sm">Mobile No</label>

<input id="mobile" name="mobile" class="border rounded w-full p-2 bg-gray-100" disabled/>

</div>

<div>

<label class="text-sm">Alternate Mobile No <span class="text-red-600">\*</span></label>

<input id="altMobile" name="altMobile" class="border rounded w-full p-2"/>

</div>

<div>

<label class="text-sm">Department</label>

<input id="department" name="department" class="border rounded w-full p-2 bg-gray-100" disabled/>

</div>

<div>

<label class="text-sm">Employee Type</label>

<input id="employeeType" name="employeeType" class="border rounded w-full p-2 bg-gray-100" disabled/>

</div>

<div>

<label class="text-sm">Location <span class="text-red-600">\*</span></label>

<select id="location" name="location" class="border rounded w-full p-2">

<option value="">Select</option>

<option>Chennai</option><option>Hyderabad</option><option>Pune</option><option>Nagpur</option>

</select>

</div>

<div>

<label class="text-sm">Category <span class="text-red-600">\*</span></label>

<select id="category" name="category" class="border rounded w-full p-2">

<option value="">Select</option>

<option>IT</option><option>HR</option><option>Admin</option><option>Finance</option>

</select>

</div>

<div>

<label class="text-sm">Priority <span class="text-red-600">\*</span></label>

<select id="priority" name="priority" class="border rounded w-full p-2">

<option value="">Select</option>

<option>Low</option><option>Medium</option><option>High</option>

</select>

</div>

<div>

<label class="text-sm">Preferred Contact <span class="text-red-600">\*</span></label>

<select id="preferredContact" name="preferredContact" class="border rounded w-full p-2">

<option value="">Select</option>

<option>Email</option><option>Call</option><option>WhatsApp</option>

</select>

</div>

</div>

<div class="mt-3">

<label class="text-sm">Subject <span class="text-red-600">\*</span></label>

<input id="subject" name="subject" maxlength="250" class="border rounded w-full p-2"/>

<p class="text-xs text-gray-500 mt-1"><span id="subjectCount">0</span>/250</p>

</div>

<div class="mt-3">

<label class="text-sm">Description <span class="text-red-600">\*</span></label>

<div id="editor" class="bg-white"></div>

<p class="text-xs text-gray-500 mt-1">Live word count: <span id="wordCount">0</span></p>

<input type="hidden" id="descriptionHtml" name="descriptionHtml"/>

</div>

<div class="mt-3">

<label class="text-sm">File Attachment (optional, max 10 MB each)</label>

<input id="attachments" name="attachments" class="border rounded w-full p-2" type="file" multiple/>

</div>

<button id="btnSubmit" class="mt-4 w-full bg-blue-600 text-white rounded p-2">Submit</button>

<p id="submitError" class="text-red-600 text-sm mt-2 hidden"></p>

<p class="text-gray-500 text-xs mt-4">

Note: Once submitted, your request will be routed to the respective support team. We are committed to addressing your concern as soon as possible. Kindly bear with us and we appreciate your patience.

</p>

</form>

</div>

</div>

<script src="https://cdn.quilljs.com/1.3.6/quill.js"></script>

<script src="/js/app.js"></script>

</body>

</html>

Create public/success.html

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1"/>

<title>Ticket Submitted</title>

<script src="https://cdn.tailwindcss.com"></script>

</head>

<body class="bg-gray-50 min-h-screen flex items-center">

<div class="max-w-xl mx-auto p-6 bg-white rounded-xl shadow text-center">

<img src="/assets/logo.png" class="mx-auto w-24 h-24" alt="Dolluz"/>

<h1 class="text-2xl font-semibold mt-3">Thank you!</h1>

<p class="mt-2">Your ticket has been submitted successfully.</p>

<p class="mt-2">Ticket ID: <span id="tid" class="font-mono font-semibold"></span></p>

<a href="/" class="mt-6 inline-block bg-blue-600 text-white px-4 py-2 rounded">Submit Another Ticket</a>

</div>

<script>

const url = new URL(window.location.href);

const tid = url.searchParams.get('id') || '';

document.getElementById('tid').textContent = tid;

</script>

</body>

</html>

Create public/js/app.js

const stageEmail = document.getElementById('stage-email');

const ticketForm = document.getElementById('ticketForm');

const btnValidate = document.getElementById('btnValidate');

const emailInput = document.getElementById('email');

const emailError = document.getElementById('emailError');

const nameEl = document.getElementById('name');

const email2El = document.getElementById('email2');

const mobileEl = document.getElementById('mobile');

const deptEl = document.getElementById('department');

const typeEl = document.getElementById('employeeType');

const subjectEl = document.getElementById('subject');

const subjectCount = document.getElementById('subjectCount');

subjectEl.addEventListener('input', () => (subjectCount.textContent = subjectEl.value.length));

const quill = new Quill('#editor', { theme: 'snow', modules: { toolbar: [['bold','italic'],[{ 'list':'ordered'},{ 'list':'bullet'}],['clean']] } });

const wordCountEl = document.getElementById('wordCount');

function getTextWords() {

const text = quill.getText().trim();

return text ? text.split(/\s+/).length : 0;

}

quill.on('text-change', () => (wordCountEl.textContent = getTextWords()));

btnValidate.addEventListener('click', async () => {

emailError.classList.add('hidden');

const email = emailInput.value.trim();

if (!email.endsWith('@dolluzcorp.com')) {

emailError.textContent = 'Only @dolluzcorp.com emails are allowed.';

emailError.classList.remove('hidden');

return;

}

const resp = await fetch('/api/validate-email', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ email })

});

const data = await resp.json();

if (!data.ok) {

emailError.textContent = 'This email is not registered with Dolluz Corp. Please contact Admin (info@dolluzcorp.com).';

emailError.classList.remove('hidden');

return;

}

nameEl.value = data.data.name;

email2El.value = data.data.email;

mobileEl.value = data.data.mobile || '';

deptEl.value = data.data.department || '';

typeEl.value = data.data.employeeType || '';

stageEmail.classList.add('hidden');

ticketForm.classList.remove('hidden');

});

// Submit

const submitBtn = document.getElementById('btnSubmit');

const submitError = document.getElementById('submitError');

ticketForm.addEventListener('submit', async (e) => {

e.preventDefault();

submitError.classList.add('hidden');

submitBtn.disabled = true;

submitBtn.textContent = 'Submitting...';

document.getElementById('descriptionHtml').value = quill.root.innerHTML;

const form = new FormData(ticketForm);

const resp = await fetch('/api/tickets', {

method: 'POST',

body: form

});

const data = await resp.json();

if (!data.ok) {

submitError.textContent = data.message || 'Please fill all the mandatory fields to submit the ticket';

submitError.classList.remove('hidden');

submitBtn.disabled = false;

submitBtn.textContent = 'Submit';

return;

}

window.location.href = `/success.html?id=${encodeURIComponent(data.ticketId)}`;

});

2.9 Test locally

1. In VS Code Terminal:

npm run db:sync

npm run seed:employees

npm run seed:admin

npm run dev

1. Open http://localhost:3000 for the portal.
2. Open http://localhost:3000/admin/login for admin.
3. Log in with the seeded admin and password you set.
4. Validate email using one of the seeded employees. Submit a ticket with a small file to confirm 10 MB limit.
5. Check mailbox for both employee and support notifications.

Chapter 3 - Push to GitHub

1. Create a new repository in GitHub named dassist under your org or account. Do not initialize with README.
2. In project root Terminal:

git init

git add .

git commit -m "Initial commit - dAssist"

git branch -M main

git remote add origin https://github.com/<<<YOUR\_GH\_USER\_OR\_ORG>>>/dassist.git

git push -u origin main

Chapter 4 - Deploy on DigitalOcean droplet root@64.227.135.222 with Nginx + Certbot

These commands are to be run exactly on the server over SSH.

4.1 SSH and system prep

ssh root@64.227.135.222

apt update && apt upgrade -y

apt install -y ufw git nginx

ufw allow OpenSSH

ufw allow 'Nginx Full'

ufw --force enable

4.2 Install Node.js and PM2

curl -fsSL https://deb.nodesource.com/setup\_20.x | bash -

apt install -y nodejs

npm i -g pm2

4.3 Clone the repo and set env

mkdir -p /var/www/dassist

cd /var/www/dassist

git clone https://github.com/<<<YOUR\_GH\_USER\_OR\_ORG>>>/dassist.git .

npm ci

Create the .env on server with same values you used locally:

nano .env

Paste contents and save.

Create folders and permissions:

mkdir -p uploads exports

chown -R www-data:www-data /var/www/dassist/uploads /var/www/dassist/exports

Sync DB on server:

npm run db:sync

Start with PM2:

pm2 start src/server.js --name dassist

pm2 save

pm2 startup systemd

Follow the printed command from pm2 startup to enable boot start.

4.4 Nginx server blocks

Create site for dassist.dolluzcorp.in and admin.dolluzcorp.in.

nano /etc/nginx/sites-available/dassist.conf

Paste:

server {

listen 80;

server\_name dassist.dolluzcorp.in;

location / {

proxy\_pass http://127.0.0.1:3000/;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

server {

listen 80;

server\_name admin.dolluzcorp.in;

location / {

proxy\_pass http://127.0.0.1:3000/admin/;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

Enable and test:

ln -s /etc/nginx/sites-available/dassist.conf /etc/nginx/sites-enabled/dassist.conf

nginx -t

systemctl reload nginx

4.5 SSL with Certbot

apt install -y certbot python3-certbot-nginx

certbot --nginx -d dassist.dolluzcorp.in -d admin.dolluzcorp.in --agree-tos -m info@dolluzcorp.com --no-eff-email

Choose redirect to HTTPS when prompted. Certbot sets up auto renew.

4.6 Log files and troubleshooting

* App logs: pm2 logs dassist
* Nginx: /var/log/nginx/access.log and error.log
* Certbot: /var/log/letsencrypt/letsencrypt.log

Chapter 5 - GoDaddy DNS for dolluzcorp.in

In GoDaddy for dolluzcorp.in:

1. Open DNS Management.
2. Add two A records:

* Name: dassist Value: 64.227.135.222 TTL: 1 hour
* Name: admin Value: 64.227.135.222 TTL: 1 hour

No CNAME here since we use the droplet directly. Wait for propagation.

Your email sending domain remains dolluzcorp.com through Gmail SMTP. Since you send via Google servers, SPF and DKIM are handled on dolluzcorp.com. Ensure your Google Workspace for dolluzcorp.com has SPF record including include:\_spf.google.com and DKIM enabled. This keeps deliverability strong even though the app runs on the .in domain.

Chapter 6 - Admin usage and workflows

1. Visit https://admin.dolluzcorp.in/admin/login.
2. Log in with the seeded admin email and password you set in seedAdmin.js.
3. Add employees in Employees page, or run a custom importer later if you give me a CSV.
4. Tickets page shows new submissions. Open, update status, add comments, and export PDF or Excel.
5. Employees never log in. They receive emails on submit, status change, and comments.

Chapter 7 - UX details honored

* Mandatory fields show red asterisk.
* Client validation plus server validation.
* Subject hard limit 250.
* Quill editor provides Bold, Italic, Bullets, Numbering with default font look similar to Calibri 11 on web.
* Live word count under description.
* Submit button disables with spinner text.
* After success, user lands on success.html with ticket id, and a button to submit another ticket.
* Errors appear in red under the button.
* Attachments capped at 10 MB each.

Chapter 8 - Common extensions you may ask for later

* Add reCAPTCHA v3 on submit endpoint.
* Rate limit by IP for /api/validate-email and /api/tickets.
* DKIM signing at app level if you move away from Gmail SMTP.
* Switch DB from SQLite to MySQL by changing Sequelize config and adding a connection DSN.

Final run check list

On your dev machine:

npm run db:sync

npm run seed:employees

npm run seed:admin

npm run dev

On the server after DNS is live:

pm2 restart dassist

nginx -t && systemctl reload nginx

certbot renew --dry-run

That is the full inch-by-inch setup with working code.