To create a modern CRUD API in Node.js with TypeScript, PostgreSQL, and other specified components, let’s follow these steps:

1. **Initialize Node.js Project:**

mkdir hospital-admissions-api  
cd hospital-admissions-api  
npm init -y

1. **Install Dependencies:**

npm install express pg prisma typescript @types/node @types/express dotenv body-parser http-errors eslint @typescript-eslint/eslint-plugin @typescript-eslint/parser jsonwebtoken bcryptjs joi winston morgan jest supertest ts-jest @types/jest

1. **Folder Structure:**

hospital-admissions-api/  
├── src/  
│ ├── controllers/  
│ │ └── admissionController.ts  
│ ├── models/  
│ │ └── admission.ts  
│ ├── routes/  
│ │ └── admissionRoutes.ts  
│ ├── database/  
│ │ └── index.ts  
│ ├── middleware/  
│ │ ├── authMiddleware.ts  
│ │ └── errorMiddleware.ts  
│ ├── utils/  
│ │ └── logger.ts  
│ ├── server.ts  
│ └── app.ts  
├── .env  
├── .eslintrc.json  
├── prisma/  
│ └── schema.prisma  
├── jest.config.js  
├── tsconfig.json  
└── package.json

1. **Initialize Prisma:**

npx prisma init

1. **Define Database Schema:**

// prisma/schema.prisma  
generator client {  
 provider = "prisma-client-js"  
}  
  
datasource db {  
 provider = "postgresql"  
 url = env("DATABASE\_URL")  
}  
  
model Admission {  
 id Int @id @default(autoincrement())  
 patientId String  
 admittedAt DateTime @default(now())  
 dischargedAt DateTime?  
}

1. **Initialize Prisma Client:**

npx prisma generate

1. **Setup Authentication and Authorization:**

// src/middleware/authMiddleware.ts  
import { Request, Response, NextFunction } from 'express';  
import jwt from 'jsonwebtoken';  
import createHttpError from 'http-errors';  
import { Logger } from '../utils/logger';  
  
export const authenticateJWT = (req: Request, res: Response, next: NextFunction) => {  
 const token = req.headers.authorization?.split(' ')[1];  
 if (token) {  
 jwt.verify(token, process.env.JWT\_SECRET!, (err, decoded) => {  
 if (err) {  
 return res.status(401).json({ message: 'Unauthorized' });  
 }  
 req.user = decoded;  
 next();  
 });  
 } else {  
 res.status(401).json({ message: 'Unauthorized' });  
 }  
};  
  
// src/middleware/errorMiddleware.ts  
import { Request, Response, NextFunction } from 'express';  
import createHttpError from 'http-errors';  
import { Logger } from '../utils/logger';  
  
export const errorHandler = (err: createHttpError.HttpError, req: Request, res: Response, next: NextFunction) => {  
 Logger.error(`${err.status || 500} - ${err.message} - ${req.originalUrl} - ${req.method} - ${req.ip}`);  
  
 res.status(err.status || 500).json({ error: err.message });  
};

1. **Implement Logging:**

// src/utils/logger.ts  
import winston from 'winston';  
  
export const Logger = winston.createLogger({  
 level: 'info',  
 format: winston.format.json(),  
 transports: [  
 new winston.transports.Console({  
 format: winston.format.simple(),  
 }),  
 new winston.transports.File({ filename: 'error.log', level: 'error' }),  
 new winston.transports.File({ filename: 'combined.log' }),  
 ],  
});

1. **Data Validation with Joi:**

// src/utils/validator.ts  
import Joi from 'joi';  
  
export const validateAdmission = (data: any) => {  
 const schema = Joi.object({  
 patientId: Joi.string().required(),  
 });  
 return schema.validate(data);  
};

1. **Write Tests:**

// jest.config.js  
module.exports = {  
 preset: 'ts-jest',  
 testEnvironment: 'node',  
 testPathIgnorePatterns: ['/node\_modules/', '/dist/'],  
};

// src/controllers/\_\_tests\_\_/admissionController.test.ts  
import request from 'supertest';  
import app from '../../app';  
  
describe('Admission API', () => {  
 it('should create a new admission', async () => {  
 const res = await request(app).post('/api/admissions').send({ patientId: '123' });  
 expect(res.status).toEqual(200);  
 expect(res.body).toHaveProperty('id');  
 });  
  
 // Other test cases for CRUD operations  
});

1. **Implement CRUD Endpoints:**

// src/controllers/admissionController.ts  
import { Request, Response } from 'express';  
import { PrismaClient } from '@prisma/client';  
import createHttpError from 'http-errors';  
import { validateAdmission } from '../utils/validator';  
  
const prisma = new PrismaClient();  
  
class AdmissionController {  
 async create(req: Request, res: Response) {  
 try {  
 const { error } = validateAdmission(req.body);  
 if (error) {  
 throw createHttpError(400, error.details[0].message);  
 }  
   
 const { patientId } = req.body;  
 const admission = await prisma.admission.create({ data: { patientId } });  
 res.json(admission);  
 } catch (error) {  
 res.status(error.status || 500).json({ error: error.message });  
 }  
 }  
  
 async getAll(req: Request, res: Response) {  
 try {  
 const admissions = await prisma.admission.findMany();  
 res.json(admissions);  
 } catch (error) {  
 res.status(500).json({ error: 'Internal Server Error' });  
 }  
 }  
  
 async getById(req: Request, res: Response) {  
 try {  
 const { id } = req.params;  
 const admission = await prisma.admission.findUnique({ where: { id: parseInt(id) } });  
 if (!admission) {  
 throw createHttpError(404, 'Admission not found');  
 }  
 res.json(admission);  
 } catch (error) {  
 res.status(error.status || 500).json({ error: error.message });  
 }  
 }  
  
 async update(req: Request, res: Response) {  
 try {  
 const { id } = req.params;  
 const { patientId } = req.body;  
 const admission = await prisma.admission.update({  
 where: { id: parseInt(id) },  
 data: { patientId },  
 });  
 res.json(admission);  
 } catch (error) {  
 res.status(error.status || 500).json({ error: error.message });  
 }  
 }  
  
 async delete(req: Request, res: Response) {  
 try {  
 const { id } = req.params;  
 await prisma.admission  
  
.delete({ where: { id: parseInt(id) } });  
 res.sendStatus(204);  
 } catch (error) {  
 res.status(error.status || 500).json({ error: error.message });  
 }  
 }  
}  
  
export default new AdmissionController();

// src/routes/admissionRoutes.ts  
import { Router } from 'express';  
import admissionController from '../controllers/admissionController';  
import { authenticateJWT } from '../middleware/authMiddleware';  
  
const router = Router();  
  
router.post('/', authenticateJWT, admissionController.create);  
router.get('/', authenticateJWT, admissionController.getAll);  
router.get('/:id', authenticateJWT, admissionController.getById);  
router.put('/:id', authenticateJWT, admissionController.update);  
router.delete('/:id', authenticateJWT, admissionController.delete);  
  
export default router;

1. **Set up Express Server:**

// src/app.ts  
import express, { Express } from 'express';  
import bodyParser from 'body-parser';  
import morgan from 'morgan';  
import { errorHandler } from './middleware/errorMiddleware';  
import admissionRoutes from './routes/admissionRoutes';  
  
class App {  
 public app: Express;  
  
 constructor() {  
 this.app = express();  
 this.config();  
 this.routes();  
 this.errorHandler();  
 }  
  
 private config(): void {  
 this.app.use(bodyParser.json());  
 this.app.use(bodyParser.urlencoded({ extended: false }));  
 this.app.use(morgan('combined'));  
 }  
  
 private routes(): void {  
 this.app.use('/api/admissions', admissionRoutes);  
 }  
  
 private errorHandler(): void {  
 this.app.use(errorHandler);  
 }  
}  
  
export default new App().app;

// src/server.ts  
import app from './app';  
  
const PORT = process.env.PORT || 3000;  
  
app.listen(PORT, () => {  
 console.log(`Server is running on port ${PORT}`);  
});

Now, you have a modern CRUD API for hospital admissions control using Node.js, TypeScript, PostgreSQL, and other specified components, including authentication, authorization, data validation, logging, and automated testing.