# Complete CI/CD Demo Project Tutorial

## Building a Full-Stack Application with Monorepo, Prisma, React, Playwright, and GitHub Actions

This comprehensive tutorial will guide you through creating a complete CI/CD demo project from scratch, covering backend API development, frontend React application, E2E testing with Playwright, and automated deployment pipelines.

## 📋 Table of Contents

1. [Project Setup & Monorepo Configuration](#Xebd94c4806d8e661c31d4bc9faf24ccf361c7b5)
2. [Backend API with Prisma Setup](#X8d4975794cdcdb7476b8d08b5f6e5719203bc28)
3. [Frontend React Application](#Xa47ff356b2e6b92a28c3ccfbdc81452ea4ccb5f)
4. [Playwright E2E Testing](#Xc8780917c56e12ad3bae487bf8421f7c7ad6707)
5. [GitHub Actions CI/CD Pipeline](#X21052700ec63a5f1bce91f49328236a7db6f644)
6. [Deployment & Production Setup](#X433da5cd0a296e9aa12a01765e50666be309d6c)

## 🎯 What We’ll Build

* **Backend**: Node.js/Express API with PostgreSQL database
* **Frontend**: React 19 application with TypeScript and Tailwind CSS
* **Authentication**: JWT-based auth with role-based access control
* **Testing**: Comprehensive E2E tests with Playwright
* **CI/CD**: Automated testing and deployment pipeline
* **Architecture**: Monorepo structure with workspace management

# 1. Project Setup & Monorepo Configuration

## 1.1 Initialize Project Structure

# Create project directory  
mkdir ci-cd-demo  
cd ci-cd-demo  
  
# Initialize root package.json  
npm init -y

## 1.2 Configure Monorepo with Workspaces

Edit package.json in the root directory:

{  
 "name": "ci-cd-demo",  
 "version": "1.0.0",  
 "description": "Full-stack CI/CD demo with React, Prisma, and Playwright",  
 "private": true,  
 "workspaces": [  
 "api",  
 "web"  
 ],  
 "scripts": {  
 "dev": "concurrently \"npm run dev --workspace=api\" \"npm run dev --workspace=web\"",  
 "build": "npm run build --workspace=api && npm run build --workspace=web",  
 "test": "npm run test --workspace=api && npm run test --workspace=web",  
 "e2e": "npm run e2e --workspace=web",  
 "lint": "npm run lint --workspace=api && npm run lint --workspace=web",  
 "clean": "rm -rf api/dist api/node\_modules web/dist web/node\_modules node\_modules"  
 },  
 "devDependencies": {  
 "concurrently": "^8.2.0"  
 },  
 "engines": {  
 "node": ">=18.0.0",  
 "npm": ">=9.0.0"  
 }  
}

## 1.3 Install Root Dependencies

npm install

## 1.4 Create Project Structure

# Create directories  
mkdir api web  
mkdir -p .github/workflows  
mkdir -p docs  
  
# Create basic files  
touch README.md  
touch .gitignore  
touch .env.example

## 1.5 Configure Git Ignore

Create .gitignore:

# Dependencies  
node\_modules/  
\*/node\_modules/  
  
# Environment files  
.env  
.env.local  
.env.development.local  
.env.test.local  
.env.production.local  
  
# Build outputs  
dist/  
\*/dist/  
build/  
\*/build/  
  
# Database  
\*.db  
\*.sqlite  
  
# Logs  
logs  
\*.log  
npm-debug.log\*  
yarn-debug.log\*  
yarn-error.log\*  
  
# IDE  
.vscode/  
.idea/  
\*.swp  
\*.swo  
\*~  
  
# OS  
.DS\_Store  
Thumbs.db  
  
# Test outputs  
test-results/  
playwright-report/  
\*/test-results/  
\*/playwright-report/  
  
# Coverage  
coverage/  
\*/coverage/

# 2. Backend API with Prisma Setup

## 2.1 Initialize API Package

cd api  
npm init -y

## 2.2 Install Dependencies

# Production dependencies  
npm install express cors dotenv bcrypt jsonwebtoken @prisma/client  
  
# Development dependencies  
npm install -D @types/node @types/express @types/cors @types/bcrypt @types/jsonwebtoken typescript ts-node nodemon prisma eslint vitest @vitest/coverage-v8

## 2.3 Configure API Package.json

Update api/package.json:

{  
 "name": "api",  
 "version": "1.0.0",  
 "description": "Backend API for CI/CD demo",  
 "main": "dist/index.js",  
 "scripts": {  
 "dev": "nodemon src/index.ts",  
 "build": "tsc",  
 "start": "node dist/index.js",  
 "start:test": "NODE\_ENV=test PORT=5175 node dist/index.js",  
 "db:migrate:dev": "prisma migrate dev",  
 "db:migrate:deploy": "prisma migrate deploy",  
 "db:migrate:test": "DATABASE\_URL=\"postgresql://postgres:postgres@localhost:5432/testdb\" prisma migrate deploy",  
 "db:seed:dev": "SEED\_DISABLE=0 ts-node -r dotenv/config prisma/seed.ts",  
 "db:seed:off": "SEED\_DISABLE=1 ts-node -r dotenv/config prisma/seed.ts",  
 "db:reset": "prisma migrate reset --force",  
 "test": "vitest --run",  
 "test:watch": "vitest",  
 "test:coverage": "vitest --coverage",  
 "lint": "eslint src/\*\*/\*.ts",  
 "lint:fix": "eslint src/\*\*/\*.ts --fix"  
 },  
 "prisma": {  
 "seed": "ts-node -r dotenv/config prisma/seed.ts"  
 },  
 "keywords": ["express", "prisma", "typescript", "api"],  
 "author": "Your Name",  
 "license": "MIT",  
 "dependencies": {  
 "@prisma/client": "^5.22.0",  
 "bcrypt": "^5.1.1",  
 "cors": "^2.8.5",  
 "dotenv": "^16.3.1",  
 "express": "^4.18.2",  
 "jsonwebtoken": "^9.0.2"  
 },  
 "devDependencies": {  
 "@types/bcrypt": "^5.0.0",  
 "@types/cors": "^2.8.14",  
 "@types/express": "^4.17.18",  
 "@types/jsonwebtoken": "^9.0.3",  
 "@types/node": "^20.8.0",  
 "@vitest/coverage-v8": "^0.34.6",  
 "eslint": "^8.51.0",  
 "nodemon": "^3.0.1",  
 "prisma": "^5.22.0",  
 "ts-node": "^10.9.1",  
 "typescript": "^5.2.2",  
 "vitest": "^0.34.6"  
 }  
}

## 2.4 TypeScript Configuration

Create api/tsconfig.json:

{  
 "compilerOptions": {  
 "target": "ES2020",  
 "module": "commonjs",  
 "lib": ["ES2020"],  
 "outDir": "./dist",  
 "rootDir": "./src",  
 "strict": true,  
 "esModuleInterop": true,  
 "skipLibCheck": true,  
 "forceConsistentCasingInFileNames": true,  
 "declaration": true,  
 "declarationMap": true,  
 "sourceMap": true,  
 "removeComments": true,  
 "noImplicitAny": true,  
 "strictNullChecks": true,  
 "strictFunctionTypes": true,  
 "noImplicitThis": true,  
 "noImplicitReturns": true,  
 "noFallthroughCasesInSwitch": true,  
 "moduleResolution": "node",  
 "baseUrl": "./",  
 "paths": {  
 "@/\*": ["src/\*"]  
 },  
 "allowSyntheticDefaultImports": true,  
 "experimentalDecorators": true,  
 "emitDecoratorMetadata": true,  
 "resolveJsonModule": true,  
 "isolatedModules": true  
 },  
 "include": [  
 "src/\*\*/\*",  
 "prisma/\*\*/\*"  
 ],  
 "exclude": [  
 "node\_modules",  
 "dist"  
 ]  
}

## 2.5 Prisma Setup

Initialize Prisma:

npx prisma init

## 2.6 Database Schema

Update api/prisma/schema.prisma:

// This is your Prisma schema file,  
// learn more about it in the docs: https://pris.ly/d/prisma-schema  
  
generator client {  
 provider = "prisma-client-js"  
}  
  
datasource db {  
 provider = "postgresql"  
 url = env("DATABASE\_URL")  
}  
  
enum AuthProvider {  
 LOCAL  
 GOOGLE  
 GITHUB  
}  
  
model User {  
 id Int @id @default(autoincrement())  
 email String @unique  
 passwordHash String?  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
 profile UserProfile?  
 roles UserRole[]  
 accounts Account[]  
 assignedTasks Task[]  
  
 @@index([email])  
}  
  
model Role {  
 id Int @id @default(autoincrement())  
 name String @unique  
 users UserRole[]  
  
 @@index([name])  
}  
  
model UserRole {  
 userId Int  
 roleId Int  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 role Role @relation(fields: [roleId], references: [id], onDelete: Cascade)  
  
 @@id([userId, roleId])  
}  
  
model UserProfile {  
 id Int @id @default(autoincrement())  
 userId Int @unique  
 fullName String?  
 avatarUrl String?  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
}  
  
model Account {  
 id Int @id @default(autoincrement())  
 userId Int  
 provider AuthProvider  
 providerAccountId String  
 accessToken String?  
 refreshToken String?  
 expiresAt DateTime?  
 createdAt DateTime @default(now())  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
  
 @@unique([provider, providerAccountId])  
 @@index([userId])  
}  
  
model Task {  
 id Int @id @default(autoincrement())  
 title String  
 description String?  
 done Boolean @default(false)  
 priority String @default("medium")  
 dueDate DateTime?  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
 assignedToId Int?  
 assignedTo User? @relation(fields: [assignedToId], references: [id], onDelete: SetNull)  
  
 @@index([done, createdAt])  
 @@index([assignedToId])  
 @@index([priority])  
}

## 2.7 Environment Configuration

Create api/.env:

# Database  
DATABASE\_URL="postgresql://postgres:password@localhost:5432/cicd\_demo"  
  
# JWT - Access Token (short-lived)  
JWT\_SECRET="your-super-secret-jwt-key-change-this-in-production"  
  
# JWT - Refresh Token (long-lived)   
JWT\_REFRESH\_SECRET="your-super-secret-refresh-key-change-this-in-production"  
  
# Server  
PORT=5174  
NODE\_ENV="development"  
  
# CORS  
ALLOWED\_ORIGINS="http://localhost:3000,http://localhost:5173"  
  
# Seeding  
SEED\_DISABLE=0

Create api/.env.example:

# Database  
DATABASE\_URL="postgresql://postgres:password@localhost:5432/cicd\_demo"  
  
# JWT - Access Token (short-lived)  
JWT\_SECRET="your-super-secret-jwt-key-change-this-in-production"  
  
# JWT - Refresh Token (long-lived)  
JWT\_REFRESH\_SECRET="your-super-secret-refresh-key-change-this-in-production"  
  
# Server  
PORT=5174  
NODE\_ENV="development"  
  
# CORS  
ALLOWED\_ORIGINS="http://localhost:3000,http://localhost:5173"  
  
# Seeding  
SEED\_DISABLE=0

## 2.8 Database Seed Data

Create api/prisma/seed.ts:

import { PrismaClient } from '@prisma/client';  
import bcrypt from 'bcrypt';  
  
const prisma = new PrismaClient();  
  
async function main() {  
 if (process.env.SEED\_DISABLE === '1') {  
 console.log('Seeding disabled via SEED\_DISABLE=1');  
 return;  
 }  
  
 console.log('Starting database seeding...');  
  
 // Create roles  
 const adminRole = await prisma.role.upsert({  
 where: { name: 'admin' },  
 update: {},  
 create: { name: 'admin' }  
 });  
  
 const userRole = await prisma.role.upsert({  
 where: { name: 'user' },  
 update: {},  
 create: { name: 'user' }  
 });  
  
 console.log('Roles created:', { adminRole, userRole });  
  
 // Create admin user  
 const adminPasswordHash = await bcrypt.hash('admin123', 10);  
 const adminUser = await prisma.user.upsert({  
 where: { email: 'admin@example.com' },  
 update: {},  
 create: {  
 email: 'admin@example.com',  
 passwordHash: adminPasswordHash,  
 profile: {  
 create: {  
 fullName: 'Admin One'  
 }  
 },  
 roles: {  
 create: [  
 { roleId: adminRole.id },  
 { roleId: userRole.id }  
 ]  
 }  
 },  
 include: {  
 profile: true,  
 roles: { include: { role: true } }  
 }  
 });  
  
 // Create regular user  
 const userPasswordHash = await bcrypt.hash('user123', 10);  
 const regularUser = await prisma.user.upsert({  
 where: { email: 'user@example.com' },  
 update: {},  
 create: {  
 email: 'user@example.com',  
 passwordHash: userPasswordHash,  
 profile: {  
 create: {  
 fullName: 'Regular User'  
 }  
 },  
 roles: {  
 create: { roleId: userRole.id }  
 }  
 },  
 include: {  
 profile: true,  
 roles: { include: { role: true } }  
 }  
 });  
  
 console.log('Users created:', { adminUser, regularUser });  
  
 // Create sample tasks  
 const tasks = await Promise.all([  
 prisma.task.upsert({  
 where: { id: 1 },  
 update: {},  
 create: {  
 title: 'Setup CI/CD Pipeline',  
 description: 'Configure GitHub Actions for automated testing and deployment',  
 priority: 'high',  
 assignedToId: adminUser.id,  
 dueDate: new Date(Date.now() + 7 \* 24 \* 60 \* 60 \* 1000) // 7 days from now  
 }  
 }),  
 prisma.task.upsert({  
 where: { id: 2 },  
 update: {},  
 create: {  
 title: 'Write E2E Tests',  
 description: 'Implement comprehensive Playwright tests',  
 priority: 'high',  
 assignedToId: regularUser.id  
 }  
 }),  
 prisma.task.upsert({  
 where: { id: 3 },  
 update: {},  
 create: {  
 title: 'Code Review Process',  
 description: 'Establish code review guidelines and process',  
 priority: 'medium',  
 done: true  
 }  
 }),  
 prisma.task.upsert({  
 where: { id: 4 },  
 update: {},  
 create: {  
 title: 'Database Optimization',  
 description: 'Optimize database queries and add proper indexes',  
 priority: 'low'  
 }  
 }),  
 prisma.task.upsert({  
 where: { id: 5 },  
 update: {},  
 create: {  
 title: 'Security Audit',  
 description: 'Perform security audit and fix vulnerabilities',  
 priority: 'high',  
 assignedToId: adminUser.id  
 }  
 })  
 ]);  
  
 console.log('Tasks created:', tasks);  
  
 console.log('Database seeding completed successfully!');  
}  
  
main()  
 .catch((e) => {  
 console.error('Seeding failed:', e);  
 process.exit(1);  
 })  
 .finally(async () => {  
 await prisma.$disconnect();  
 });

## 2.9 Authentication Helper Functions

Create api/src/auth.ts:

import jwt from 'jsonwebtoken';  
import { Request, Response, NextFunction } from 'express';  
import { PrismaClient } from '@prisma/client';  
  
const prisma = new PrismaClient();  
  
interface AuthRequest extends Request {  
 user?: {  
 id: number;  
 email: string;  
 roles: string[];  
 };  
}  
  
interface TokenPayload {  
 userId: number;  
 email: string;  
}  
  
// Generate access token (short-lived)  
export const generateAccessToken = (payload: TokenPayload): string => {  
 return jwt.sign(payload, process.env.JWT\_SECRET!, { expiresIn: '15m' });  
};  
  
// Generate refresh token (long-lived)  
export const generateRefreshToken = (payload: TokenPayload): string => {  
 return jwt.sign(payload, process.env.JWT\_REFRESH\_SECRET!, { expiresIn: '7d' });  
};  
  
// Verify refresh token  
export const verifyRefreshToken = (token: string): TokenPayload => {  
 return jwt.verify(token, process.env.JWT\_REFRESH\_SECRET!) as TokenPayload;  
};  
  
// Authentication middleware  
export const authMiddleware = async (req: AuthRequest, res: Response, next: NextFunction) => {  
 try {  
 const authHeader = req.headers.authorization;  
   
 if (!authHeader) {  
 return res.status(401).json({ error: 'No authorization header provided' });  
 }  
  
 const token = authHeader.split(' ')[1]; // Bearer <token>  
   
 if (!token) {  
 return res.status(401).json({ error: 'No token provided' });  
 }  
  
 const decoded = jwt.verify(token, process.env.JWT\_SECRET!) as TokenPayload;  
   
 const user = await prisma.user.findUnique({  
 where: { id: decoded.userId },  
 include: {  
 roles: {  
 include: {  
 role: true  
 }  
 }  
 }  
 });  
  
 if (!user) {  
 return res.status(401).json({ error: 'Invalid token' });  
 }  
  
 req.user = {  
 id: user.id,  
 email: user.email,  
 roles: user.roles.map(ur => ur.role.name)  
 };  
  
 next();  
 } catch (error) {  
 return res.status(401).json({ error: 'Invalid or expired token' });  
 }  
};

## 2.10 Complete API Implementation

Create the main server file api/src/index.ts:

import express from "express";  
import cors from "cors";  
import { PrismaClient } from "@prisma/client";  
import "dotenv/config";  
import bcrypt from "bcrypt";  
import {   
 generateAccessToken,   
 generateRefreshToken,   
 verifyRefreshToken,  
 authMiddleware   
} from "./auth";  
  
const app = express();  
app.use(cors());  
app.use(express.json());  
const prisma = new PrismaClient();  
  
// Health check endpoint with database connectivity test  
app.get("/api/health", async (\_, res) => {  
 await prisma.$queryRaw`SELECT 1`;  
 res.json({ ok: true, ts: new Date().toISOString() });  
});  
  
// Registration endpoint with advanced features  
app.post("/api/register", async (req, res) => {  
 const { email, password, username } = req.body;  
   
 console.log('Registration attempt:', { email, username, hasPassword: !!password });  
   
 if (!email || !username) {  
 return res.status(400).json({ error: "Email and username are required" });  
 }  
  
 try {  
 // Check if user already exists  
 const existingUser = await prisma.user.findUnique({  
 where: { email }  
 });  
  
 if (existingUser) {  
 console.log('User already exists:', email);  
 return res.status(400).json({ error: "User with this email already exists" });  
 }  
  
 // Hash password if provided  
 let passwordHash = null;  
 if (password) {  
 passwordHash = await bcrypt.hash(password, 10);  
 console.log('Password hashed successfully');  
 }  
  
 // Ensure "user" role exists (dynamic role creation)  
 let userRole = await prisma.role.findUnique({  
 where: { name: "user" }  
 });  
  
 if (!userRole) {  
 console.log('Creating user role...');  
 userRole = await prisma.role.create({  
 data: { name: "user" }  
 });  
 }  
  
 console.log('User role found/created:', userRole);  
  
 // Create user with profile and role in a transaction  
 const newUser = await prisma.$transaction(async (tx) => {  
 console.log('Creating user in transaction...');  
 const user = await tx.user.create({  
 data: {  
 email,  
 passwordHash,  
 profile: {  
 create: {  
 fullName: username  
 }  
 },  
 roles: {  
 create: {  
 roleId: userRole.id  
 }  
 }  
 },  
 include: {  
 profile: true,  
 roles: {  
 include: {  
 role: true  
 }  
 }  
 }  
 });  
  
 console.log('User created successfully:', { id: user.id, email: user.email, roles: user.roles.length });  
 return user;  
 });  
  
 // Generate JWT tokens (access + refresh)  
 const tokenPayload = { userId: newUser.id, email: newUser.email };  
 const accessToken = generateAccessToken(tokenPayload);  
 const refreshToken = generateRefreshToken(tokenPayload);  
  
 console.log('Registration completed successfully for:', email);  
  
 // Return user data with tokens  
 res.status(201).json({  
 id: newUser.id,  
 email: newUser.email,  
 username: newUser.profile?.fullName || username,  
 fullName: newUser.profile?.fullName,  
 roles: newUser.roles.map(r => r.role.name),  
 accessToken,  
 refreshToken  
 });  
 } catch (error) {  
 console.error('Registration error:', error);  
 res.status(500).json({ error: "Internal server error" });  
 }  
});  
  
// Login endpoint with JWT tokens  
app.post("/api/login", async (req, res) => {  
 const { email } = req.body;  
   
 if (!email) {  
 return res.status(400).json({ error: "Email is required" });  
 }  
  
 try {  
 const user = await prisma.user.findUnique({  
 where: { email },  
 include: {  
 profile: true,  
 roles: {  
 include: {  
 role: true  
 }  
 }  
 }  
 });  
  
 if (!user) {  
 return res.status(401).json({ error: "User not found" });  
 }  
  
 // Generate JWT tokens  
 const tokenPayload = { userId: user.id, email: user.email };  
 const accessToken = generateAccessToken(tokenPayload);  
 const refreshToken = generateRefreshToken(tokenPayload);  
  
 // Return user data with tokens  
 res.json({  
 id: user.id,  
 email: user.email,  
 username: user.profile?.fullName || email.split('@')[0],  
 fullName: user.profile?.fullName,  
 roles: user.roles.map(r => r.role.name),  
 accessToken,  
 refreshToken  
 });  
 } catch (error) {  
 console.error('Login error:', error);  
 res.status(500).json({ error: "Internal server error" });  
 }  
});  
  
// Refresh token endpoint for token renewal  
app.post("/api/refresh", async (req, res) => {  
 const { refreshToken } = req.body;  
  
 if (!refreshToken) {  
 return res.status(400).json({ error: "Refresh token is required" });  
 }  
  
 try {  
 const payload = verifyRefreshToken(refreshToken);  
   
 // Generate new access token  
 const newAccessToken = generateAccessToken({  
 userId: payload.userId,  
 email: payload.email  
 });  
  
 res.json({ accessToken: newAccessToken });  
 } catch (error) {  
 return res.status(401).json({ error: "Invalid or expired refresh token" });  
 }  
});  
  
// Protected endpoints - require authentication  
  
// Get tasks with pagination support  
app.get("/api/tasks", authMiddleware, async (req, res) => {  
 const page = parseInt(req.query.page as string) || 1;  
 const limit = parseInt(req.query.limit as string) || 10;  
 const skip = (page - 1) \* limit;  
  
 // Get total count for pagination metadata  
 const totalTasks = await prisma.task.count();  
  
 const tasks = await prisma.task.findMany({  
 include: {  
 assignedTo: {  
 include: {  
 profile: true  
 }  
 }  
 },  
 orderBy: {  
 createdAt: 'desc'  
 },  
 skip: skip,  
 take: limit  
 });  
  
 const totalPages = Math.ceil(totalTasks / limit);  
  
 res.json({  
 tasks,  
 pagination: {  
 currentPage: page,  
 totalPages,  
 totalItems: totalTasks,  
 itemsPerPage: limit,  
 hasNextPage: page < totalPages,  
 hasPreviousPage: page > 1  
 }  
 });  
});  
  
// Create new task  
app.post("/api/tasks", authMiddleware, async (req, res) => {  
 const { title, assignedToId } = req.body;  
 const task = await prisma.task.create({   
 data: {   
 title,  
 assignedToId: assignedToId ? parseInt(assignedToId) : null  
 },  
 include: {  
 assignedTo: {  
 include: {  
 profile: true  
 }  
 }  
 }  
 });  
 res.status(201).json(task);  
});  
  
// Update task  
app.put("/api/tasks/:id", authMiddleware, async (req, res) => {  
 const { id } = req.params;  
 const { title, assignedToId, done } = req.body;  
   
 try {  
 const task = await prisma.task.update({  
 where: { id: parseInt(id) },  
 data: {  
 title: title || undefined,  
 assignedToId: assignedToId !== undefined ? (assignedToId ? parseInt(assignedToId) : null) : undefined,  
 done: done !== undefined ? done : undefined  
 },  
 include: {  
 assignedTo: {  
 include: {  
 profile: true  
 }  
 }  
 }  
 });  
 res.json(task);  
 } catch (error) {  
 console.error('Update task error:', error);  
 res.status(404).json({ error: "Task not found" });  
 }  
});  
  
// Delete task  
app.delete("/api/tasks/:id", authMiddleware, async (req, res) => {  
 const { id } = req.params;  
   
 try {  
 await prisma.task.delete({  
 where: { id: parseInt(id) }  
 });  
 res.status(204).send();  
 } catch (error) {  
 console.error('Delete task error:', error);  
 res.status(404).json({ error: "Task not found" });  
 }  
});  
  
// Get users with pagination support  
app.get("/api/users", authMiddleware, async (req, res) => {  
 const page = parseInt(req.query.page as string) || 1;  
 const limit = parseInt(req.query.limit as string) || 10;  
 const skip = (page - 1) \* limit;  
  
 // Get total count for pagination metadata  
 const totalUsers = await prisma.user.count();  
  
 const users = await prisma.user.findMany({  
 include: {  
 profile: true,  
 roles: {  
 include: {  
 role: true  
 }  
 }  
 },  
 skip: skip,  
 take: limit,  
 orderBy: {  
 createdAt: 'asc'  
 }  
 });  
  
 const totalPages = Math.ceil(totalUsers / limit);  
  
 res.json({  
 users,  
 pagination: {  
 currentPage: page,  
 totalPages,  
 totalItems: totalUsers,  
 itemsPerPage: limit,  
 hasNextPage: page < totalPages,  
 hasPreviousPage: page > 1  
 }  
 });  
});  
  
const PORT = Number(process.env.PORT || 5175);  
app.listen(PORT, () => console.log(`API on :${PORT}`));

# 3. Frontend React Application

## 3.1 Initialize React Project

cd ../web  
npm init -y

## 3.2 Install React Dependencies

# Production dependencies  
npm install react react-dom react-router-dom  
  
# Development dependencies  
npm install -D @types/react @types/react-dom @vitejs/plugin-react vite typescript tailwindcss postcss autoprefixer @types/node eslint globals typescript-eslint eslint-plugin-react-hooks eslint-plugin-react-refresh @playwright/test dotenv-cli

## 3.3 Configure Web Package.json

Update web/package.json:

{  
 "name": "web",  
 "private": true,  
 "version": "0.0.0",  
 "type": "module",  
 "scripts": {  
 "dev": "vite --mode development",  
 "build": "tsc -b && vite build",  
 "lint": "eslint .",  
 "preview": "vite preview --port 5173",  
 "e2e": "dotenv -e .env.test -- playwright test",  
 "e2e:dev": "dotenv -e .env.development -- playwright test",  
 "e2e:report": "playwright show-report --host=0.0.0.0 --port=9323"  
 },  
 "dependencies": {  
 "react": "^18.2.0",  
 "react-dom": "^18.2.0",  
 "react-router-dom": "^6.15.0"  
 },  
 "devDependencies": {  
 "@eslint/js": "^9.9.0",  
 "@playwright/test": "^1.40.0",  
 "@types/node": "^20.8.0",  
 "@types/react": "^18.2.27",  
 "@types/react-dom": "^18.2.12",  
 "@vitejs/plugin-react": "^4.1.0",  
 "autoprefixer": "^10.4.16",  
 "dotenv-cli": "^7.3.0",  
 "eslint": "^8.52.0",  
 "eslint-plugin-react-hooks": "^4.6.0",  
 "eslint-plugin-react-refresh": "^0.4.4",  
 "globals": "^13.23.0",  
 "postcss": "^8.4.31",  
 "tailwindcss": "^3.3.5",  
 "typescript": "^5.2.2",  
 "typescript-eslint": "^6.9.0",  
 "vite": "^4.5.0"  
 }  
}

## 3.4 Vite Configuration

Create web/vite.config.ts:

import { defineConfig } from 'vite';  
import react from '@vitejs/plugin-react';  
  
export default defineConfig({  
 plugins: [react()],  
 server: {  
 port: 5173,  
 host: true,  
 proxy: {  
 '/api': {  
 target: 'http://localhost:5174',  
 changeOrigin: true  
 }  
 }  
 },  
 build: {  
 outDir: 'dist',  
 sourcemap: true  
 },  
 resolve: {  
 alias: {  
 '@': '/src'  
 }  
 }  
});

## 3.5 TypeScript Configuration

Create web/tsconfig.json:

{  
 "compilerOptions": {  
 "target": "ES2020",  
 "useDefineForClassFields": true,  
 "lib": ["ES2020", "DOM", "DOM.Iterable"],  
 "module": "ESNext",  
 "skipLibCheck": true,  
 "moduleResolution": "bundler",  
 "allowImportingTsExtensions": true,  
 "resolveJsonModule": true,  
 "isolatedModules": true,  
 "noEmit": true,  
 "jsx": "react-jsx",  
 "strict": true,  
 "noUnusedLocals": true,  
 "noUnusedParameters": true,  
 "noFallthroughCasesInSwitch": true,  
 "baseUrl": ".",  
 "paths": {  
 "@/\*": ["src/\*"]  
 }  
 },  
 "include": ["src"],  
 "references": [{ "path": "./tsconfig.node.json" }]  
}

Create web/tsconfig.node.json:

{  
 "compilerOptions": {  
 "composite": true,  
 "skipLibCheck": true,  
 "module": "ESNext",  
 "moduleResolution": "bundler",  
 "allowSyntheticDefaultImports": true,  
 "strict": true  
 },  
 "include": ["vite.config.ts"]  
}

## 3.6 Tailwind CSS Setup

npx tailwindcss init -p

Update web/tailwind.config.js:

/\*\* @type {import('tailwindcss').Config} \*/  
export default {  
 content: [  
 "./index.html",  
 "./src/\*\*/\*.{js,ts,jsx,tsx}",  
 ],  
 theme: {  
 extend: {},  
 },  
 plugins: [],  
}

## 3.7 Create Basic HTML Template

Create web/index.html:

<!doctype html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <link rel="icon" type="image/svg+xml" href="/vite.svg" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>CI/CD Demo</title>  
 </head>  
 <body>  
 <div id="root"></div>  
 <script type="module" src="/src/main.tsx"></script>  
 </body>  
</html>

## 3.8 Environment Configuration

Create web/.env.development:

VITE\_API\_URL=http://localhost:5174/api

Create web/.env.test:

VITE\_API\_URL=http://localhost:5175/api

Create web/.env.example:

VITE\_API\_URL=http://localhost:5174/api

# 4. Playwright E2E Testing

## 4.1 Initialize Playwright

cd web  
npx playwright install

## 4.2 Playwright Configuration

Create web/playwright.config.ts:

/// <reference types="node" />  
import { defineConfig, devices } from '@playwright/test';  
  
export default defineConfig({  
 testDir: './tests/e2e',  
 fullyParallel: true,  
 forbidOnly: !!process.env.CI,  
 retries: process.env.CI ? 2 : 0,  
 workers: process.env.CI ? 1 : undefined,  
 reporter: 'html',  
 use: {  
 baseURL: 'http://localhost:5173',  
 headless: true,  
 trace: 'on-first-retry',  
 },  
 projects: [  
 {  
 name: 'chromium',  
 use: { ...devices['Desktop Chrome'] },  
 },  
 {  
 name: 'firefox',  
 use: { ...devices['Desktop Firefox'] },  
 },  
 {  
 name: 'webkit',  
 use: { ...devices['Desktop Safari'] },  
 }  
 ],  
 webServer: {  
 command: 'npm run dev',  
 url: 'http://localhost:5173',  
 reuseExistingServer: !process.env.CI,  
 },  
});

## 4.3 E2E Test Examples

Create web/tests/e2e/auth.spec.ts:

import { test, expect } from '@playwright/test';  
  
test.describe('Authentication Flow', () => {  
 test('should display login page', async ({ page }) => {  
 await page.goto('/login');  
   
 await expect(page).toHaveTitle(/CI\/CD Demo/);  
 await expect(page.locator('h1')).toContainText('Welcome Back');  
 await expect(page.locator('input[name="email"]')).toBeVisible();  
 await expect(page.locator('button[type="submit"]')).toContainText('Sign In');  
 });  
  
 test('should navigate to register page', async ({ page }) => {  
 await page.goto('/login');  
   
 await page.locator('text=Sign up here').click();  
 await expect(page).toHaveURL('/register');  
 await expect(page.locator('h1')).toContainText('Create Account');  
 });  
  
 test('should successfully login with admin@example.com', async ({ page }) => {  
 await page.goto('/login');  
   
 await page.fill('input[name="email"]', 'admin@example.com');  
 await page.locator('button[type="submit"]').click();  
   
 await expect(page).toHaveURL('/');  
 await expect(page.locator('h2')).toContainText('Tasks');  
 await expect(page.locator('text=Welcome,')).toBeVisible();  
 });  
  
 test('should successfully register new user', async ({ page }) => {  
 const uniqueEmail = `test-${Date.now()}@example.com`;  
   
 await page.goto('/register');  
   
 await page.fill('input[name="email"]', uniqueEmail);  
 await page.fill('input[name="username"]', 'Test User');  
 await page.fill('input[name="password"]', 'testpassword123');  
 await page.fill('input[name="confirmPassword"]', 'testpassword123');  
   
 await page.locator('button[type="submit"]').click();  
   
 await expect(page).toHaveURL('/');  
 await expect(page.locator('h2')).toContainText('Tasks');  
 await expect(page.locator('text=Welcome, Test User')).toBeVisible();  
 });  
});

# 5. GitHub Actions CI/CD Pipeline

## 5.1 Basic CI/CD Workflow

Create .github/workflows/ci-cd.yml:

name: CI/CD Pipeline  
  
on:  
 push:  
 branches: [ main, develop ]  
 pull\_request:  
 branches: [ main ]  
  
env:  
 NODE\_VERSION: '18'  
 POSTGRES\_VERSION: '15'  
  
jobs:  
 lint-and-test:  
 runs-on: ubuntu-latest  
   
 services:  
 postgres:  
 image: postgres:15  
 env:  
 POSTGRES\_PASSWORD: postgres  
 POSTGRES\_DB: testdb  
 options: >-  
 --health-cmd pg\_isready  
 --health-interval 10s  
 --health-timeout 5s  
 --health-retries 5  
 ports:  
 - 5432:5432  
  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v4  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v4  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
 cache: 'npm'  
  
 - name: Install dependencies  
 run: npm ci  
  
 - name: Lint code  
 run: npm run lint  
  
 - name: Setup database  
 env:  
 DATABASE\_URL: postgresql://postgres:postgres@localhost:5432/testdb  
 run: |  
 cd api  
 npx prisma migrate deploy  
 npm run db:seed:dev  
  
 - name: Run API tests  
 env:  
 DATABASE\_URL: postgresql://postgres:postgres@localhost:5432/testdb  
 JWT\_SECRET: test-secret-key  
 run: npm run test --workspace=api  
  
 - name: Build applications  
 run: npm run build  
  
 - name: Install Playwright browsers  
 run: npx playwright install --with-deps --workspace=web  
  
 - name: Start test server  
 env:  
 DATABASE\_URL: postgresql://postgres:postgres@localhost:5432/testdb  
 JWT\_SECRET: test-secret-key  
 run: |  
 npm run start:test --workspace=api &  
 sleep 5  
  
 - name: Run E2E tests  
 run: npm run e2e --workspace=web  
  
 - name: Upload Playwright report  
 uses: actions/upload-artifact@v3  
 if: always()  
 with:  
 name: playwright-report  
 path: web/playwright-report/  
 retention-days: 30  
  
 deploy:  
 needs: lint-and-test  
 runs-on: ubuntu-latest  
 if: github.ref == 'refs/heads/main'  
   
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v4  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v4  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
 cache: 'npm'  
  
 - name: Install dependencies  
 run: npm ci  
  
 - name: Build applications  
 run: npm run build  
  
 - name: Deploy to staging  
 run: |  
 echo "Deploying to staging environment..."  
 # Add your deployment commands here

## 5.2 Running the Complete Setup

Follow these steps to get everything running:

### Prerequisites

1. **PostgreSQL Database**: Install and create a database named cicd\_demo
2. **Node.js 18+**: Required for all applications
3. **Git**: For version control

### Setup Commands

# 1. Clone/Create the project  
git init  
git add .  
git commit -m "Initial commit"  
  
# 2. Setup database  
cd api  
npm run db:migrate:dev  
npm run db:seed:dev  
  
# 3. Start backend API  
npm run dev  
  
# 4. In another terminal, start frontend  
cd ../web  
npm run dev  
  
# 5. Run E2E tests (in another terminal)  
cd web  
npm run e2e

### Accessing the Application

* **Frontend**: http://localhost:5173
* **Backend API**: http://localhost:5174
* **API Health Check**: http://localhost:5174/health

### Test Accounts

* **Admin**: admin@example.com (no password required)
* **User**: user@example.com (password: user123)

## 🎉 Conclusion

You now have a complete full-stack application with:

✅ **Monorepo structure** with workspace management  
✅ **Backend API** with Express, Prisma, and PostgreSQL  
✅ **JWT Authentication** with role-based access control  
✅ **React Frontend** with TypeScript and Tailwind CSS  
✅ **E2E Testing** with Playwright across multiple browsers  
✅ **CI/CD Pipeline** with GitHub Actions  
✅ **Production-ready** error handling and logging

This tutorial provides a solid foundation for building modern full-stack applications with automated testing and deployment pipelines.