// 1. Performance Optimization Implementation

// API Rate Limiting

import rateLimit from 'express-rate-limit';

import RedisStore from 'rate-limit-redis';

const apiLimiter = rateLimit({

store: new RedisStore({

client: redisClient,

prefix: 'rate\_limit:'

}),

windowMs: 15 \* 60 \* 1000, // 15 minutes

max: 100 // limit each IP to 100 requests per windowMs

});

// Caching Implementation

const cacheMiddleware = async (req, res, next) => {

const key = `cache:${req.url}`;

const cached = await redis.get(key);

if (cached) {

return res.json(JSON.parse(cached));

}

res.sendResponse = res.json;

res.json = (body) => {

redis.set(key, JSON.stringify(body), 'EX', 300); // 5 min cache

res.sendResponse(body);

};

next();

};

// 2. Security Implementation

// CSRF Protection

import { csrf } from '@vercel/edge';

export const config = {

matcher: '/api/:path\*',

};

export default function middleware(req) {

const token = csrf.generateToken();

return new Response(null, {

headers: {

'X-CSRF-Token': token,

},

});

}

// Content Security Policy

const csp = {

'default-src': ["'self'"],

'script-src': ["'self'", "'unsafe-inline'", 'https://cdnjs.cloudflare.com'],

'style-src': ["'self'", "'unsafe-inline'"],

'img-src': ["'self'", 'data:', 'https:'],

'connect-src': ["'self'", 'https://api.linkedin.com'],

};

// 3. Deployment Scripts

// Pre-deployment checks

async function runPreDeploymentChecks() {

await Promise.all([

checkDependencies(),

runSecurityScans(),

validateEnvVars(),

checkDatabaseMigrations(),

]);

}

// Database migration

async function runMigrations() {

const { data, error } = await supabase.rpc('run\_migrations', {

dryRun: process.env.NODE\_ENV !== 'production',

});

if (error) {

throw new Error(`Migration failed: ${error.message}`);

}

}

// Cache warming

async function warmCache() {

const routes = await getStaticRoutes();

await Promise.all(

routes.map(route =>

fetch(`${process.env.API\_URL}${route}`)

)

);

}