# **Feature Branching & CI/CD Documentation**

## **Overview**

This document outlines the feature branching strategy and CI/CD pipeline for the Chat Application project. It provides guidelines for development workflow, automated testing, deployment processes, and best practices for maintaining code quality and delivery efficiency.

## **Branching Strategy**

### **Git Flow Model**

We use a modified Git Flow strategy optimized for continuous deployment:

main (production)  
├── develop (staging)  
├── feature/feature-name  
├── hotfix/hotfix-name  
└── release/version-number

### **Branch Types**

#### **Main Branches**

**main** - Production Branch

* Contains production-ready code
* Protected branch with strict rules
* Automatically deployed to production
* Only accepts merges from develop or hotfix/\*
* All commits must be signed and reviewed

**develop** - Integration Branch

* Contains latest development changes
* Staging environment deployment target
* Integration point for all features
* Protected branch requiring pull request reviews

#### **Supporting Branches**

**feature/\*** - Feature Branches

* Created from develop branch
* Naming convention: feature/JIRA-123-short-description
* Merged back to develop via Pull Request
* Deleted after successful merge

**hotfix/\*** - Hotfix Branches

* Created from main for urgent production fixes
* Naming convention: hotfix/JIRA-456-critical-fix
* Merged to both main and develop
* Triggers immediate production deployment

**release/\*** - Release Branches

* Created from develop for release preparation
* Naming convention: release/v1.2.0
* Minor bug fixes and version bumps only
* Merged to main and back-merged to develop

### **Branch Protection Rules**

#### **Main Branch Protection**

Required status checks:  
 - CI/CD Pipeline  
 - Code Quality Gates  
 - Security Scan  
 - All Tests Pass  
   
Required reviews: 2  
Dismiss stale reviews: true  
Require review from code owners: true  
Restrict pushes to admins: false  
Require signed commits: true

#### **Develop Branch Protection**

Required status checks:  
 - CI/CD Pipeline  
 - Code Quality Gates  
 - Integration Tests  
   
Required reviews: 1  
Dismiss stale reviews: true  
Require review from code owners: false  
Allow force pushes: false

## **Feature Development Workflow**

### **1. Feature Planning**

* Create JIRA ticket with detailed requirements
* Estimate effort and assign to sprint
* Define acceptance criteria
* Plan testing approach

### **2. Branch Creation**

# Start from latest develop  
git checkout develop  
git pull origin develop  
  
# Create feature branch  
git checkout -b feature/CHAT-123-add-message-history  
  
# Push initial branch  
git push -u origin feature/CHAT-123-add-message-history

### **3. Development Process**

# Regular commits with meaningful messages  
git add .  
git commit -m "feat(chat): add message history storage functionality  
  
- Implement local storage for chat messages  
- Add message persistence across sessions  
- Update ChatInterface component  
- Add unit tests for storage utilities  
  
Closes CHAT-123"  
  
# Push changes regularly  
git push origin feature/CHAT-123-add-message-history

### **4. Pre-Pull Request Checklist**

* All tests pass locally
* Code follows coding standards
* Documentation updated
* No console.log or debug code
* TypeScript types defined
* Security considerations addressed

### **5. Pull Request Creation**

Pull Request Template  
  
Description  
Brief description of changes and motivation.  
  
 Type of Change  
- Bug fix (non-breaking change)  
- New feature (non-breaking change)  
- Breaking change (fix or feature causing existing functionality to change)  
- Documentation update  
  
How Has This Been Tested?  
- Unit tests  
- Integration tests  
- Manual testing  
- Cross-browser testing  
  
Checklist  
- Code follows coding standards  
- Self-review completed  
- Comments added for complex logic  
- Documentation updated  
- Tests added/updated  
- No breaking changes or breaking changes documented  
  
 **6. Code Review Process**

* Automated checks must pass
* Peer review by team member
* Code owner approval (if required)
* Address feedback promptly
* Maintain professional communication

### **7. Merge and Cleanup**

After approval, merge via GitHub UI  
Delete feature branch locally  
git checkout develop  
git pull origin develop  
git branch -d feature/CHAT-123-add-message-history

## **CI/CD Pipeline Architecture**

### **Pipeline Overview**

graph LR  
 A[Push to Feature] --> B[CI Pipeline]  
 B --> C[Build & Test]  
 C --> D[Code Quality]  
 D --> E[Security Scan]  
 E --> F[Deploy to Preview]  
   
 G[Merge to Develop] --> H[CI Pipeline]  
 H --> I[Build & Test]  
 I --> J[Integration Tests]  
 J --> K[Deploy to Staging]  
   
 L[Merge to Main] --> M[CD Pipeline]  
 M --> N[Build & Test]  
 N --> O[Production Deploy]  
 O --> P[Health Check]  
 P --> Q[Notification]

### **Pipeline Stages**

#### **1. Code Quality Stage**

* ESLint checks
* Prettier formatting validation
* TypeScript compilation
* Dependency vulnerability scan

#### **2. Testing Stage**

* Unit tests with Jest
* Component tests with React Testing Library
* Integration tests
* E2E tests (on staging)

#### **3. Build Stage**

* Next.js application build
* Static asset optimization
* Bundle size analysis
* Build artifact storage

#### **4. Security Stage**

* SAST (Static Application Security Testing)
* Dependency security audit
* Secret scanning
* License compliance check

#### **5. Deployment Stage**

* Environment-specific configuration
* Database migrations (if applicable)
* Health checks
* Rollback capability

## **GitHub Actions Configuration**

### **Main CI/CD Workflow**

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

name: CI/CD Pipeline  
  
on:  
 push:  
 branches: [ main, develop ]  
 pull\_request:  
 branches: [ main, develop ]  
  
env:  
 NODE\_VERSION: '18'  
 AZURE\_CLIENT\_ID: ${{ secrets.AZURE\_CLIENT\_ID }}  
 AZURE\_TENANT\_ID: ${{ secrets.AZURE\_TENANT\_ID }}  
  
jobs:  
 # Code Quality and Testing  
 quality-check:  
 runs-on: ubuntu-latest  
 outputs:  
 version: ${{ steps.version.outputs.version }}  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
 with:  
 fetch-depth: 0  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v4  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
 cache: 'npm'  
  
 - name: Install Dependencies  
 run: npm ci  
  
 - name: Type Check  
 run: npm run type-check  
  
 - name: Lint Check  
 run: npm run lint:strict  
  
 - name: Format Check  
 run: npm run format:check  
  
 - name: Run Unit Tests  
 run: npm run test:unit  
 env:  
 CI: true  
  
 - name: Run Integration Tests  
 run: npm run test:integration  
 env:  
 CI: true  
  
 - name: Generate Version  
 id: version  
 run: |  
 if [[ ${{ github.ref }} == 'refs/heads/main' ]]; then  
 VERSION="v$(date +'%Y.%m.%d')-${{ github.run\_number }}"  
 else  
 VERSION="preview-${{ github.run\_number }}"  
 fi  
 echo "version=$VERSION" >> $GITHUB\_OUTPUT  
  
 - name: Upload Test Coverage  
 uses: codecov/codecov-action@v3  
 with:  
 token: ${{ secrets.CODECOV\_TOKEN }}  
 files: ./coverage/lcov.info  
  
 # Security Scanning  
 security-scan:  
 runs-on: ubuntu-latest  
 needs: quality-check  
   
 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: Run Security Audit  
 run: npm audit --audit-level=high  
  
 - name: Run Snyk Security Scan  
 uses: snyk/actions/node@master  
 env:  
 SNYK\_TOKEN: ${{ secrets.SNYK\_TOKEN }}  
 with:  
 args: --severity-threshold=high  
  
 - name: Upload Snyk Results  
 uses: github/codeql-action/upload-sarif@v2  
 if: always()  
 with:  
 sarif\_file: snyk.sarif  
  
 # Build Application  
 build:  
 runs-on: ubuntu-latest  
 needs: [quality-check, security-scan]  
   
 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 Application  
 run: npm run build  
 env:  
 NEXT\_PUBLIC\_VERSION: ${{ needs.quality-check.outputs.version }}  
  
 - name: Analyze Bundle Size  
 run: npx @next/bundle-analyzer  
  
 - name: Upload Build Artifacts  
 uses: actions/upload-artifact@v4  
 with:  
 name: build-files-${{ needs.quality-check.outputs.version }}  
 path: |  
 .next/  
 out/  
 public/  
 retention-days: 30  
  
 # Deploy to Preview (Feature Branches)  
 deploy-preview:  
 runs-on: ubuntu-latest  
 needs: [quality-check, build]  
 if: github.event\_name == 'pull\_request'  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
  
 - name: Download Build Artifacts  
 uses: actions/download-artifact@v4  
 with:  
 name: build-files-${{ needs.quality-check.outputs.version }}  
  
 - name: Deploy to Vercel Preview  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID }}  
 working-directory: ./  
  
 - name: Comment PR with Preview URL  
 uses: actions/github-script@v7  
 with:  
 script: |  
 github.rest.issues.createComment({  
 issue\_number: context.issue.number,  
 owner: context.repo.owner,  
 repo: context.repo.repo,  
 body: ‘ Preview deployment ready! Check it out at: https://chat-app-preview-${{ github.event.number }}.vercel.app'  
 })  
  
 # Deploy to Staging (Develop Branch)  
 deploy-staging:  
 runs-on: ubuntu-latest  
 needs: [quality-check, build]  
 if: github.ref == 'refs/heads/develop'  
 environment: staging  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
  
 - name: Download Build Artifacts  
 uses: actions/download-artifact@v4  
 with:  
 name: build-files-${{ needs.quality-check.outputs.version }}  
  
 - name: Deploy to Staging  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID }}  
 vercel-args: '--prod --env staging'  
  
 - name: Run E2E Tests  
 run: npm run test:e2e  
 env:  
 E2E\_BASE\_URL: <https://chat-app-staging.vercel.app> - name: Notify Slack  
 uses: 8398a7/action-slack@v3  
 with:  
 status: ${{ job.status }}  
 channel: '#deployments'  
 webhook\_url: ${{ secrets.SLACK\_WEBHOOK }}  
  
 # Deploy to Production (Main Branch)  
 deploy-production:  
 runs-on: ubuntu-latest  
 needs: [quality-check, build]  
 if: github.ref == 'refs/heads/main'  
 environment: production  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
  
 - name: Download Build Artifacts  
 uses: actions/download-artifact@v4  
 with:  
 name: build-files-${{ needs.quality-check.outputs.version }}  
  
 - name: Deploy to Production  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID }}  
 vercel-args: '--prod'  
  
 - name: Health Check  
 run: |  
 curl -f <https://chat-app-prod.vercel.app/api/health> || exit 1  
  
 - name: Create GitHub Release  
 uses: actions/create-release@v1  
 env:  
 GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}  
 with:  
 tag\_name: ${{ needs.quality-check.outputs.version }}  
 release\_name: Release ${{ needs.quality-check.outputs.version }}  
 body: |  
 Changes in this Release:  
 ${{ github.event.head\_commit.message }}  
 draft: false  
 prerelease: false  
  
 - name: Notify Teams  
 uses: 8398a7/action-slack@v3  
 with:  
 status: ${{ job.status }}  
 channel: '#general'  
 webhook\_url: ${{ secrets.SLACK\_WEBHOOK }}  
 text: ' Production deployment successful! Version: ${{ needs.quality-check.outputs.version }}'

### **Feature Branch Workflow**

Create .github/workflows/feature-branch.yml:

name: Feature Branch CI  
  
on:  
 push:  
 branches-ignore: [ main, develop ]  
  
jobs:  
 validate:  
 runs-on: ubuntu-latest  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v4  
 with:  
 node-version: '18'  
 cache: 'npm'  
  
 - name: Install Dependencies  
 run: npm ci  
  
 - name: Type Check  
 run: npm run type-check  
  
 - name: Lint Check  
 run: npm run lint  
  
 - name: Format Check  
 run: npm run format:check  
  
 - name: Run Tests  
 run: npm test  
 env:  
 CI: true  
  
 - name: Build Check  
 run: npm run build  
  
 - name: Comment PR Status  
 if: github.event\_name == 'pull\_request'  
 uses: actions/github-script@v7  
 with:  
 script: |  
 const { data: comments } = await github.rest.issues.listComments({  
 owner: context.repo.owner,  
 repo: context.repo.repo,  
 issue\_number: context.issue.number,  
 });  
   
 const botComment = comments.find(comment =>   
 comment.user.type === 'Bot' &&   
 comment.body.includes('## CI Status')  
 );  
   
 const body = `## CI Status  
   
 All checks passed successfully!  
   
 - TypeScript compilation  
 - ESLint checks  
 - Code formatting  
 - Unit tests  
 - Build successful  
   
 Ready for review! ;  
   
 if (botComment) {  
 github.rest.issues.updateComment({  
 owner: context.repo.owner,  
 repo: context.repo.repo,  
 comment\_id: botComment.id,  
 body: body  
 });  
 } else {  
 github.rest.issues.createComment({  
 owner: context.repo.owner,  
 repo: context.repo.repo,  
 issue\_number: context.issue.number,  
 body: body  
 });  
 }

### **Dependency Update Workflow**

Create .github/workflows/dependency-update.yml:

name: Dependency Updates  
  
on:  
 schedule:  
 - cron: '0 9 \* \* MON' # Every Monday at 9 AM  
 workflow\_dispatch:  
  
jobs:  
 update-dependencies:  
 runs-on: ubuntu-latest  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
 with:  
 token: ${{ secrets.GITHUB\_TOKEN }}  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v4  
 with:  
 node-version: '18'  
 cache: 'npm'  
  
 - name: Update Dependencies  
 run: |  
 npm update  
 npm audit fix  
  
 - name: Run Tests  
 run: npm test  
  
 - name: Create Pull Request  
 uses: peter-evans/create-pull-request@v5  
 with:  
 token: ${{ secrets.GITHUB\_TOKEN }}  
 commit-message: 'chore: update dependencies'  
 title: 'Automated Dependency Updates'  
 body: |  
 ## Automated Dependency Updates  
   
 This PR contains automated dependency updates.  
   
 ### Changes  
 - Updated npm dependencies to latest compatible versions  
 - Applied security fixes via npm audit  
   
 Testing  
 - All tests pass  
 - Build succeeds  
   
 Please review and merge if everything looks good.  
 branch: chore/dependency-updates  
 delete-branch: true

## **Environment Management**

### **Environment Configuration**

#### **Development Environment**

# .env.local (not committed)  
NODE\_ENV=development  
AZURE\_CLIENT\_ID=your-dev-client-id  
AZURE\_TENANT\_ID=your-tenant-id  
AI360\_API\_URL=https://api.lab45.ai/v1.1  
NEXT\_PUBLIC\_APP\_ENV=development

#### **Staging Environment**

# Vercel Environment Variables  
NODE\_ENV=production  
AZURE\_CLIENT\_ID=staging-client-id  
AZURE\_TENANT\_ID=your-tenant-id  
AI360\_API\_URL=https://api.lab45.ai/v1.1  
NEXT\_PUBLIC\_APP\_ENV=staging  
NEXT\_PUBLIC\_VERSION=staging-latest

#### **Production Environment**

# Vercel Environment Variables  
NODE\_ENV=production  
AZURE\_CLIENT\_ID=prod-client-id  
AZURE\_TENANT\_ID=your-tenant-id  
AI360\_API\_URL=https://api.lab45.ai/v1.1  
NEXT\_PUBLIC\_APP\_ENV=production  
NEXT\_PUBLIC\_VERSION=v1.0.0

### **Environment-Specific Configuration**

Create config/environments.ts:

interface EnvironmentConfig {  
 apiUrl: string;  
 debug: boolean;  
 analytics: boolean;  
 azure: {  
 clientId: string;  
 tenantId: string;  
 authority: string;  
 };  
}  
  
const environments: Record<string, EnvironmentConfig> = {  
 development: {  
 apiUrl: '<http://localhost:3000/api>',  
 debug: true,  
 analytics: false,  
 azure: {  
 clientId: process.env.AZURE\_CLIENT\_ID!,  
 tenantId: process.env.AZURE\_TENANT\_ID!,  
 authority: `https://login.microsoftonline.com/${process.env.AZURE\_TENANT\_ID}`  
 }  
 },  
 staging: {  
 apiUrl: '<https://chat-app-staging.vercel.app/api>',  
 debug: true,  
 analytics: false,  
 azure: {  
 clientId: process.env.AZURE\_CLIENT\_ID!,  
 tenantId: process.env.AZURE\_TENANT\_ID!,  
 authority: `https://login.microsoftonline.com/${process.env.AZURE\_TENANT\_ID}`  
 }  
 },  
 production: {  
 apiUrl: '<https://chat-app-prod.vercel.app/api>',  
 debug: false,  
 analytics: true,  
 azure: {  
 clientId: process.env.AZURE\_CLIENT\_ID!,  
 tenantId: process.env.AZURE\_TENANT\_ID!,  
 authority: `https://login.microsoftonline.com/${process.env.AZURE\_TENANT\_ID}`  
 }  
 }  
};  
  
export const config = environments[process.env.NEXT\_PUBLIC\_APP\_ENV || 'development'];

## **Testing Strategy**

### **Test Configuration**

Update package.json with testing scripts:

{  
 "scripts": {  
 "test": "jest",  
 "test:watch": "jest --watch",  
 "test:coverage": "jest --coverage",  
 "test:unit": "jest --testPathPattern=\_\_tests\_\_/unit",  
 "test:integration": "jest --testPathPattern=\_\_tests\_\_/integration",  
 "test:e2e": "playwright test",  
 "test:e2e:ui": "playwright test --ui"  
 }  
}

### **Jest Configuration**

Create jest.config.js:

const nextJest = require('next/jest');  
  
const createJestConfig = nextJest({  
 dir: './',  
});  
  
const customJestConfig = {  
 setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],  
 testEnvironment: 'jest-environment-jsdom',  
 testPathIgnorePatterns: [  
 '<rootDir>/.next/',  
 '<rootDir>/node\_modules/',  
 '<rootDir>/e2e/'  
 ],  
 collectCoverageFrom: [  
 'app/\*\*/\*.{js,jsx,ts,tsx}',  
 '!app/\*\*/\*.d.ts',  
 '!app/\*\*/layout.tsx',  
 '!app/\*\*/page.tsx'  
 ],  
 coverageThreshold: {  
 global: {  
 branches: 80,  
 functions: 80,  
 lines: 80,  
 statements: 80  
 }  
 },  
 moduleNameMapping: {  
 '^@/(.\*)$': '<rootDir>/$1'  
 }  
};  
  
module.exports = createJestConfig(customJestConfig);

### **Playwright E2E Configuration**

Create playwright.config.ts:

import { defineConfig, devices } from '@playwright/test';  
  
export default defineConfig({  
 testDir: './e2e',  
 fullyParallel: true,  
 forbidOnly: !!process.env.CI,  
 retries: process.env.CI ? 2 : 0,  
 workers: process.env.CI ? 1 : undefined,  
 reporter: 'html',  
   
 use: {  
 baseURL: process.env.E2E\_BASE\_URL || '<http://localhost:3000>',  
 trace: 'on-first-retry',  
 screenshot: 'only-on-failure',  
 },  
  
 projects: [  
 {  
 name: 'chromium',  
 use: { ...devices['Desktop Chrome'] },  
 },  
 {  
 name: 'firefox',  
 use: { ...devices['Desktop Firefox'] },  
 },  
 {  
 name: 'webkit',  
 use: { ...devices['Desktop Safari'] },  
 },  
 {  
 name: 'Mobile Chrome',  
 use: { ...devices['Pixel 5'] },  
 },  
 ],  
  
 webServer: {  
 command: 'npm run dev',  
 url: '<http://localhost:3000>',  
 reuseExistingServer: !process.env.CI,  
 },  
});

### **Sample E2E Test**

Create e2e/chat-flow.spec.ts:

import { test, expect } from '@playwright/test';  
  
test.describe('Chat Application', () => {  
 test.beforeEach(async ({ page }) => {  
 await page.goto('/');  
 });  
  
 test('should display sign in button when not authenticated', async ({ page }) => {  
 await expect(page.getByRole('button', { name: /sign in/i })).toBeVisible();  
 });  
  
 test('should allow user to send message after authentication', async ({ page }) => {  
 // Mock authentication  
 await page.route('\*\*/api/completions', async route => {  
 await route.fulfill({  
 status: 200,  
 contentType: 'application/json',  
 body: JSON.stringify({  
 data: { content: 'This is a test response' }  
 }),  
 });  
 });  
  
 // Simulate authenticated state  
 await page.evaluate(() => {  
 localStorage.setItem('msal.account.keys', JSON.stringify(['test-account']));  
 });  
  
 await page.reload();  
  
 // Test chat functionality  
 await page.fill('input[placeholder\*="Type your message"]', 'Hello, AI!');  
 await page.click('button[type="submit"]');  
  
 await expect(page.getByText('Hello, AI!')).toBeVisible();  
 await expect(page.getByText('This is a test response')).toBeVisible();  
 });  
  
 test('should handle error states gracefully', async ({ page }) => {  
 await page.route('\*\*/api/completions', async route => {  
 await route.fulfill({  
 status: 500,  
 contentType: 'application/json',  
 body: JSON.stringify({ error: 'Server error' }),  
 });  
 });  
  
 // Simulate authenticated state and send message  
 await page.evaluate(() => {  
 localStorage.setItem('msal.account.keys', JSON.stringify(['test-account']));  
 });  
  
 await page.reload();  
 await page.fill('input[placeholder\*="Type your message"]', 'Test message');  
 await page.click('button[type="submit"]');  
  
 await expect(page.getByText(/error/i)).toBeVisible();  
 });  
});

## **Deployment Process**

### **Vercel Configuration**

Create vercel.json:

{  
 "buildCommand": "npm run build",  
 "outputDirectory": "out",  
 "framework": "nextjs",  
 "regions": ["iad1", "sfo1"],  
 "env": {  
 "AZURE\_CLIENT\_ID": "@azure-client-id",  
 "AZURE\_TENANT\_ID": "@azure-tenant-id",  
 "AI360\_API\_URL": "@ai360-api-url"  
 },  
 "build": {  
 "env": {  
 "NEXT\_PUBLIC\_APP\_ENV": "production"  
 }  
 },  
 "functions": {  
 "app/api/completions/route.ts": {  
 "maxDuration": 30  
 }  
 },  
 "headers": [  
 {  
 "source": "/api/(.\*)",  
 "headers": [  
 {  
 "key": "Cache-Control",  
 "value": "no-cache, no-store, must-revalidate"  
 },  
 {  
 "key": "X-Content-Type-Options",  
 "value": "nosniff"  
 },  
 {  
 "key": "X-Frame-Options",  
 "value": "DENY"  
 }  
 ]  
 }  
 ]  
}

### **Health Check Endpoint**

Create app/api/health/route.ts:

import { NextResponse } from 'next/server';  
  
export async function GET() {  
 const healthCheck = {  
 status: 'healthy',  
 timestamp: new Date().toISOString(),  
 version: process.env.NEXT\_PUBLIC\_VERSION || 'unknown',  
 environment: process.env.NEXT\_PUBLIC\_APP\_ENV || 'unknown',  
 uptime: process.uptime(),  
 memory: process.memoryUsage(),  
 checks: {  
 api: 'healthy',  
 database: 'n/a', // Add if using database  
 external\_services: 'healthy'  
 }  
 };  
  
 return NextResponse.json(healthCheck);  
}

### **Deployment Scripts**

Create scripts/deploy.sh:

#!/bin/bash  
  
set -e  
  
ENVIRONMENT=${1:-staging}  
VERSION=${2:-latest}  
  
echo " Deploying to $ENVIRONMENT..."  
  
# Pre-deployment checks  
echo " Running pre-deployment checks..."  
npm run lint:strict  
npm run type-check  
npm run test  
  
# Build application  
echo " Building application..."  
npm run build  
  
# Deploy based on environment  
case $ENVIRONMENT in  
 "staging")  
 echo " Deploying to staging..."  
 vercel --prod --env staging  
 ;;  
 "production")  
 echo " Deploying to production..."  
 vercel --prod  
 ;;  
 \*)  
 echo " Unknown environment: $ENVIRONMENT"  
 exit 1  
 ;;  
esac  
  
# Post-deployment verification  
echo " Running post-deployment checks..."  
sleep 10 # Wait for deployment to be live  
  
if [ "$ENVIRONMENT" = "production" ]; then  
 HEALTH\_URL="https://chat-app-prod.vercel.app/api/health"  
else  
 HEALTH\_URL="https://chat-app-staging.vercel.app/api/health"  
fi  
  
HTTP\_STATUS=$(curl -s -o /dev/null -w "%{http\_code}" $HEALTH\_URL)  
  
if [ "$HTTP\_STATUS" = "200" ]; then  
 echo " Deployment successful! Health check passed."  
else  
 echo " Deployment failed! Health check returned status: $HTTP\_STATUS"  
 exit 1  
fi  
  
echo " Deployment to $ENVIRONMENT completed successfully!"

## **Code Review Guidelines**

### **Review Checklist**

#### **Functionality**

* Feature works as described in requirements
* Edge cases are handled appropriately
* Error handling is comprehensive
* Performance implications considered

#### **Code Quality**

* Code follows established patterns
* Functions are small and focused
* Variable names are descriptive
* Comments explain complex logic

#### **Testing**

* Unit tests cover new functionality
* Integration tests updated if needed
* Manual testing performed
* Test cases cover edge scenarios

#### **Security**

* Input validation implemented
* Authentication/authorization checked
* No sensitive data exposed
* Security best practices followed

#### **Documentation**

* README updated if needed
* API documentation updated
* Code comments added for complex logic
* Type definitions provided

### **Review Process**

1. **Automated Checks**: All CI checks must pass
2. **Self Review**: Author reviews their own code
3. **Peer Review**: At least one team member reviews
4. **Code Owner Review**: Required for sensitive areas
5. **Approval**: All required approvals obtained
6. **Merge**: Squash and merge to maintain clean history

### **Review Comments**

Use these prefixes for review comments:

* **MUST**: Critical issue that blocks merge
* **SHOULD**: Important suggestion for improvement
* **CONSIDER**: Optional suggestion or alternative approach
* **QUESTION**: Clarification needed
* **PRAISE**: Positive feedback

Example:

MUST: Add error handling for the API request on line 45.  
SHOULD: Consider using a more descriptive variable name than 'data'.  
CONSIDER: Could we extract this logic into a separate utility function?  
QUESTION: Why did we choose this approach over using the existing hook?  
PRAISE: Great test coverage for this component!

## **Security and Compliance**

### **Security Scans**

#### **SAST (Static Application Security Testing)**

# CodeQL Analysis  
- name: Initialize CodeQL  
 uses: github/codeql-action/init@v2  
 with:  
 languages: javascript, typescript  
  
- name: Perform CodeQL Analysis  
 uses: github/codeql-action/analyze@v2

#### **Dependency Scanning**

# Snyk Security Scan  
- name: Run Snyk to check for vulnerabilities  
 uses: snyk/actions/node@master  
 env:  
 SNYK\_TOKEN: ${{ secrets.SNYK\_TOKEN }}  
 with:  
 args: --severity-threshold=high

#### **Secret Scanning**

# TruffleHog Secret Scanning  
- name: TruffleHog OSS  
 uses: trufflesecurity/trufflehog@main  
 with:  
 path: ./  
 base: main  
 head: HEAD  
 extra\_args: --debug --only-verified

### **Compliance Requirements**

#### **Security Headers**

// middleware.ts  
import { NextResponse } from 'next/server';  
import type { NextRequest } from 'next/server';  
  
export function middleware(request: NextRequest) {  
 const response = NextResponse.next();  
  
 // Security headers  
 response.headers.set('X-Content-Type-Options', 'nosniff');  
 response.headers.set('X-Frame-Options', 'DENY');  
 response.headers.set('X-XSS-Protection', '1; mode=block');  
 response.headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');  
 response.headers.set(  
 'Content-Security-Policy',  
 "default-src 'self'; script-src 'self' 'unsafe-eval'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; connect-src 'self' <https://login.microsoftonline.com> <https://api.lab45.ai>;"  
 );  
  
 return response;  
}  
  
export const config = {  
 matcher: [  
 '/((?!api|\_next/static|\_next/image|favicon.ico).\*)',  
 ],  
};

#### **Audit Logging**

// utils/audit.ts  
interface AuditEvent {  
 userId?: string;  
 action: string;  
 resource: string;  
 timestamp: string;  
 ip?: string;  
 userAgent?: string;  
}  
  
export function logAuditEvent(event: AuditEvent): void {  
 if (process.env.NODE\_ENV === 'production') {  
 // Send to audit logging service  
 console.log('AUDIT:', JSON.stringify(event));  
 }  
}

## **Monitoring and Rollback**

### **Application Monitoring**

#### **Health Checks**

// app/api/health/route.ts (extended)  
export async function GET() {  
 const checks = await Promise.allSettled([  
 checkDatabase(),  
 checkExternalAPIs(),  
 checkMemoryUsage(),  
 checkDiskSpace()  
 ]);  
  
 const health = {  
 status: checks.every(check => check.status === 'fulfilled') ? 'healthy' : 'unhealthy',  
 timestamp: new Date().toISOString(),  
 version: process.env.NEXT\_PUBLIC\_VERSION,  
 checks: {  
 database: checks[0].status === 'fulfilled' ? 'healthy' : 'unhealthy',  
 external\_apis: checks[1].status === 'fulfilled' ? 'healthy' : 'unhealthy',  
 memory: checks[2].status === 'fulfilled' ? 'healthy' : 'unhealthy',  
 disk: checks[3].status === 'fulfilled' ? 'healthy' : 'unhealthy'  
 }  
 };  
  
 const statusCode = health.status === 'healthy' ? 200 : 503;  
 return NextResponse.json(health, { status: statusCode });  
}

#### **Error Tracking**

// utils/errorTracking.ts  
import \* as Sentry from '@sentry/nextjs';  
  
export function initErrorTracking(): void {  
 if (process.env.NODE\_ENV === 'production') {  
 Sentry.init({  
 dsn: process.env.SENTRY\_DSN,  
 environment: process.env.NEXT\_PUBLIC\_APP\_ENV,  
 tracesSampleRate: 0.1,  
 });  
 }  
}  
  
export function captureError(error: Error, context?: Record<string, any>): void {  
 if (process.env.NODE\_ENV === 'production') {  
 Sentry.captureException(error, { extra: context });  
 } else {  
 console.error('Error:', error, context);  
 }  
}

### **Rollback Strategy**

#### **Automated Rollback**

# .github/workflows/rollback.yml  
name: Emergency Rollback  
  
on:  
 workflow\_dispatch:  
 inputs:  
 version:  
 description: 'Version to rollback to'  
 required: true  
 reason:  
 description: 'Reason for rollback'  
 required: true  
  
jobs:  
 rollback:  
 runs-on: ubuntu-latest  
 environment: production  
   
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v4  
 with:  
 ref: ${{ github.event.inputs.version }}  
  
 - name: Deploy Previous Version  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID }}  
 vercel-args: '--prod'  
  
 - name: Verify Rollback  
 run: |  
 sleep 30  
 curl -f <https://chat-app-prod.vercel.app/api/health> - name: Notify Team  
 uses: 8398a7/action-slack@v3  
 with:  
 status: custom  
 custom\_payload: |  
 {  
 text: "Emergency rollback completed",  
 attachments: [{  
 color: "warning",  
 fields: [{  
 title: "Version",  
 value: "${{ github.event.inputs.version }}",  
 short: true  
 }, {  
 title: "Reason",  
 value: "${{ github.event.inputs.reason }}",  
 short: true  
 }]  
 }]  
 }  
 env:  
 SLACK\_WEBHOOK\_URL: ${{ secrets.SLACK\_WEBHOOK }}

#### **Manual Rollback Steps**

# 1. Identify the last known good version  
git tag --sort=-version:refname | head -5  
  
# 2. Create hotfix branch from good version  
git checkout -b hotfix/rollback-v1.2.3 v1.2.3  
  
# 3. Push and deploy  
git push origin hotfix/rollback-v1.2.3  
  
# 4. Trigger production deployment  
# (This will be handled by CI/CD pipeline)  
  
# 5. Verify deployment  
curl -f <https://chat-app-prod.vercel.app/api/health># 6. Notify stakeholders  
# Send notification via Slack/email

## **Troubleshooting Guide**

### **Common CI/CD Issues**

#### **Build Failures**

# Check build logs  
npm run build 2>&1 | tee build.log  
  
# Common fixes:  
# 1. Clear cache  
rm -rf .next node\_modules package-lock.json  
npm install  
  
# 2. Check TypeScript errors  
npm run type-check  
  
# 3. Check for missing environment variables  
grep -r "process.env" app/ --include="\*.ts" --include="\*.tsx"

#### **Test Failures**

# Run tests with verbose output  
npm test -- --verbose  
  
# Run specific test file  
npm test -- ChatInterface.test.tsx  
  
# Check test coverage  
npm run test:coverage  
  
# Debug test in watch mode  
npm run test:watch

#### **Deployment Issues**

# Check Vercel deployment logs  
vercel logs  
  
# Verify environment variables  
vercel env ls  
  
# Test deployment locally  
vercel dev  
  
# Check health endpoint  
curl -v <https://your-app.vercel.app/api/health>

### **Performance Issues**

#### **Bundle Size Analysis**

# Analyze bundle size  
npm run build  
npx @next/bundle-analyzer  
  
# Check for large dependencies  
npm ls --depth=0 --long

#### **Runtime Performance**

// Add performance monitoring  
export function withPerformanceMonitoring<T extends (...args: any[]) => any>(  
 fn: T,  
 name: string  
): T {  
 return ((...args: any[]) => {  
 const start = performance.now();  
 const result = fn(...args);  
 const end = performance.now();  
   
 if (end - start > 100) { // Log slow operations  
 console.warn(`Slow operation: ${name} took ${end - start}ms`);  
 }  
   
 return result;  
 }) as T;  
}

### **Security Incident Response**

#### **Security Incident Playbook**

1. **Detection**: Monitor security alerts from CI/CD pipeline
2. **Assessment**: Evaluate severity and impact
3. **Containment**: Disable affected features if necessary
4. **Investigation**: Review logs and identify root cause
5. **Remediation**: Deploy security fixes
6. **Recovery**: Restore normal operations
7. **Post-incident**: Document lessons learned

#### **Emergency Contacts**

# .github/SECURITY.md  
## Security Contacts  
  
- \*\*Security Team\*\*: [security@company.com](mailto:security@company.com)- \*\*On-call Engineer\*\*: +1-xxx-xxx-xxxx  
- \*\*DevOps Lead\*\*: [devops@company.com](mailto:devops@company.com)## Incident Response  
For security incidents, please:  
1. Do NOT create public GitHub issues  
2. Email [security@company.com](mailto:security@company.com) immediately  
3. Include severity level and impact assessment