AI Code Assistant - Usage Examples

Example 1: Automatic Code Cleanup

```
import { cleanupOperations } from './ai-assistant/core/cleanup/cleanupOperations'
import { changeManager } from './ai-assistant/services/changeManager'

async function cleanupCodebase() {
  // Initialize
  await changeManager.initialize()

  // Scan for cleanup opportunities
  const operations = await cleanupOperations.scanForCleanup('./src')

  console.log(`Found ${operations.length} cleanup opportunities`)

  // Process each operation
  for (const op of operations) {
    if (op.autoApply) {
      console.log(`Auto-applying: ${op.description}`)
      // Changes are automatically proposed and assessed
    }
  }
}
```

Example 2: AI-Powered Code Analysis

```
import { ollamaService } from './ai-assistant/services/ollamaService'
import fs from 'fs/promises'

async function analyzeFile(filePath: string) {
  // Read file
  const code = await fs.readFile(filePath, 'utf-8')

  // Analyze with AI
  const analysis = await ollamaService.analyzeCode(code, filePath)

  console.log('AI Analysis:')
  console.log(analysis)

  // Get improvement suggestions
  const suggestions = await ollamaService.suggestImprovements(
    code,
    'Focus on performance and type safety'
  )

  console.log('\nSuggestions:')
  console.log(suggestions)
}

// Usage
analyzeFile('./src/components/MyComponent.tsx')
```

Example 3: Safe Code Refactoring

```
import { changeManager } from './ai-assistant/services/changeManager'
import { ollamaService } from './ai-assistant/services/ollamaService'
import fs from 'fs/promises'

async function refactorWithSafety(filePath: string, instruction: string) {
  // Read current code
  const currentCode = await fs.readFile(filePath, 'utf-8')

  // Get AI refactoring suggestion
  const refactoredCode = await ollamaService.refactorCode(
    currentCode,
    instruction
  )

  // Propose the change
  const { change, assessment } = await changeManager.proposeChange(
    filePath,
    'refactor',
    `Refactor: ${instruction}`,
    refactoredCode,
    'deepseek-coder',
    'AI-suggested refactoring for improved code quality'
  )

  console.log(`Risk Score: ${assessment.score}/10`)
  console.log(`Recommendation: ${assessment.recommendation}`)

  // Execute based on risk
  if (assessment.recommendation === 'auto-apply') {
    console.log('Safe change - auto-applying')
    await changeManager.executeChange(change.id)
  } else if (assessment.recommendation === 'create-pr') {
    console.log('Medium risk - creating PR for review')
    await changeManager.executeChange(change.id)
  } else {
    console.log('High risk - requires manual approval')
    // Change stays in pending state
  }
}

// Usage
refactorWithSafety(
  './src/lib/utils.ts',
  'Extract duplicate code into reusable functions'
)
```

Example 4: Batch Processing Multiple Files

```
import { codeIndexer } from './ai-assistant/core/indexer/codeIndexer'
import { cleanupOperations } from './ai-assistant/core/cleanup/cleanupOperations'
import { changeManager } from './ai-assistant/services/changeManager'

async function batchCleanup(pattern: string) {
  // Index codebase
  const index = await codeIndexer.indexCodebase('./src')

  // Find matching files
  const files = codeIndexer.searchFiles(pattern)

  console.log(`Processing ${files.length} files...`)

  // Process each file
  for (const fileIndex of files) {
    console.log(`\nProcessing: ${fileIndex.filePath}`)

    // Remove unused imports
    const importChange = await cleanupOperations.removeUnusedImports(
      fileIndex.filePath
    )

    if (importChange) {
      const { change, assessment } = await changeManager.proposeChange(
        fileIndex.filePath,
        'update',
        importChange.description,
        '', // Content would be generated
        'cleanup-engine',
        importChange.reasoning
      )

      // Auto-apply safe changes
      if (assessment.score === 10) {
        await changeManager.executeChange(change.id)
        console.log('  ✅ Applied')
      }
    }
  }
}

// Usage: Clean all TypeScript files in components
batchCleanup('components/.*\\.tsx?$')
```

Example 5: Interactive Code Review

```
import { changeManager } from './ai-assistant/services/changeManager'
import readline from 'readline'

async function interactiveReview() {
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
  })

  // Get pending changes
  const pending = changeManager.getPendingChanges()

  console.log(`\n${pending.length} changes pending review\n`)

  for (const change of pending) {
    console.log('-'.repeat(60))
    console.log(`File: ${change.filePath}`)
    console.log(`Type: ${change.type}`)
    console.log(`Description: ${change.description}`)
    console.log(`Risk Score: ${change.riskScore}/10`)
    console.log(`\nDiff:\n${change.diff}`)
    console.log('-'.repeat(60))

    const answer = await new Promise<string>(resolve => {
      rl.question('\nApprove? (y/n/s=skip): ', resolve)
    })

    if (answer.toLowerCase() === 'y') {
      await changeManager.approveChange(change.id)
      console.log('✅ Approved and applied')
    } else if (answer.toLowerCase() === 'n') {
      await changeManager.rejectChange(change.id, 'Manual rejection')
      console.log('❌ Rejected')
    } else {
      console.log('⏭️  Skipped')
    }
  }

  rl.close()

  // Show statistics
  const stats = changeManager.getStatistics()
  console.log('\n📊 Statistics:')
  console.log(`   Approved: ${stats.approved}`)
  console.log(`   Rejected: ${stats.rejected}`)
  console.log(`   Pending: ${stats.pending}`)
}

// Usage
interactiveReview()
```

Example 6: Generate Documentation

```
import { ollamaService } from './ai-assistant/services/ollamaService'
import { codeIndexer } from './ai-assistant/core/indexer/codeIndexer'
import fs from 'fs/promises'

async function generateDocs(filePath: string) {
  // Index the file
  await codeIndexer.indexCodebase(filePath)
  const fileIndex = codeIndexer.getFileIndex(filePath)

  if (!fileIndex) {
    console.log('File not found')
    return
  }

  // Read file content
  const content = await fs.readFile(filePath, 'utf-8')

  // Generate docs for each function
  for (const func of fileIndex.functions) {
    if (!func.isExported) continue

    console.log(`\nGenerating docs for: ${func.name}`)

    // Extract function code
    const lines = content.split('\n')
    const funcCode = lines.slice(func.lineStart - 1, func.lineEnd).join('\n')

    // Generate documentation
    const docs = await ollamaService.generateDocumentation(funcCode, func.name)

    console.log(docs)
  }
}

// Usage
generateDocs('./src/lib/utils.ts')
```

Example 7: Custom Risk Rules

```
import { riskAssessor } from './ai-assistant/core/risk-engine/riskAssessor'
import { CodeChange } from './ai-assistant/config/types'

// Create a custom change with specific characteristics
const customChange: CodeChange = {
  id: 'custom-1',
  timestamp: new Date(),
  type: 'update',
  filePath: './src/app/api/auth/route.ts', // Auth file = high risk
  description: 'Update authentication logic',
  riskScore: 0,
  status: 'pending',
  aiModel: 'deepseek-coder',
  reasoning: 'Improving security',
  diff: '... large diff with 150 lines changed ...'
}

// Assess risk
const assessment = riskAssessor.assessRisk(customChange)

console.log('Risk Assessment:')
console.log(`  Score: ${assessment.score}/10`)
console.log(`  Category: ${assessment.category}`)
console.log(`  Recommendation: ${assessment.recommendation}`)
console.log('\nRisk Factors:')
assessment.factors.forEach(factor => {
  console.log(`  - ${factor.name}: ${factor.impact}/10`)
  console.log(`    ${factor.description}`)
})
```

Example 8: Rollback System

```
import { changeManager } from './ai-assistant/services/changeManager'
import { safetySystem } from './ai-assistant/core/safety/safetySystem'

async function demonstrateRollback() {
  // Get applied changes
  const applied = changeManager.getAppliedChanges()

  console.log(`${applied.length} changes have been applied`)

  // Show recent changes
  const recent = applied.slice(-5)
  console.log('\nRecent changes:')
  recent.forEach((change, i) => {
    console.log(`${i + 1}. ${change.description} (${change.filePath})`)
  })

  // Rollback the last change
  if (recent.length > 0) {
    const lastChange = recent[recent.length - 1]
    console.log(`\nRolling back: ${lastChange.description}`)

    await changeManager.rollbackChange(lastChange.id)
    console.log('✅ Rollback successful')

    // List available backups
    const backups = await safetySystem.listBackups()
    console.log(`\n${backups.length} backups available`)
  }
}

// Usage
demonstrateRollback()
```

Example 9: Integration with CI/CD

```
import { cleanupOperations } from './ai-assistant/core/cleanup/cleanupOperations'
import { changeManager } from './ai-assistant/services/changeManager'

async function cicdCleanup() {
  console.log('🔍 Running AI Code Assistant in CI/CD mode...')

  // Initialize
  await changeManager.initialize()

  // Scan for issues
  const operations = await cleanupOperations.scanForCleanup('./src')

  if (operations.length === 0) {
    console.log('✅ No issues found')
    process.exit(0)
  }

  console.log(`⚠️  Found ${operations.length} issues`)

  // Only auto-apply safe changes in CI/CD
  const safeOps = operations.filter(op => op.autoApply)

  console.log(`🔧 Auto-fixing ${safeOps.length} safe issues...`)

  // Apply safe fixes
  for (const op of safeOps) {
    console.log(`  - ${op.description}`)
  }

  // Report remaining issues
  const remaining = operations.length - safeOps.length
  if (remaining > 0) {
    console.log(`\n⚠️  ${remaining} issues require manual review`)
    process.exit(1)
  }

  console.log('\n✅ All issues resolved')
  process.exit(0)
}

// Usage in CI/CD pipeline
cicdCleanup()
```

Example 10: Web API Integration

```
// In your Next.js API route
import { NextRequest, NextResponse } from 'next/server'
import { ollamaService } from '@/ai-assistant/services/ollamaService'
import { changeManager } from '@/ai-assistant/services/changeManager'

export async function POST(request: NextRequest) {
  const { action, filePath, code } = await request.json()

  switch (action) {
    case 'analyze':
      const analysis = await ollamaService.analyzeCode(code, filePath)
      return NextResponse.json({ analysis })

    case 'suggest':
      const suggestions = await ollamaService.suggestImprovements(code, '')
      return NextResponse.json({ suggestions })

    case 'apply':
      const { change, assessment } = await changeManager.proposeChange(
        filePath,
        'update',
        'User-requested change',
        code,
        'deepseek-coder',
        'Manual change via web UI'
      )

      return NextResponse.json({ change, assessment })

    default:
      return NextResponse.json({ error: 'Invalid action' }, { status: 400 })
  }
}
```

Tips for Best Results

1. **Start Small**: Test on a few files before running on entire codebase
2. **Review PRs**: Always review auto-generated PRs before merging
3. **Monitor Backups**: Regularly check and clean old backups
4. **Adjust Risk Scores**: Customize risk thresholds based on your needs
5. **Use Specific Prompts**: More specific AI prompts yield better results
6. **Test in Development**: Always test changes in dev environment first
7. **Keep Ollama Updated**: Regularly update models for better performance
8. **Monitor Resources**: Watch CPU/memory usage during large operations