Mode Specifications

Version: 1.0.0
Status: Draft
Last Updated: 2026-01-07
Owner: Code-Factory Core Team

Table of Contents

Overview

Code-Factory operates through four distinct modes, each designed for a specific phase of the software development lifecycle. Each mode is a self-contained workflow with its own state machine, UI, and interactions with the LLM and external services.

Mode Summary

Mode	Purpose	Input	Output
INTAKE	Capture requirements as specs	User requirements (text/voice)	Markdown specification
REVIEW	Analyze existing code against specs	Code directory + spec(s)	Analysis report
CHANGE_ORDER	Implement changes from specs	Change request + spec	Code changes + PR
RESCUE	Debug and fix issues	Error/problem description	Solution + fixes

Common Patterns

All modes share these characteristics:
- **Interactive TUI**: Real-time feedback and progress
- **LLM-powered**: AI assists but human validates
- **Git-integrated**: All outputs are versioned

- **Reversible**: Can undo/rollback operations
- **Collaborative**: GitHub integration optional

INTAKE Mode

Purpose

Transform natural language requirements into structured, comprehensive specifications that serve as the source of truth for implementation.

User Flow


$ factory intake
Step 1: Requirement Gathering
What do you want to build?
User authentication system with email/password and JWT tokens. Should support password reset via email.
[Continue] [Voice Input] [Import from File]
Step 2: Clarifying Questions (AI-generated)
I have a few questions to better understand your needs:
1. What database will you be using?
> PostgreSQL
2. Should we support OAuth providers (Google, GitHub)?
> No, just email/password for now
3. What's your preference for password hashing?
> bcrypt is fine
[Continue]
Step 3: Specification Generation
Generating specification...
85%
Current section: API Endpoints
✓ Overview
✓ Requirements
✓ Data Model
→ API Endpoints
○ Security Considerations
○ Testing Strategy
Step 4: Review & Edit
Sections
User Authentication
● Overview
Overview


● Requirements
● Data Model
● API Spec
● Security
● Testing
This specification defines a secure...
Requirements
Functional
- Users must register with email...
[e] Edit [s] Save [r] Regenerate [q] Cancel
Step 5: Save & Commit
Saving specification...
✓ Created: contracts/specs/user-authentication.md
✓ Updated: contracts/README.md
✓ Git commit: "Add spec: User Authentication"
Optional: Push to GitHub?
[y] Yes [n] No, stay local
Success!
Specification created:
contracts/specs/user-authentication.md
Next steps:
• Review the spec: factory review
• Start implementation: factory change-order
• Share with team: git push origin main
[Press any key to exit]

Technical Specification

Command Line Interface

```
# Interactive mode (default)
factory intake

# With initial prompt
factory intake --prompt "Build a REST API for user management"

# From file
factory intake --file requirements.txt

# Voice input (if supported)
factory intake --voice

# Skip questions (use defaults)
factory intake --no-questions --prompt "..."

# Specify output location
factory intake --output contracts/specs/custom-name.md

# Use specific template
factory intake --template contracts/templates/api-spec.md
```

Configuration Options

```
intake:
  # LLM settings for spec generation
  llm:
    temperature: 0.7        # Creativity vs consistency
    max_tokens: 4096        # Max spec length

  # Question generation
  questions:
    enabled: true           # Ask clarifying questions
    max_questions: 5        # Limit number of questions

  # Spec structure
  template: "default"       # or custom path
  sections:                 # Required sections
    - overview
    - requirements
    - api_specification
    - data_model
    - security
    - testing

  # Auto-save
  auto_save: true           # Save drafts automatically
  save_interval: 30s        # How often to save drafts
```

State Machine

```go
type IntakeState int

const (
    IntakeStateInit IntakeState = iota
    IntakeStateGatheringInput
    IntakeStateAskingQuestions
    IntakeStateGeneratingSpec
    IntakeStateReviewEdit
    IntakeStateSaving
    IntakeStateComplete
    IntakeStateError
)

type IntakeContext struct {
    // User input
    UserRequirements string
    AnsweredQuestions map[string]string

    // Generated content
    GeneratedSpec *Specification
    Questions     []Question

    // State
    CurrentState   IntakeState
    CurrentSection string
    Progress       float64

    // Services
    LLM    LLMService
    Git    GitService
    GitHub GitHubService
}

func (ctx *IntakeContext) Advance() error {
    switch ctx.CurrentState {
    case IntakeStateInit:
        return ctx.transitionToGathering()
    case IntakeStateGatheringInput:
        return ctx.transitionToQuestions()
    case IntakeStateAskingQuestions:
        return ctx.transitionToGeneration()
    case IntakeStateGeneratingSpec:
        return ctx.transitionToReview()
    case IntakeStateReviewEdit:
        return ctx.transitionToSaving()
    case IntakeStateSaving:
        return ctx.transitionToComplete()
    }
    return nil
}
```

LLM Prompts

Initial Spec Generation:

```
You are a technical specification writer. Your task is to transform user requirements
into a comprehensive, implementation-ready specification.

User Requirements:
{user_input}

Clarifications:
{answered_questions}

Generate a detailed specification with these sections:
1. Overview - High-level description
2. Requirements - Functional and non-functional
3. Data Model - Database schema, entities
4. API Specification - Endpoints, request/response formats
5. Security Considerations - Authentication, authorization, data protection
6. Testing Strategy - Unit, integration, e2e tests

Format: Markdown with YAML frontmatter
Style: Clear, precise, actionable
Audience: Software engineers

Begin specification:
```

Question Generation:

```
You are helping gather requirements for a software project.

User wants to build:
{user_input}

Generate 3-5 clarifying questions to better understand their needs. Focus on:
- Technical stack preferences
- Scale and performance requirements
- Security and compliance needs
- Integration with existing systems

Format as JSON array:
[
  {"question": "...", "type": "choice", "options": ["A", "B"]},
  {"question": "...", "type": "text"}
]
```

File Output

contracts/specs/{feature-name}.md:

```
---
id: user-auth-001
title: User Authentication
status: draft
created: 2026-01-07T10:30:00Z
updated: 2026-01-07T10:30:00Z
author: johndoe
tags: [auth, security, backend]
priority: high
version: 1.0.0
---

# User Authentication

## Overview

This specification defines the user authentication system for the application. It
provides secure email/password authentication with JWT token-based session management
and password reset functionality.

**Goals:**
- Secure user registration and login
- Token-based authentication for APIs
- Password reset via email
- Account security best practices

**Non-Goals:**
- OAuth/SSO integration (future phase)
- Multi-factor authentication (future phase)
- Password strength meter UI

## Requirements

### Functional Requirements

**FR-1: User Registration**
- Users must be able to register with email and password
- Email must be unique and validated
- Password must meet minimum security requirements (8+ chars, mixed case, number)
- Confirmation email sent after registration
- Account not active until email confirmed

**FR-2: User Login**
- Users authenticate with email and password
- Successful login returns JWT access token and refresh token
- Failed login attempts are rate-limited (5 attempts per 15 minutes)
- Account locked after 10 failed attempts

**FR-3: Password Reset**
- Users can request password reset via email
- Reset link expires after 1 hour
- Old password invalidated after reset
- User notified of password change

**FR-4: Token Management**
- Access tokens valid for 15 minutes
- Refresh tokens valid for 7 days
- Refresh tokens rotated on use
- All tokens invalidated on logout

### Non-Functional Requirements
```

```
**NFR-1: Security**
- Passwords hashed with bcrypt (cost factor 12)
- JWT tokens signed with RS256
- HTTPS required for all auth endpoints
- Protection against common attacks (CSRF, XSS, injection)

**NFR-2: Performance**
- Login response time < 200ms (p95)
- Registration response time < 500ms (p95)
- Password reset email sent < 2 seconds

**NFR-3: Availability**
- 99.9% uptime for auth service
- Graceful degradation if email service down

## Data Model

### User Entity

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    email_verified BOOLEAN DEFAULT FALSE,
    account_locked BOOLEAN DEFAULT FALSE,
    failed_login_attempts INTEGER DEFAULT 0,
    last_login_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_email_verified ON users(email_verified);
```

Refresh Token Entity

```sql
CREATE TABLE refresh_tokens (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token_hash VARCHAR(255) UNIQUE NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    revoked BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_refresh_tokens_user_id ON refresh_tokens(user_id);
CREATE INDEX idx_refresh_tokens_expires_at ON refresh_tokens(expires_at);
```
```

Password Reset Token Entity

```
CREATE TABLE password_reset_tokens (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token_hash VARCHAR(255) UNIQUE NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_password_reset_tokens_user_id ON password_reset_tokens(user_id);
```

API Specification

POST /api/auth/register

Register a new user account.

Request:

```
{
  "email": "user@example.com",
  "password": "SecureP@ssw0rd"
}
```

Response (201 Created):

```
{
  "user_id": "550e8400-e29b-41d4-a716-446655440000",
  "message": "Registration successful. Please check your email to verify your ac-
count."
}
```

Error Responses:
- 400: Invalid email or password format
- 409: Email already registered
- 500: Server error

Rate Limit: 5 requests per minute per IP

POST /api/auth/login

Authenticate user and return tokens.

Request:

```
{
  "email": "user@example.com",
  "password": "SecureP@ssw0rd"
}
```

Response (200 OK):

```
{
  "access_token": "eyJhbGc...",
  "refresh_token": "eyJhbGc...",
  "token_type": "Bearer",
  "expires_in": 900
}
```

Error Responses:
- 401: Invalid credentials
- 403: Account locked or not verified
- 429: Too many login attempts
- 500: Server error

Rate Limit: 5 requests per 15 minutes per email

POST /api/auth/refresh

Refresh access token using refresh token.

Request:

```
{
  "refresh_token": "eyJhbGc..."
}
```

Response (200 OK):

```
{
  "access_token": "eyJhbGc...",
  "refresh_token": "eyJhbGc...",
  "token_type": "Bearer",
  "expires_in": 900
}
```

Error Responses:
- 401: Invalid or expired refresh token
- 500: Server error

POST /api/auth/logout

Invalidate current refresh token.

Request Headers:

```
Authorization: Bearer {access_token}
```

Request:

```
{
  "refresh_token": "eyJhbGc..."
}
```

Response (204 No Content)

POST /api/auth/password/reset-request

Request password reset email.

Request:

```
{
  "email": "user@example.com"
}
```

Response (200 OK):

```
{
  "message": "If an account exists with that email, a password reset link has been sent."
}
```

Note: Always return success to prevent email enumeration.

Rate Limit: 3 requests per hour per email

POST /api/auth/password/reset

Reset password using token from email.

Request:

```
{
  "token": "abc123...",
  "new_password": "NewSecureP@ssw0rd"
}
```

Response (200 OK):

```
{
  "message": "Password reset successful. You can now log in with your new password."
}
```

Error Responses:
- 400: Invalid token or password
- 404: Token not found or expired
- 500: Server error

Security Considerations

Authentication

1. **Password Hashing**
   - Use bcrypt with cost factor 12
   - Never log or store plain text passwords
   - Implement password history (prevent reuse of last 5)

2. **JWT Tokens**
   - Sign with RS256 (asymmetric)
   - Include: user_id, email, exp, iat, jti
   - Rotate signing keys every 90 days
   - Store public keys for verification

3. **Rate Limiting**
   - Implement at application and infrastructure level
   - Use distributed rate limiter (Redis) for horizontal scaling
   - Log excessive failed attempts

Authorization

1. **Token Validation**
   - Verify signature
   - Check expiration
   - Validate issuer and audience
   - Check against revocation list (for sensitive operations)

2. **Refresh Token Rotation**
   - Issue new refresh token on each use
   - Invalidate old refresh token immediately
   - Detect token reuse attacks

Data Protection

1. **Sensitive Data**
   - Email addresses: GDPR compliance
   - Password hashes: Never expose in APIs
   - Tokens: Secure transmission only (HTTPS)

2. **Audit Logging**
   - Log all authentication events
   - Include: timestamp, user_id, IP, user agent, outcome
   - Retain logs for 90 days
   - Alert on suspicious patterns

Common Attacks

1. **Brute Force**
   - Rate limiting on login endpoint
   - Account lockout after threshold
   - CAPTCHA after 3 failed attempts

2. **Credential Stuffing**
   - Monitor for unusual login patterns

   - Implement device fingerprinting
   - Optional: breach password detection

3. **Token Theft**
   - Short access token lifetime
   - Refresh token rotation
   - Secure cookie attributes (HttpOnly, Secure, SameSite)

Testing Strategy

Unit Tests

User Registration:
- Valid registration succeeds
- Duplicate email rejected
- Weak password rejected
- Email validation works

User Login:
- Valid credentials succeed
- Invalid credentials fail
- Locked account cannot login
- Unverified account cannot login

Token Management:
- Access token generation and validation
- Refresh token rotation
- Token expiration handling

Password Reset:
- Reset request generates valid token
- Token expiration works
- Used token cannot be reused

Integration Tests

Registration Flow:
1. Register new user
2. Verify email sent
3. Confirm email verification
4. Login successfully

Login Flow:
1. Login with valid credentials
2. Receive tokens
3. Access protected endpoint with access token
4. Refresh token before expiry
5. Logout and verify tokens invalid

Password Reset Flow:
1. Request password reset
2. Receive email with token
3. Reset password with token

4. Login with new password
5. Verify old password doesn't work

End-to-End Tests

Scenarios:
- New user registration → verification → first login
- Existing user login → token refresh → logout
- Forgotten password → reset → login
- Multiple failed login attempts → account lock → support unlock

Security Tests

- SQL injection attempts
- XSS attempts in email field
- CSRF token validation
- Rate limiting enforcement
- Token forgery attempts
- Password strength enforcement

Performance Tests

- Login endpoint: 1000 req/s sustained
- Registration endpoint: 100 req/s sustained
- Token refresh endpoint: 500 req/s sustained
- Database connection pooling under load

Implementation Notes

Technology Stack

Backend:
- Language: Go 1.21+
- Framework: Chi router or Gin
- Database: PostgreSQL 14+
- Cache: Redis 7+ (for rate limiting, token blacklist)

Libraries:
- JWT: github.com/golang-jwt/jwt/v5
- Password hashing: golang.org/x/crypto/bcrypt
- Email: github.com/wneessen/go-mail
- Validation: github.com/go-playground/validator/v10

Configuration

```yaml
auth:
  jwt:
    private_key_path: /path/to/private.pem
    public_key_path: /path/to/public.pem
    access_token_ttl: 15m
    refresh_token_ttl: 168h  # 7 days

  password:
    bcrypt_cost: 12
    min_length: 8
    require_uppercase: true
    require_lowercase: true
    require_number: true
    require_special: false

  rate_limit:
    login_attempts: 5
    login_window: 15m
    registration_per_ip: 5
    registration_window: 1h

  email:
    from: noreply@example.com
    smtp_host: smtp.example.com
    smtp_port: 587
    verification_link_ttl: 24h
    reset_link_ttl: 1h
```

Error Handling

Error Codes:

- AUTH001: Invalid credentials
- AUTH002: Account locked
- AUTH003: Email not verified
- AUTH004: Token expired
- AUTH005: Token invalid
- AUTH006: Password too weak
- AUTH007: Email already registered

Error Response Format:

```yaml
{
  "error": {
    "code": "AUTH001",
    "message": "Invalid email or password",
    "details": null
  }
}
```

Monitoring

Metrics to Track:

- Registration rate
- Login success/failure rate
- Token refresh rate

- Password reset requests
- Failed login attempts per user
- API response times

Alerts:
- High failed login rate
- Unusual registration spike
- Password reset abuse
- Database connection errors
- Email service failures

Dependencies

Internal

- User service (for profile management)
- Email service (for verification, notifications)
- Audit logging service

External

- PostgreSQL database
- Redis cache
- Email SMTP server

Deployment Considerations

Environment Variables:

```
DATABASE_URL=postgresql://...
REDIS_URL=redis://...
JWT_PRIVATE_KEY=...
JWT_PUBLIC_KEY=...
SMTP_HOST=...
SMTP_USERNAME=...
SMTP_PASSWORD=...
```

Database Migrations:
- Use migration tool (golang-migrate, goose)
- Version all schema changes
- Test migrations in staging first

Secrets Management:
- Use secret manager (AWS Secrets Manager, HashiCorp Vault)
- Never commit secrets to version control
- Rotate secrets regularly

Related Specifications

- User Profile Management (./user-profile.md)
- Email Service (./email-service.md)
- API Gateway Configuration (./api-gateway.md)

Revision History

Version	Date	Author	Changes
1.0.0	2026-01-07	johndoe	Initial specification

Approval:
- [ ] Product Owner
- [ ] Technical Lead
- [ ] Security Team
- [ ] QA Lead

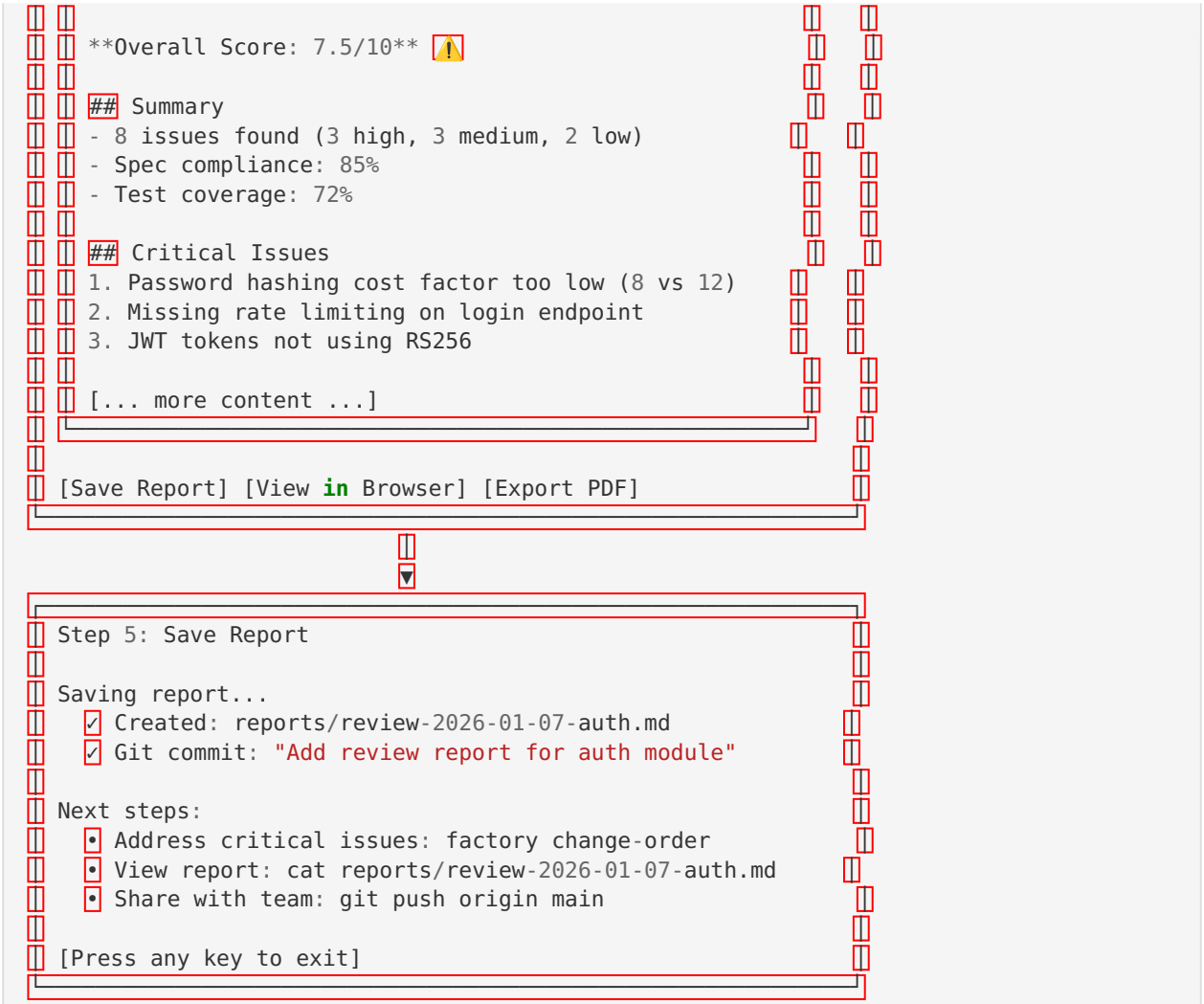
REVIEW Mode

Purpose

Analyze existing codebase against specifications to identify gaps, violations, technical debt, and improvement opportunities. Generates comprehensive reports with actionable recommendations.

User Flow


$ factory review src/auth
Step 1: Code Discovery
Scanning directory: src/auth/
Found 12 files:
✓ src/auth/handler.go
✓ src/auth/service.go
✓ src/auth/repository.go
✓ src/auth/models.go
✓ ... and 8 more
Total lines of code: 2,847
[Continue] [Filter files] [Cancel]
Step 2: Spec Matching
Looking for related specifications...
Found 2 matching specs:
✓ contracts/specs/user-authentication.md
✓ contracts/specs/security-standards.md
Would you like to include these specs in the review?
[Yes, review against specs] [No, general code review]
Step 3: Analysis
Analyzing codebase...
[ ] 75%
Current file: src/auth/service.go
Current check: Security best practices
Checks completed:
✓ Code structure and organization
✓ Spec compliance
✓ Security vulnerabilities
→ Best practices adherence
○ Performance issues
○ Test coverage analysis
Step 4: Report Preview
Code Review Report


Overall Score: 7.5/10 ⚠️
Summary
- 8 issues found (3 high, 3 medium, 2 low)
- Spec compliance: 85%
- Test coverage: 72%
Critical Issues
1. Password hashing cost factor too low (8 vs 12)
2. Missing rate limiting on login endpoint
3. JWT tokens not using RS256
[... more content ...]
[Save Report] [View in Browser] [Export PDF]
Step 5: Save Report
Saving report...
✓ Created: reports/review-2026-01-07-auth.md
✓ Git commit: "Add review report for auth module"
Next steps:
• Address critical issues: factory change-order
• View report: cat reports/review-2026-01-07-auth.md
• Share with team: git push origin main
[Press any key to exit]

Technical Specification

Command Line Interface

```
# Review current directory
factory review

# Review specific directory
factory review src/auth

# Review specific files
factory review src/auth/handler.go src/auth/service.go

# Review against specific spec
factory review --spec contracts/specs/user-authentication.md src/auth

# Review with specific focus
factory review --focus security src/auth
factory review --focus performance src/auth

# Output options
factory review --output reports/custom-report.md src/auth
factory review --format json src/auth
factory review --format html src/auth

# Comparison mode
factory review --compare main..feature-branch src/auth

# Watch mode (continuous review)
factory review --watch src/auth
```

Configuration Options

```yaml
review:
  # Analysis settings
  llm:
    temperature: 0.3        # Lower for more consistent analysis
    max_tokens: 8192

  # What to analyze
  checks:
    - spec_compliance      # Check against specifications
    - security            # Security vulnerabilities
    - performance         # Performance issues
    - best_practices      # Language-specific best practices
    - code_smells         # Anti-patterns, code smells
    - test_coverage       # Test adequacy
    - documentation       # Comments, docs

  # Thresholds
  thresholds:
    overall_score: 7.0    # Fail if below
    spec_compliance: 80   # Minimum % compliance
    test_coverage: 70     # Minimum % coverage

  # Report settings
  report:
    format: markdown      # markdown, json, html
    include_code_snippets: true
    max_issues_per_category: 10
    severity_levels: [critical, high, medium, low, info]

  # Performance
  max_file_size: 10485760  # 10MB
  max_files: 100
  concurrent_analysis: 3
```

State Machine

```go
type ReviewState int

const (
    ReviewStateInit ReviewState = iota
    ReviewStateDiscovery
    ReviewStateSpecMatching
    ReviewStateAnalyzing
    ReviewStateReportGeneration
    ReviewStateSaving
    ReviewStateComplete
    ReviewStateError
)

type ReviewContext struct {
    // Input
    TargetPath    string
    TargetFiles   []string
    SpecFiles     []string
    Focus         []string

    // Discovered data
    Files         []FileInfo
    MatchedSpecs  []*Specification

    // Analysis results
    Issues        []Issue
    Metrics       *Metrics
    Suggestions   []Suggestion

    // State
    CurrentState  ReviewState
    CurrentFile   string
    Progress      float64

    // Services
    LLM           LLMService
    Git           GitService
}

type Issue struct {
    Severity    string     // critical, high, medium, low, info
    Category    string     // security, performance, spec_compliance, etc.
    Title       string
    Description string
    File        string
    Line        int
    Column      int
    Code        string     // Problematic code snippet
    Suggestion  string     // How to fix
    References  []string   // Links to docs, specs
}

type Metrics struct {
    OverallScore      float64
    SpecCompliance    float64
    TestCoverage      float64
    CodeQuality       float64
    SecurityScore     float64
    PerformanceScore  float64

    LinesOfCode       int
    FilesAnalyzed     int
```

```
    IssuesFound       map[string]int  // By severity
}
```

Analysis Workflow

```
func (ctx *ReviewContext) RunAnalysis() (*Report, error) {
    // 1. Discover files
    files, err := ctx.discoverFiles()
    if err != nil {
        return nil, err
    }
    ctx.Files = files

    // 2. Match specifications
    specs, err := ctx.matchSpecs()
    if err != nil {
        return nil, err
    }
    ctx.MatchedSpecs = specs

    // 3. Analyze files concurrently
    issues := make([]Issue, 0)
    semaphore := make(chan struct{}, ctx.Config.ConcurrentAnalysis)

    for _, file := range files {
        semaphore <- struct{}{}
        go func(f FileInfo) {
            defer func() { <-semaphore }()

            fileIssues := ctx.analyzeFile(f, specs)
            issues = append(issues, fileIssues...)
        }(file)
    }

    // Wait for all analyses to complete
    for i := 0; i < cap(semaphore); i++ {
        semaphore <- struct{}{}
    }

    ctx.Issues = issues

    // 4. Calculate metrics
    ctx.Metrics = ctx.calculateMetrics()

    // 5. Generate suggestions
    ctx.Suggestions = ctx.generateSuggestions()

    // 6. Build report
    report := ctx.buildReport()

    return report, nil
}

func (ctx *ReviewContext) analyzeFile(file FileInfo, specs []*Specification) []Issue {
    issues := make([]Issue, 0)

    // Read file content
    content, _ := os.ReadFile(file.Path)

    // Build analysis prompt
    prompt := ctx.buildAnalysisPrompt(file, content, specs)

    // Send to LLM
    response, _ := ctx.LLM.Generate(context.Background(), GenerateRequest{
        Prompt: prompt,
        System: "You are a code reviewer analyzing code against specifications.",
    })
```

```
    // Parse LLM response to extract issues
    fileIssues := ctx.parseIssues(response.Text, file.Path)
    issues = append(issues, fileIssues...)

    return issues
}
```

LLM Prompts

Code Analysis Prompt:

```
You are conducting a code review. Analyze the following code against the specification
and best practices.

**Specification:**
{spec_content}

**File:** {file_path}
**Language:** {language}

**Code:**
```{language}
{code_content}
```

Review Focus:

Provide your analysis in JSON format:
{
“issues”: [
{
“severity”: “high”,
“category”: “security”,
“title”: “Brief title”,
“description”: “Detailed description”,
“line”: 42,
“code”: “problematic code snippet”,
“suggestion”: “How to fix”,
“spec_reference”: “Which part of spec is violated”
}
],
“positive_observations”: [“What’s done well”],
“overall_assessment”: “General feedback”
}
```

```
#### Report Format

**reports/review-{date}-{target}.md:**
```markdown
---
type: code_review
target: src/auth
date: 2026-01-07T14:30:00Z
reviewer: factory-ai
specs_used:
  - contracts/specs/user-authentication.md
overall_score: 7.5
---

# Code Review Report: Authentication Module

**Generated:** 2026-01-07 14:30:00
**Target:** `src/auth/`
**Specifications:** user-authentication.md
**Files Analyzed:** 12
**Lines of Code:** 2,847

## Executive Summary

Overall Score: **7.5/10** ⚠️

The authentication module is generally well-implemented but has several areas requir-
ing attention, particularly around security configurations and spec compliance.

**Key Findings:**
- ✓ Code structure is clean and maintainable
- ✓ Error handling is comprehensive
- ⚠️ Security configurations don't match spec requirements
- ⚠️ Missing rate limiting implementation
- ⚠️ Test coverage below target (72% vs 80%)

## Metrics

| Metric | Score | Target | Status |
|--------|-------|--------|--------|
| Overall | 7.5/10 | 8.0/10 | ⚠️ Below target |
| Spec Compliance | 85% | 95% | ⚠️ Below target |
| Security | 6.8/10 | 9.0/10 | ⚠️ Needs improvement |
| Performance | 8.2/10 | 8.0/10 | ✓ Meets target |
| Code Quality | 8.5/10 | 8.0/10 | ✓ Exceeds target |
| Test Coverage | 72% | 80% | ⚠️ Below target |

## Issues by Severity

- **Critical:** 0
- **High:** 3
- **Medium:** 3
- **Low:** 2
- **Info:** 1

---

## Critical Issues

None found. ✓

---
```
```

```
## High Priority Issues

### 1. Password Hashing Cost Too Low

**Category:** Security / Spec Violation
**File:** `src/auth/service.go`
**Line:** 45

**Issue:**
bcrypt cost factor is set to 8, but the specification requires 12.

```go
// Current implementation
hash, err := bcrypt.GenerateFromPassword([]byte(password), 8)
```

Specification Says:

> Passwords hashed with bcrypt (cost factor 12)
> [user-authentication.md, Line 234]

Impact:
Lower cost factor makes passwords easier to brute force. With modern hardware, cost factor 8 can be broken significantly faster than cost factor 12.

Recommendation:

```go
// Use cost factor 12 as specified
hash, err := bcrypt.GenerateFromPassword([]byte(password), 12)
```

Effort: Low (5 minutes)
Priority: Fix immediately

---

2. Missing Rate Limiting on Login Endpoint

Category: Security / Spec Violation
File: `src/auth/handler.go`
Line: 78

Issue:
Login endpoint has no rate limiting implemented, contrary to specification.

```go
func (h *AuthHandler) Login(w http.ResponseWriter, r *http.Request) {
    // No rate limiting check here
    var req LoginRequest
    json.NewDecoder(r.Body).Decode(&req)
    // ... authentication logic
}
```

Specification Says:

> Failed login attempts are rate-limited (5 attempts per 15 minutes)
> [user-authentication.md, Line 89]
```

Impact:
Vulnerable to brute force attacks. Attackers can try unlimited password combinations.

Recommendation:
Implement middleware for rate limiting:

```
func (h *AuthHandler) Login(w http.ResponseWriter, r *http.Request) {
    // Check rate limit
    if !h.rateLimiter.Allow(r.Context(), getClientIP(r), "login", 5, 15*time.Minute) {
        http.Error(w, "Too many login attempts", http.StatusTooManyRequests)
        return
    }

    // ... rest of login logic
}
```

Effort: Medium (2-3 hours, requires Redis setup)
Priority: Fix this week

---

3. JWT Signature Algorithm Mismatch

Category: Security / Spec Violation
File: `src/auth/jwt.go`
Line: 23

Issue:
Using HS256 (symmetric) instead of RS256 (asymmetric) for JWT signing.

```
token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
```

Specification Says:

> JWT tokens signed with RS256
> [user-authentication.md, Line 242]

Impact:
HS256 uses a shared secret, meaning any service that validates tokens can also create them. RS256 with public/private keys is more secure for distributed systems.

Recommendation:

```
// Load private key
privateKey, _ := rsa.GenerateKey(rand.Reader, 2048)

// Sign with RS256
token := jwt.NewWithClaims(jwt.SigningMethodRS256, claims)
tokenString, _ := token.SignedString(privateKey)
```

Effort: Medium (4 hours, requires key management)
Priority: Fix this sprint

Medium Priority Issues

4. Token Expiry Time Inconsistency

Category: Spec Violation
File: `src/auth/jwt.go`
Line: 30

Issue:
Access token expiry is set to 30 minutes instead of 15 minutes as specified.

Current: 30 minutes
Spec: 15 minutes

Recommendation: Update to match specification for consistency.

---

5. Missing Email Verification Check on Login

Category: Spec Violation
File: `src/auth/service.go`
Line: 92

Issue:
Login allows unverified users to authenticate, contrary to specification.

Recommendation:

```
if !user.EmailVerified {
    return nil, errors.New("email not verified")
}
```

---

6. Inefficient Database Query

Category: Performance
File: `src/auth/repository.go`
Line: 56

Issue:
N+1 query problem when loading user refresh tokens.

Recommendation: Use JOIN or preload to fetch related data in one query.

Low Priority Issues

7. Missing JSDoc Comments

Category: Documentation
Files: Multiple

Issue: Public functions lack documentation comments.

Recommendation: Add comments for better maintainability.

---

8. Unused Import

Category: Code Quality
File: `src/auth/handler.go`
Line: 5

Issue: `time` package imported but not used.

Positive Observations ✓

1. **Clean Architecture:** Good separation of concerns (handler → service → repository)
2. **Error Handling:** Comprehensive error handling throughout
3. **Input Validation:** Strong validation on all user inputs
4. **Test Coverage:** Tests exist for core functionality (72% coverage)
5. **Code Style:** Consistent formatting and naming conventions

---

Recommendations

Immediate Actions (This Week)

1. Fix password hashing cost factor
2. Implement rate limiting on login endpoint
3. Add email verification check

Short Term (This Sprint)

1. Migrate to RS256 for JWT signing
2. Fix token expiry times
3. Optimize database queries
4. Increase test coverage to 80%+

Long Term (Next Quarter)

1. Consider adding multi-factor authentication
2. Implement token refresh rotation
3. Add comprehensive security logging
4. Set up automated security scanning

Spec Compliance Details

Requirement	Status	Notes
FR-1: User Registration	✓ Pass	Fully implemented
FR-2: User Login	⚠️ Partial	Missing rate limiting
FR-3: Password Reset	✓ Pass	Implemented correctly
FR-4: Token Management	⚠️ Partial	Wrong expiry times
NFR-1: Security	⚠️ Partial	Config mismatches
NFR-2: Performance	✓ Pass	Meets targets
NFR-3: Availability	✓ Pass	Good error handling

Overall Compliance: 85%

Test Coverage Analysis

```
src/auth/handler.go      78% ████████▒▒
src/auth/service.go      82% ████████▒▒
src/auth/repository.go   65% ███████▒▒▒
src/auth/jwt.go          45% ████▒▒▒▒▒▒
src/auth/models.go       95% ██████████
```

Missing Tests:
- JWT token validation edge cases
- Rate limiting behavior
- Account lockout logic
- Password reset token expiry

Appendix: Files Analyzed

1. src/auth/handler.go (234 lines)
2. src/auth/service.go (456 lines)
3. src/auth/repository.go (189 lines)
4. src/auth/jwt.go (123 lines)
5. src/auth/models.go (67 lines)
6. src/auth/middleware.go (89 lines)
7. src/auth/validator.go (145 lines)
8. src/auth/errors.go (45 lines)
9. src/auth/handler_test.go (312 lines)

10. src/auth/service_test.go (267 lines)
11. src/auth/repository_test.go (178 lines)
12. src/auth/jwt_test.go (89 lines)

Total: 2,847 lines of code

Report Generated by: Code-Factory v1.0.0
Analysis Time: 2m 34s
LLM Model: llama3.2:latest

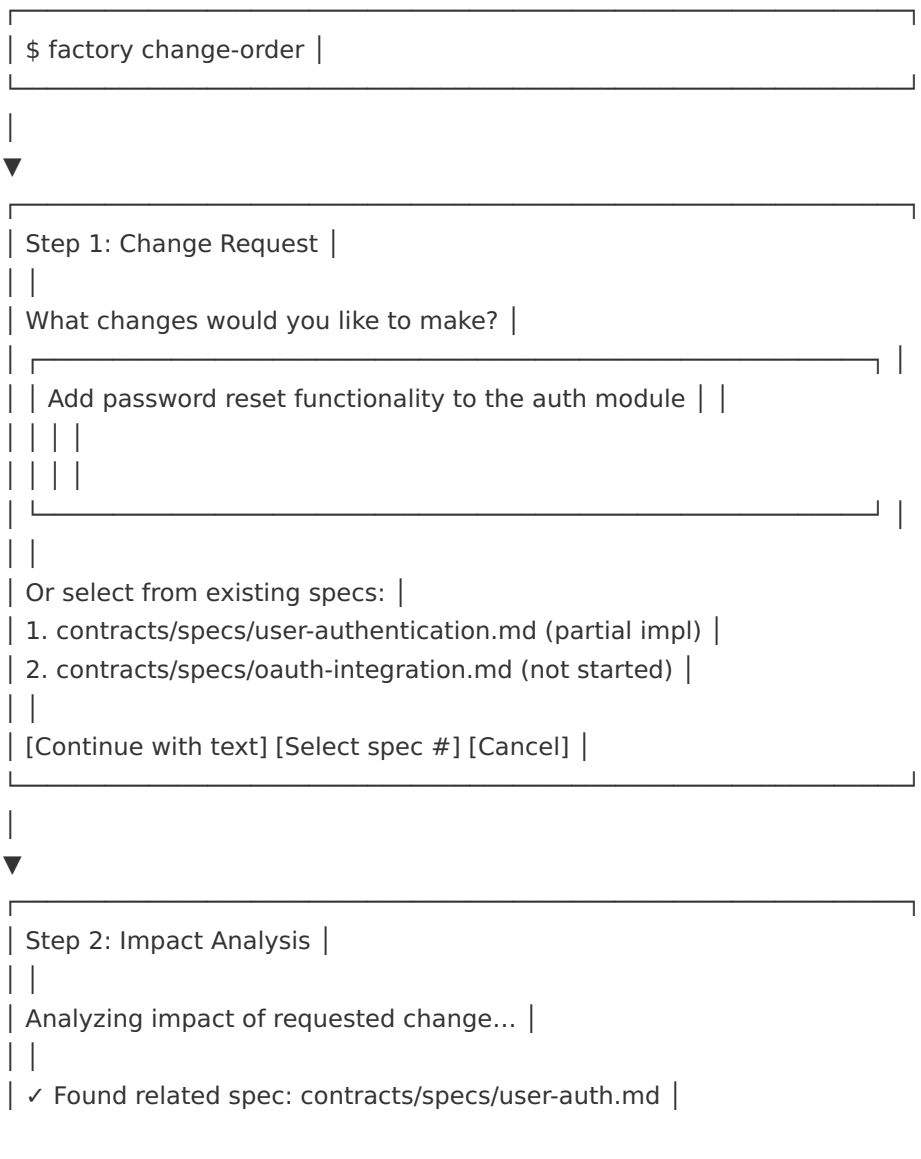
```
---

## CHANGE_ORDER Mode

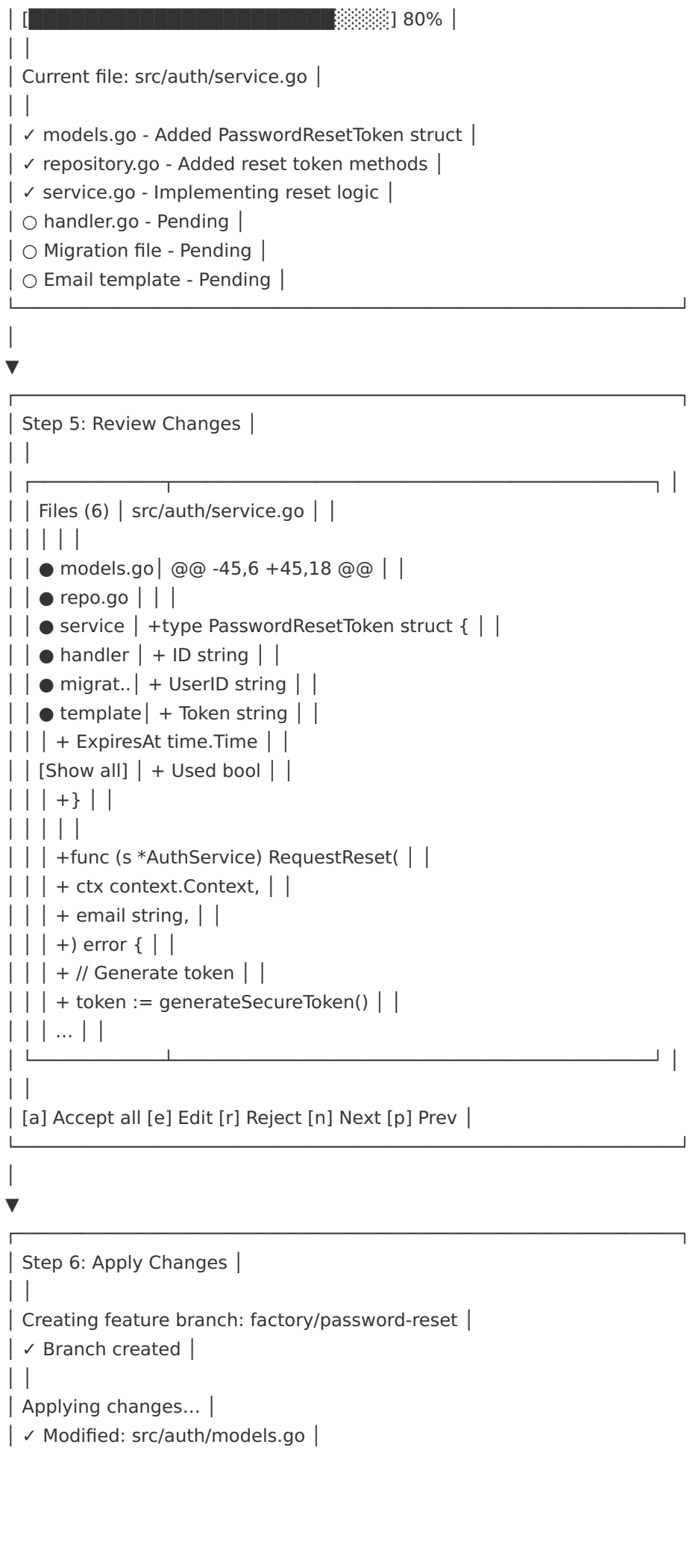
#### Purpose

Implement code changes based on specifications or change requests. Generates code,
creates feature branches, and optionally submits pull requests to GitHub.

#### User Flow
```



```
│ ✓ Identified affected files: │
│ - src/auth/service.go (will modify) │
│ - src/auth/handler.go (will modify) │
│ - src/auth/repository.go (will add method) │
│ - src/auth/models.go (will add struct) │
│ │
│ ✓ Estimated changes: ~200 lines │
│ ✓ Complexity: Medium │
│ ✓ Risk level: Low │
│ │
│ [Continue] [Adjust scope] [Cancel] │
└──────────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 3: Change Plan │
│ │
│ ┌──────────────────────────────────────────────────────────────┐ │
│ │ # Implementation Plan │ │
│ │ │ │
│ │ ## Changes Overview │ │
│ │ Add password reset functionality with token-based │ │
│ │ email workflow. │ │
│ │ │ │
│ │ ## Steps │ │
│ │ 1. Add PasswordResetToken model │ │
│ │ 2. Add database migration │ │
│ │ 3. Implement reset request handler │ │
│ │ 4. Implement reset confirmation handler │ │
│ │ 5. Add email template │ │
│ │ 6. Add tests │ │
│ │ │ │
│ │ ## Files to Modify │ │
│ │ - src/auth/models.go (+25 lines) │ │
│ │ - src/auth/repository.go (+45 lines) │ │
│ │ - src/auth/service.go (+67 lines) │ │
│ │ - src/auth/handler.go (+38 lines) │ │
│ │ - migrations/006_password_reset.sql (new file) │ │
│ │ - templates/password-reset-email.html (new file) │ │
│ └──────────────────────────────────────────────────────────────┘ │
│ │
│ [Approve plan] [Modify plan] [Cancel] │
└──────────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 4: Code Generation │
│ │
│ Generating code changes... │
│ │
```


[████████████████████████▒▒▒▒▒] 80%
Current file: src/auth/service.go
✓ models.go - Added PasswordResetToken struct
✓ repository.go - Added reset token methods
✓ service.go - Implementing reset logic
○ handler.go - Pending
○ Migration file - Pending
○ Email template - Pending
Step 5: Review Changes
Files (6)
src/auth/service.go
● models.go
● repo.go
● service
● handler
● migrat..
● template
[Show all]
@@ -45,6 +45,18 @@
+type PasswordResetToken struct {
+ ID string
+ UserID string
+ Token string
+ ExpiresAt time.Time
+ Used bool
+}
+func (s *AuthService) RequestReset(
+ ctx context.Context,
+ email string,
+) error {
+ // Generate token
+ token := generateSecureToken()
...
[a] Accept all [e] Edit [r] Reject [n] Next [p] Prev
Step 6: Apply Changes
Creating feature branch: factory/password-reset
✓ Branch created
Applying changes...
✓ Modified: src/auth/models.go

```
│ ✓ Modified: src/auth/repository.go │
│ ✓ Modified: src/auth/service.go │
│ ✓ Modified: src/auth/handler.go │
│ ✓ Created: migrations/006_password_reset.sql │
│ ✓ Created: templates/password-reset-email.html │
│ │
│ ✓ Git commit: "feat: Add password reset functionality" │
│ │
│ [Continue] │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│ Step 7: GitHub Integration (Optional) │
│ │
│ Create pull request on GitHub? │
│ │
│ Branch: factory/password-reset │
│ Base: main │
│ Title: Add password reset functionality │
│ Description: [Auto-generated from spec] │
│ │
│ Labels: enhancement, code-factory │
│ Reviewers: [Auto-detected from CODEOWNERS] │
│ │
│ [Create PR] [Push without PR] [Stay local] │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│ Success! │
│ │
│ ✓ Changes applied to branch: factory/password-reset │
│ ✓ Pull request created: #123 │
│ │
│ Pull Request: │
│ https://github.com/user/repo/pull/123 │
│ │
│ Next steps: │
│ • Review the PR and make any adjustments │
│ • Run tests: make test │
│ • Merge when ready │
│ │
│ [View PR in browser] [Continue working] [Exit] │
└──────────────────────────────────────────────────────────────┘
```

```
### Technical Specification

#### Command Line Interface

```bash
# Interactive mode
factory change-order

# With description
factory change-order --description "Add password reset feature"

# From spec file
factory change-order --spec contracts/specs/user-auth.md

# Specific files only
factory change-order --files src/auth/service.go,src/auth/handler.go \
    --description "Fix rate limiting"

# Auto-approve (dangerous!)
factory change-order --auto-approve --description "Update dependencies"

# Dry run (show plan only)
factory change-order --dry-run --description "Refactor auth module"

# GitHub options
factory change-order --no-pr --description "..."  # Don't create PR
factory change-order --draft-pr --description "..."  # Create draft PR
```

Configuration

```yaml
change_order:
  # Code generation
  llm:
    temperature: 0.2        # Lower for more deterministic code
    max_tokens: 8192

  # Git workflow
  git:
    branch_prefix: "factory/"
    commit_message_template: "{type}: {title}"
    auto_stage: true

  # GitHub PR
  github:
    create_pr: ask          # ask, always, never
    pr_template: ".github/pull_request_template.md"
    default_labels: ["code-factory", "automated"]
    request_reviews: true
    auto_assign_codeowners: true

  # Safety checks
  safety:
    require_approval: true
    backup_before_apply: true
    run_tests_before_commit: false
    max_files_changed: 20
    max_lines_changed: 1000
```

```

RESCUE Mode

Purpose

Debug issues, diagnose problems, and generate solutions when things go wrong. Acts as an AI pair programmer to help understand and fix errors, failed tests, or unexpected behavior.

User Flow


$ factory rescue
Step 1: Problem Description
What's the problem?
Tests are failing after I added password reset
Or provide:
• Error message / stack trace
• Test output
• Log file
[Continue] [Paste error] [Upload log] [Cancel]
Step 2: Context Gathering
Gathering context...
✓ Found error in: src/auth/service_test.go
✓ Read failing test: TestPasswordReset
✓ Read implementation: src/auth/service.go
✓ Checked recent git history
✓ Found related spec: contracts/specs/user-auth.md
Would you like me to run the tests to see the error?
[Yes, run tests] [No, I'll provide output] [Skip]
Step 3: Problem Analysis
Analyzing the problem...
[████████████████████████] 100%
Problem Diagnosis
Root Cause
The `PasswordResetToken` model is missing a database migration, causing tests to fail when trying to create the table.
Evidence
1. Test error: "table password_reset_tokens does not exist"
2. No migration file found for this table
3. Model defined in models.go but DB not updated
Contributing Factors

```
- Migration was not created in change-order
- Tests run against clean database

## Confidence: High (95%)

[Show solution] [Request more analysis] [Cancel]
```

▼

```
Step 4: Solution

# Proposed Solution

Create the missing database migration.

## Action Items
1. Create migration file:
   migrations/006_password_reset_tokens.sql

2. Run migration in test setup

3. Verify tests pass

## Migration SQL
```sql
CREATE TABLE password_reset_tokens (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL,
    token_hash VARCHAR(255) NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Testing
After applying: `go test ./src/auth/...`

[Apply solution] [Edit] [Explain more] [Try different fix]
```

▼

```
Step 5: Apply & Verify

Applying solution...
  ✓ Created: migrations/006_password_reset_tokens.sql
  ✓ Git commit: "fix: Add missing password reset migration"

Running tests...
  ✓ TestPasswordReset passed
  ✓ TestPasswordResetExpiry passed
  ✓ All tests in auth package passed

Problem solved! ✓

[View details] [Exit]
```

Technical Specification

Command Line Interface

```
# Interactive mode
factory rescue

# With problem description
factory rescue --problem "Tests failing in auth module"

# With error message
factory rescue --error "$(cat error.log)"

# From failing test
factory rescue --test TestPasswordReset

# With specific files for context
factory rescue --files src/auth/service.go,src/auth/service_test.go

# Auto-apply fix (if confident)
factory rescue --auto-fix --problem "..."

# Explain only (no fix)
factory rescue --explain-only --problem "..."
```

Configuration

```yaml
rescue:
  # Analysis
  llm:
    temperature: 0.3
    max_tokens: 8192

  # Context gathering
  context:
    max_files: 20
    include_tests: true
    include_specs: true
    git_history_limit: 10

  # Solution generation
  solutions:
    max_solutions: 3      # Offer multiple solutions
    require_confidence: 0.7  # Minimum confidence to suggest fix

  # Testing
  testing:
    run_tests_after_fix: true
    test_timeout: 5m

  # Safety
  safety:
    create_backup: true
    require_approval: true
```

Cross-Mode Features

Shared Capabilities

1. Context Management

All modes have access to project context:
- Specifications from `/contracts/`
- Previous reports from `/reports/`
- Git history
- Current codebase state

2. LLM Integration

All modes use the same LLM interface:
- Streaming responses for better UX
- Token counting and limits
- Error handling and retries
- Response caching

3. Git Integration

All modes interact with git:
- Auto-commit changes
- Create feature branches
- Detect conflicts
- Rollback capability

4. GitHub Integration (Optional)

When enabled:
- Create PRs
- Add comments
- Link to issues
- Sync with remote

5. Undo/Rollback

All modes support rollback:

```
factory undo              # Undo last operation
factory undo --list       # Show undo history
factory undo --to=abc123  # Rollback to specific commit
```

6. History

Track all operations:

```
factory history           # Show recent operations
factory history --mode intake  # Filter by mode
factory history --export history.json  # Export
```

Keyboard Shortcuts

Global (all modes):
- `Ctrl+C` - Cancel operation
- `?` - Show help

- `Ctrl+Z` - Undo last action
- `Tab` - Autocomplete
- `↑↓` - Navigate history
- `/` - Search

Mode-specific:
- `e` - Edit (in review/change-order)
- `d` - Show diff
- `r` - Regenerate (intake/rescue)
- `a` - Accept all (change-order)

Revision History

Version	Date	Changes	Author
1.0.0	2026-01-07	Initial specification	Code-Factory Team