Factory Bootstrap Specification

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

Table of Contents

Overview

The Factory Bootstrap process is the first-run experience for Code-Factory. It must be:
- **Effortless**: Single command `factory init` should handle everything
- **Beautiful**: Charm.sh TUI with clear progress indicators
- **Intelligent**: Auto-detect configurations and minimize user input
- **Resilient**: Graceful fallbacks for offline/no-GitHub scenarios
- **Secure**: Proper secret handling with user consent

Design Principles

1. **Zero to Hero in 60 Seconds**: From install to first spec generation
2. **Progressive Disclosure**: Show complexity only when needed
3. **Safety First**: All destructive actions require explicit confirmation
4. **Teach as You Go**: Provide context and help inline
5. **Fail Gracefully**: Never leave user in broken state

One-Click Onboarding Flow

Entry Point: `factory init`

```
$ factory init
```

Flow Diagram

```
┌──────────────────────────────────────────────────────────────┐
│ 1. Welcome Screen                                            │
│    - ASCII art logo                                          │
│    - Version info                                            │
│    - What will happen next                                   │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
│ 2. Environment Detection                                     │
│    - Check if ~/.config/factory/ exists                      │
│    - Detect internet connectivity                            │
│    - Check for existing config.yaml                          │
│    - Detect if running in GitHub Codespaces/Gitpod         │
└──────────────────────────────────────────────────────────────┘
                              ↓
              ┌───────────────┴──────────────┐
              │                              │
         ┌────▼─────┐                 ┌──────▼─────────┐
         │ First Run │                │ Existing Config │
         └────┬─────┘                 └──────┬─────────┘
              │                              │
              │                        ┌─────▼──────┐
              │                        │ Show Summary│
              │                        │ Ask Reconfig│
              │                        └─────┬──────┘
              │                              │
              └────────────┬─────────────────┘
                           ↓
┌──────────────────────────────────────────────────────────────┐
│ 3. LLM Configuration                                         │
│    - Auto-detect Ollama (port 11434)                         │
│    - If found: Test connection, show model list              │
│    - If not: Offer BYOK options (OpenAI, Anthropic, etc.)  │
│    - Store choice in config                                  │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
│ 4. GitHub Integration (Optional)                             │
│    - Ask: "Connect to GitHub for seamless workflow?"         │
│    - Yes → OAuth flow (see section below)                     │
│    - No → Continue with local-only mode                       │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
│ 5. Project Initialization                                    │
│    - Ask for project name (default: current directory)       │
│    - Create /contracts and /reports directories              │
│    - Generate initial README.md in /contracts                │
│    - Create .gitignore with sensible defaults                │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
│ 6. Success Screen                                            │
│    - Show configured components                              │
│    - Next steps guide                                        │
│    - Quick start commands                                    │
└──────────────────────────────────────────────────────────────┘
```

Detailed Step Specifications

Step 1: Welcome Screen

UI Layout:

```
╔═══════════════════════════════════════════════════════════════╗
║                                                               ║
║     _____ ___  ____  _____     _____  _  ___ _____ ___  ______  __
║    / ____|/ _ \|  _ \|  ____|    |  __ \  /\  / ____|_   _/ __ \|  _ \ \ / /
║   | |    | | | | | | | |__  _____| |__ | /  \| |      | || |  | | |__) \ V /
║   | |    | | | | | | |  __||_____|  __|| / /\ \ |      | || |  | |  _  / | |
║   | |____| |_| | |_| | |____      | |   | / ____ \ |____| || |__| | | \ \ | |
║    \_____|\___/|____/|______|     |_|   |_/    \_\_____|_____\____/|_|  \_\|_|
║                                                               ║
║                    Spec-Driven Software Factory               ║
║                         Version 1.0.0                         ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝

Welcome to Code-Factory! 👋

This setup wizard will configure your development factory in about a minute.

What we'll do:
  • Configure your AI assistant (Ollama or bring-your-own-key)
  • Optionally connect to GitHub for seamless collaboration
  • Set up your project structure
  • Create your first contract template

Press Enter to continue, or Ctrl+C to exit...
```

Implementation Notes:
- Use `lipgloss` for styling
- Animate the welcome text (fade-in effect)
- Show version from build metadata
- Keyboard: Enter = Continue, Ctrl+C = Exit, ? = Help

Step 2: Environment Detection

Display:

```
Checking your environment...

  ✓ Internet connectivity: Online
  ✓ Configuration directory: Creating ~/.config/factory/
  ✓ Runtime environment: Local machine
  ○ Existing configuration: None found

Detected environment: Local Development
```

Detection Logic:

```go
type EnvironmentInfo struct {
    IsOnline          bool
    HasExistingConfig bool
    ConfigPath        string
    IsCodespaces      bool
    IsGitpod          bool
    IsCI              bool
}

func DetectEnvironment() (*EnvironmentInfo, error) {
    env := &EnvironmentInfo{}

    // Check internet
    env.IsOnline = checkInternet("https://api.github.com/zen")

    // Check for existing config
    configPath := filepath.Join(os.UserHomeDir(), ".config", "factory", "config.yaml")
    env.ConfigPath = configPath
    env.HasExistingConfig = fileExists(configPath)

    // Detect cloud environments
    env.IsCodespaces = os.Getenv("CODESPACES") == "true"
    env.IsGitpod = os.Getenv("GITPOD_WORKSPACE_ID") != ""
    env.IsCI = os.Getenv("CI") == "true"

    return env, nil
}
```

Behavior:
- If `HasExistingConfig == true`: Show summary screen and ask “Reconfigure? [y/N]”
- If offline: Skip GitHub integration, proceed with local-only
- If in Codespaces/Gitpod: Show warning about ephemeral storage

Step 3: LLM Configuration

3.1 Ollama Auto-Detection

Display:

```
Configuring AI Assistant...

Checking for local Ollama installation...
  ✓ Ollama detected at http://localhost:11434
  ✓ Connection successful

Available models:
  • llama3.2:latest (recommended)
  • codellama:latest
  • mistral:latest

Select your preferred model:
> llama3.2:latest
  codellama:latest
  mistral:latest
  [Other model...]
```

Detection Code:

```go
func DetectOllama() (*OllamaInfo, error) {
    // Try common ports
    ports := []int{11434, 11435}

    for _, port := range ports {
        url := fmt.Sprintf("http://localhost:%d/api/tags", port)
        resp, err := http.Get(url)
        if err != nil {
            continue
        }
        defer resp.Body.Close()

        if resp.StatusCode == 200 {
            var result struct {
                Models []struct {
                    Name string `json:"name"`
                    Size int64  `json:"size"`
                } `json:"models"`
            }
            json.NewDecoder(resp.Body).Decode(&result)

            return &OllamaInfo{
                Available: true,
                Endpoint:  fmt.Sprintf("http://localhost:%d", port),
                Models:    result.Models,
            }, nil
        }
    }

    return &OllamaInfo{Available: false}, nil
}
```

3.2 BYOK (Bring Your Own Key) Flow

If Ollama not detected:

```
No local Ollama installation found.

You can:
  1. Install Ollama now (recommended for privacy)
     → Visit https://ollama.ai/download
  2. Use an external LLM provider

Select provider:
> OpenAI (GPT-4, GPT-3.5)
  Anthropic (Claude 3)
  Google (Gemini)
  Azure OpenAI
  Custom endpoint

[Enter API Key]: ••••••••••••••••••••••••••••••••

Testing connection...
  ✓ API key valid
  ✓ Model access confirmed: gpt-4o
```

Configuration Storage:

```yaml
# ~/.config/factory/config.yaml
llm:
  provider: "ollama"  # or "openai", "anthropic", "google", "azure", "custom"
  endpoint: "http://localhost:11434"
  model: "llama3.2:latest"
  # For BYOK:
  # api_key_ref: "factory.llm.openai.key"  # Reference to secret in keyring
```

Secret Handling:
- Never store API keys in plain text config
- Use OS keyring (keychain on macOS, Secret Service on Linux, Credential Manager on Windows)
- Library: `github.com/zalando/go-keyring`

```go
import "github.com/zalando/go-keyring"

func StoreAPIKey(provider, key string) error {
    service := "factory.llm." + provider
    return keyring.Set(service, "api_key", key)
}

func GetAPIKey(provider string) (string, error) {
    service := "factory.llm." + provider
    return keyring.Get(service, "api_key")
}
```

Step 4: GitHub Integration

4.1 Prompt Screen

```
GitHub Integration (Optional)

Connecting to GitHub enables:
  • Automatic PR creation from CHANGE_ORDER mode
  • Direct spec commits to your repositories
  • Seamless team collaboration
  • Issue tracking integration

This requires:
  • GitHub OAuth authentication
  • Installing the Code-Factory GitHub App
  • Granting repository access

Would you like to connect GitHub? [y/N]: _
```

4.2 OAuth Flow (see detailed section below)

Step 5: Project Initialization

```
Setting up your project...

Project name (default: code-factory): my-awesome-app
Project path: /home/user/projects/my-awesome-app

Creating structure:
  ✓ /contracts/ - Your specifications live here
  ✓ /reports/ - Generated reports and analysis
  ✓ README.md - Project documentation
  ✓ .gitignore - Sensible defaults

Initializing git repository...
  ✓ Git initialized
  ✓ Initial commit created

Project ready! 🎉
```

Generated Files:

`/contracts/README.md`:

```markdown
# Project Contracts

This directory contains your project specifications in plain markdown.

## Structure

- **specs/** - Feature specifications
- **architecture/** - System architecture documents
- **decisions/** - Architecture Decision Records (ADRs)

## Getting Started

1. Create a new spec: `factory intake`
2. Review existing code: `factory review`
3. Request changes: `factory change-order`

## Spec Format

See `template.md` for the standard format.
```

`/.gitignore`:

```
# Factory outputs
/reports/
*.factory.tmp

# OS
.DS_Store
Thumbs.db

# Secrets
.env
*.key
*.pem

# IDE
.vscode/
.idea/
*.swp
*.swo
```

Step 6: Success Screen

```
╔═══════════════════════════════════════════════════════════════╗
║                    Setup Complete! 🎉                          ║
╚═══════════════════════════════════════════════════════════════╝

Your Code-Factory is ready to use!

Configuration:
  • AI Provider: Ollama (llama3.2:latest)
  • GitHub: Connected (@username)
  • Project: /home/user/projects/my-awesome-app

Next Steps:

  1. Create your first specification:
     $ factory intake

  2. Review existing code:
     $ factory review src/

  3. Request a code change:
     $ factory change-order

  4. Get help anytime:
     $ factory help

Tips:
  • All specs are stored in /contracts/ as markdown
  • Reports are generated in /reports/
  • Use `factory config` to reconfigure anytime

Happy building! 🏭

Press any key to exit...
```

GitHub OAuth & App Installation

Overview

GitHub integration uses **OAuth Device Flow** (ideal for CLI) and requires installing the **Code-Factory GitHub App** for repository access.

Why GitHub App vs Personal Access Token?

Feature	GitHub App	PAT
User consent	✅ Per-repo	❌ All-or-nothing
Token rotation	✅ Automatic	❌ Manual
Audit trail	✅ Detailed	⚠️ Limited
Revocation	✅ Per-installation	❌ All access
Rate limits	✅ Higher	⚠️ Lower

OAuth Device Flow

Why Device Flow?
- No need for localhost callback server
- Works in SSH sessions, containers, remote machines
- User-friendly: shows code + URL

Flow Diagram:

```
┌──────────┐                                      ┌──────────┐
│ Factory  │                                      │  GitHub  │
│   CLI    │                                      │   API    │
└────┬─────┘                                      └────┬─────┘
     │                                                 │
     │  POST /login/device/code                        │
     │  (client_id, scope)                             │
     │────────────────────────────────────────────────>│
     │                                                 │
     │  { device_code, user_code, verification_uri }   │
     │<────────────────────────────────────────────────│
     │                                                 │
     │  [Display to user]                              │
     │  "Visit https://github.com/login/device"        │
     │  "Enter code: ABCD-1234"                        │
     │                                                 │
     │  [User opens browser and enters code]           │
     │                        │                        │
     │                        └───────────────────────>│
     │                                                 │ [User authorizes]
     │                                                 │
     │  POST /login/oauth/access_token (poll)          │
     │  (device_code)                                  │
     │────────────────────────────────────────────────>│
     │                                                 │
     │  { access_token }                               │
     │<────────────────────────────────────────────────│
     │                                                 │
     │  GET /user (verify)                             │
     │────────────────────────────────────────────────>│
     │                                                 │
     │  { login, name, ... }                           │
     │<────────────────────────────────────────────────│
     │                                                 │
```

Implementation

Step 1: Initiate Device Flow

```go
type DeviceCodeResponse struct {
    DeviceCode      string `json:"device_code"`
    UserCode        string `json:"user_code"`
    VerificationURI string `json:"verification_uri"`
    ExpiresIn       int    `json:"expires_in"`
    Interval        int    `json:"interval"`
}

func InitiateDeviceFlow(clientID string) (*DeviceCodeResponse, error) {
    data := url.Values{}
    data.Set("client_id", clientID)
    data.Set("scope", "repo read:user")

    resp, err := http.PostForm(
        "https://github.com/login/device/code",
        data,
    )
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    var result DeviceCodeResponse
    if err := json.NewDecoder(resp.Body).Decode(&result); err != nil {
        return nil, err
    }

    return &result, nil
}
```

Step 2: Display to User

```
Connecting to GitHub...

1. Visit: https://github.com/login/device
2. Enter this code: ABCD-1234

╔══════════════════════════════════╗
║                                  ║
║         ABCD-1234               ║
║                                  ║
║    (Code copied to clipboard)    ║
║                                  ║
╚══════════════════════════════════╝

Waiting for authorization... [Press Ctrl+C to cancel]

[ Spinner animation ]
```

UI Enhancement:
- Copy code to clipboard automatically
- Optionally open browser (ask user first)
- Show countdown timer (usually 15 minutes to complete)

Step 3: Poll for Token

```go
func PollForAccessToken(clientID, deviceCode string, interval int) (string, error) {
    ticker := time.NewTicker(time.Duration(interval) * time.Second)
    defer ticker.Stop()

    timeout := time.After(15 * time.Minute)

    for {
        select {
        case <-timeout:
            return "", errors.New("authorization timeout")
        case <-ticker.C:
            token, err := checkAuthorization(clientID, deviceCode)
            if err == nil {
                return token, nil
            }
            // Continue polling on "authorization_pending"
            // Return error on other errors
        }
    }
}

func checkAuthorization(clientID, deviceCode string) (string, error) {
    data := url.Values{}
    data.Set("client_id", clientID)
    data.Set("device_code", deviceCode)
    data.Set("grant_type", "urn:ietf:params:oauth:grant-type:device_code")

    resp, err := http.PostForm(
        "https://github.com/login/oauth/access_token",
        data,
    )
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    var result struct {
        AccessToken string `json:"access_token"`
        Error       string `json:"error"`
    }
    json.NewDecoder(resp.Body).Decode(&result)

    if result.Error == "authorization_pending" {
        return "", errors.New("pending")
    } else if result.Error != "" {
        return "", errors.New(result.Error)
    }

    return result.AccessToken, nil
}
```

Step 4: Verify Token & Get User Info

```go
func GetAuthenticatedUser(token string) (*GitHubUser, error) {
    req, _ := http.NewRequest("GET", "https://api.github.com/user", nil)
    req.Header.Set("Authorization", "Bearer "+token)
    req.Header.Set("Accept", "application/vnd.github.v3+json")

    client := &http.Client{Timeout: 10 * time.Second}
    resp, err := client.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        return nil, fmt.Errorf("authentication failed: %d", resp.StatusCode)
    }

    var user GitHubUser
    if err := json.NewDecoder(resp.Body).Decode(&user); err != nil {
        return nil, err
    }

    return &user, nil
}
```

Success Display:

```
✓ Authorization successful!

Authenticated as: @username (John Doe)
Email: john@example.com

Permissions granted:
  • Read user profile
  • Access public and private repositories
```

GitHub App Installation

When to Install:

- After successful OAuth (if user wants repo access)
- Can be deferred to first use of GitHub features

Flow:

```
To enable repository operations, you need to install the Code-Factory GitHub App.

This will allow Code-Factory to:
  • Create branches and PRs
  • Read repository contents
  • Create commits
  • Add comments and reviews

Installation URL:
https://github.com/apps/code-factory/installations/new

1. Open the URL above
2. Select repositories to grant access
3. Complete installation

[Open in browser] [Skip for now]

Waiting for installation confirmation...
```

Verification:

```go
func CheckAppInstallation(token string) (bool, []string, error) {
    req, _ := http.NewRequest(
        "GET",
        "https://api.github.com/user/installations",
        nil,
    )
    req.Header.Set("Authorization", "Bearer "+token)
    req.Header.Set("Accept", "application/vnd.github.v3+json")

    client := &http.Client{Timeout: 10 * time.Second}
    resp, err := client.Do(req)
    if err != nil {
        return false, nil, err
    }
    defer resp.Body.Close()

    var result struct {
        Installations []struct {
            AppSlug string `json:"app_slug"`
            Account struct {
                Login string `json:"login"`
            } `json:"account"`
        } `json:"installations"`
    }
    json.NewDecoder(resp.Body).Decode(&result)

    for _, inst := range result.Installations {
        if inst.AppSlug == "code-factory" {
            // Get accessible repositories
            repos, _ := getInstallationRepos(token, inst.Account.Login)
            return true, repos, nil
        }
    }

    return false, nil, nil
}
```

Required Permissions and Scopes

OAuth Scopes:

```
repo                 # Full repository access
read:user            # User profile info
read:org             # Organization membership
workflow             # GitHub Actions access (optional)
```

GitHub App Permissions:

```yaml
# app.yml (for GitHub App configuration)
default_permissions:
  contents: write        # Read/write repository contents
  pull_requests: write   # Create and manage PRs
  issues: write          # Create and manage issues
  metadata: read         # Read repository metadata

events:
  - pull_request
  - push
  - issue_comment
```

Secret Storage

Configuration File:

```yaml
# ~/.config/factory/config.yaml
github:
  enabled: true
  username: "johndoe"
  token_ref: "factory.github.oauth.token"  # Keyring reference
  app_installed: true
  installation_id: 12345678
```

Keyring Storage:

```go
// Store OAuth token
keyring.Set("factory.github", "oauth.token", accessToken)

// Store installation token (if using GitHub App API)
keyring.Set("factory.github", "installation.token", installationToken)
```

Security Considerations:
- Tokens are NEVER written to config file or logs
- Use OS-native secure storage
- Implement token refresh logic
- Handle revocation gracefully

LLM Configuration

Detection Strategy

Priority Order:

1. **Ollama** (localhost:11434) - Check first
2. **Environment variables** - OPENAI_API_KEY, ANTHROPIC_API_KEY, etc.
3. **Existing config** - ~/.config/factory/config.yaml
4. **Interactive prompt** - Ask user

Ollama Integration

Detection:

```go
func DetectOllama(ctx context.Context) (*OllamaConfig, error) {
    endpoints := []string{
        "http://localhost:11434",
        "http://127.0.0.1:11434",
        os.Getenv("OLLAMA_HOST"),
    }

    for _, endpoint := range endpoints {
        if endpoint == "" {
            continue
        }

        // Try to connect
        url := endpoint + "/api/tags"
        req, _ := http.NewRequestWithContext(ctx, "GET", url, nil)

        client := &http.Client{Timeout: 2 * time.Second}
        resp, err := client.Do(req)
        if err != nil {
            continue
        }
        defer resp.Body.Close()

        if resp.StatusCode == 200 {
            var result struct {
                Models []OllamaModel `json:"models"`
            }
            json.NewDecoder(resp.Body).Decode(&result)

            return &OllamaConfig{
                Endpoint: endpoint,
                Available: true,
                Models: result.Models,
            }, nil
        }
    }

    return &OllamaConfig{Available: false}, nil
}

type OllamaModel struct {
    Name       string    `json:"name"`
    Size       int64     `json:"size"`
    ModifiedAt time.Time `json:"modified_at"`
    Details    struct {
        Format string `json:"format"`
        Family string `json:"family"`
    } `json:"details"`
}
```

Model Selection UI:

```
Select Ollama Model:

> ● llama3.2:latest          7.4 GB   [recommended]
  ○ codellama:latest         7.4 GB   [for code]
  ○ mistral:latest           4.1 GB
  ○ llama2:latest           3.8 GB
  ○ deepseek-coder:latest   6.7 GB

  [↑↓] Navigate  [Enter] Select  [/] Search  [q] Quit

Tip: codellama is optimized for code generation
```

Configuration:

```yaml
llm:
  provider: ollama
  endpoint: http://localhost:11434
  model: llama3.2:latest
  options:
    temperature: 0.7
    num_ctx: 8192      # Context window
    num_predict: 2048  # Max tokens
```

BYOK (Bring Your Own Key) Providers

Supported Providers:

1. **OpenAI**
   - Models: gpt-4o, gpt-4, gpt-3.5-turbo
   - API: https://api.openai.com/v1
   - Key format: sk-...

2. **Anthropic**
   - Models: claude-3-opus, claude-3-sonnet
   - API: https://api.anthropic.com/v1
   - Key format: sk-ant-...

3. **Google (Gemini)**
   - Models: gemini-pro, gemini-ultra
   - API: https://generativelanguage.googleapis.com/v1
   - Key format: AIza...

4. **Azure OpenAI**
   - Models: Custom deployments
   - API: https://{resource}.openai.azure.com
   - Auth: API Key or Azure AD

5. **Custom Endpoint**
   - Any OpenAI-compatible API
   - Examples: LM Studio, LocalAI, vLLM

Interactive Configuration:

```
LLM Provider: OpenAI

API Key: •••••••••••••••••••••••••••••••••••••••• [Show]

Testing connection...
  ✓ Connection successful
  ✓ Available models: gpt-4o, gpt-4, gpt-3.5-turbo

Select model:
> gpt-4o (recommended)
  gpt-4
  gpt-3.5-turbo

Advanced Options: [Configure]
  • Temperature: 0.7
  • Max tokens: 2048
  • Top P: 1.0

[Save] [Cancel]
```

Validation:

```go
func ValidateOpenAIKey(apiKey string) error {
    req, _ := http.NewRequest(
        "GET",
        "https://api.openai.com/v1/models",
        nil,
    )
    req.Header.Set("Authorization", "Bearer "+apiKey)

    client := &http.Client{Timeout: 10 * time.Second}
    resp, err := client.Do(req)
    if err != nil {
        return fmt.Errorf("connection failed: %w", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode == 401 {
        return errors.New("invalid API key")
    } else if resp.StatusCode != 200 {
        return fmt.Errorf("API error: %d", resp.StatusCode)
    }

    return nil
}
```

Configuration Persistence

Config File Structure:

```yaml
# ~/.config/factory/config.yaml
version: "1.0"

llm:
  provider: "openai"  # ollama, openai, anthropic, google, azure, custom

  # For Ollama
  endpoint: "http://localhost:11434"
  model: "llama3.2:latest"

  # For BYOK (secrets stored in keyring)
  api_key_ref: "factory.llm.openai.key"
  model: "gpt-4o"

  # Common options
  options:
    temperature: 0.7
    max_tokens: 2048
    top_p: 1.0

github:
  enabled: true
  username: "johndoe"
  token_ref: "factory.github.oauth.token"
  app_installed: true

project:
  name: "my-awesome-app"
  path: "/home/user/projects/my-awesome-app"
  contracts_dir: "contracts"
  reports_dir: "reports"
```

Secret Management

Principle: Never Store Secrets in Plain Text

Storage Locations (in order of preference):

1. **OS Keyring** - System-native secure storage (primary)
2. **Environment Variables** - For CI/CD and containers
3. **Encrypted File** - Fallback for systems without keyring

OS Keyring Implementation

Library: `github.com/zalando/go-keyring`

Operations:

```go
package secrets

import (
    "fmt"
    "github.com/zalando/go-keyring"
)

const ServiceName = "factory"

// Store a secret
func Store(key, value string) error {
    return keyring.Set(ServiceName, key, value)
}

// Retrieve a secret
func Get(key string) (string, error) {
    value, err := keyring.Get(ServiceName, key)
    if err == keyring.ErrNotFound {
        return "", fmt.Errorf("secret not found: %s", key)
    }
    return value, err
}

// Delete a secret
func Delete(key string) error {
    return keyring.Delete(ServiceName, key)
}

// List all secret keys
func List() ([]string, error) {
    // Note: go-keyring doesn't support listing
    // We maintain a list in config file
    return []string{}, nil
}
```

Secret References in Config:

```yaml
llm:
  provider: openai
  api_key_ref: "llm.openai.key"  # Reference to keyring entry

github:
  token_ref: "github.oauth.token"
```

Fallback: Encrypted File

For systems without keyring support (e.g., headless Linux servers):

```go
import (
    "crypto/aes"
    "crypto/cipher"
    "crypto/rand"
    "encoding/base64"
    "io"
)

// Encrypt using AES-GCM with key derived from machine ID + user password
func EncryptSecret(plaintext, password string) (string, error) {
    key := deriveKey(password, getMachineID())

    block, err := aes.NewCipher(key)
    if err != nil {
        return "", err
    }

    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return "", err
    }

    nonce := make([]byte, gcm.NonceSize())
    io.ReadFull(rand.Reader, nonce)

    ciphertext := gcm.Seal(nonce, nonce, []byte(plaintext), nil)
    return base64.StdEncoding.EncodeToString(ciphertext), nil
}

// Store in ~/.config/factory/secrets.enc
```

Environment Variable Fallback

Priority:

1. Keyring
2. Environment variable
3. Prompt user

```go
func GetLLMAPIKey(provider string) (string, error) {
    // Try keyring first
    key, err := secrets.Get("llm." + provider + ".key")
    if err == nil {
        return key, nil
    }

    // Try environment variable
    envVar := strings.ToUpper(provider) + "_API_KEY"
    if key := os.Getenv(envVar); key != "" {
        return key, nil
    }

    // Prompt user
    return promptForAPIKey(provider)
}
```

User Consent & Confirmation

Before Storing Secrets:

```
Store API Key Securely?

Your OpenAI API key will be stored in:
  • macOS: Keychain
  • Linux: Secret Service (gnome-keyring/kwallet)
  • Windows: Credential Manager

This allows Factory to use the API without re-prompting.

You can remove it anytime with: factory config reset

[Yes, store securely] [No, use environment variable] [Cancel]
```

On Access:

First time accessing a stored secret:

```
Factory needs to access your OpenAI API key from Keychain.

[Allow] [Deny]

Tip: You can revoke access anytime in System Preferences
```

User Experience & Feedback

Progress Indicators

Spinner Components:

```go
import "github.com/charmbracelet/bubbles/spinner"

type model struct {
    spinner  spinner.Model
    status   string
    progress float64
}

func (m model) View() string {
    return fmt.Sprintf(
        "%s %s... %.0f%%",
        m.spinner.View(),
        m.status,
        m.progress*100,
    )
}
```

Progress Bar:

```
Setting up Code-Factory...

[████████████████░░░░░░░░] 65%

Current step: Configuring LLM provider...
```

Error Handling

Principle: Never leave user confused or in broken state

Error Display:

```
╔═══════════════════════════════════════════════════════════════╗
║  ⚠️  Setup Encountered an Issue                                ║
╚═══════════════════════════════════════════════════════════════╝

Error: Could not connect to Ollama

Possible causes:
  • Ollama is not running
  • Ollama is running on a different port
  • Firewall is blocking connection

Suggested actions:
  1. Start Ollama: systemctl start ollama
  2. Check status: ollama list
  3. Try manual endpoint: factory config llm --endpoint=...

[Retry] [Use different provider] [Abort setup]

Need help? Visit: https://docs.code-factory.dev/troubleshooting
```

Error Recovery:

```go
func handleSetupError(err error) Action {
    switch {
    case isNetworkError(err):
        return Action{
            Message: "Network connection failed",
            Suggestions: []string{
                "Check internet connectivity",
                "Try again in a moment",
                "Continue with offline mode",
            },
            CanRetry: true,
        }

    case isAuthError(err):
        return Action{
            Message: "Authentication failed",
            Suggestions: []string{
                "Check your credentials",
                "Regenerate API key",
                "Try different authentication method",
            },
            CanRetry: true,
        }

    default:
        return Action{
            Message: "Unexpected error occurred",
            Suggestions: []string{
                "Save error log",
                "Report issue on GitHub",
                "Contact support",
            },
            CanRetry: false,
        }
    }
}
```

Contextual Help

Inline Tips:

```
💡 Tip: Press '?' at any time for help

💡 Tip: You can reconfigure anytime with `factory config`

💡 Tip: All specs are version-controlled. Use git to track changes.
```

Help Dialog (Press '?'):

```
╔══════════════════════════════════════════════════════════════╗
║ Help: LLM Configuration                                      ║
╚══════════════════════════════════════════════════════════════╝

Ollama vs BYOK:

Ollama (Recommended):
  ✓ Free and open source
  ✓ Runs locally (private)
  ✓ No API costs
  ✗ Requires local resources
  ✗ Limited to smaller models

BYOK (OpenAI, Claude, etc.):
  ✓ Access to latest models
  ✓ No local resources needed
  ✓ Generally faster
  ✗ Costs per API call
  ✗ Data sent to third party

Recommendation:
  • Development: Use Ollama (free, private)
  • Production: Use BYOK (better quality)

[Close] [Learn more online]
```

Confirmation Before Destructive Actions

Example:

```
⚠️  Warning: Overwrite Existing Configuration?

Current configuration will be replaced:
  • LLM Provider: Ollama → OpenAI
  • GitHub: Disconnected → Connected

This action cannot be undone.

Type 'yes' to confirm: _
```

Fallback Scenarios

Scenario 1: Offline Mode

Detection:

```go
func IsOnline() bool {
    _, err := net.DialTimeout("tcp", "github.com:443", 3*time.Second)
    return err == nil
}
```

Behavior:

```
⚠️  No internet connection detected

Running in offline mode:
  ✓ Ollama (local) will be used
  ✗ GitHub integration unavailable
  ✗ External LLM providers unavailable

Continue with offline mode? [Y/n]
```

Limitations:
- No GitHub OAuth
- No external LLM APIs
- Must use local Ollama

Scenario 2: No Ollama + Offline

Cannot proceed:

```
╔═══════════════════════════════════════════════════════════════╗
║ ⚠️   Setup Cannot Continue                                     ║
╚═══════════════════════════════════════════════════════════════╝

Code-Factory requires an LLM provider, but:
  • No internet connection (cannot use external APIs)
  • Ollama not detected (no local LLM)

To continue, either:
  1. Install Ollama: https://ollama.ai/download
  2. Connect to internet and use external provider

[Retry detection] [Exit setup]
```

Scenario 3: GitHub OAuth Failure

Fallback Options:

```
GitHub OAuth failed

Alternative authentication methods:
  1. Personal Access Token (classic)
     → Create at: https://github.com/settings/tokens
     → Requires: repo, read:user

  2. GitHub CLI (gh)
     → Use existing authentication: gh auth status

  3. Skip GitHub integration
     → Continue with local-only mode

Select option [1-3]: _
```

Scenario 4: Keyring Not Available

Detection & Fallback:

```
func IsKeyringAvailable() bool {
    err := keyring.Set("factory.test", "test", "test")
    if err != nil {
        return false
    }
    keyring.Delete("factory.test", "test")
    return true
}
```

Fallback Flow:

```
⚠️  Secure keyring not available

Your system doesn't have a keyring service (common in headless servers).

Alternative storage options:
  1. Environment variables (recommended for servers)
     → Export OPENAI_API_KEY=sk-...

  2. Encrypted file (less secure)
     → Requires master password

  3. Config file (NOT recommended - plain text)
     → Only for testing/development

Select option [1-3]: _
```

Scenario 5: Running in CI/CD

Detection:

```
func IsCI() bool {
    return os.Getenv("CI") == "true" ||
           os.Getenv("GITHUB_ACTIONS") == "true" ||
           os.Getenv("GITLAB_CI") == "true"
}
```

Behavior:

```
CI/CD environment detected

Skipping interactive setup.

Configure via environment variables:
  export FACTORY_LLM_PROVIDER=openai
  export OPENAI_API_KEY=sk-...
  export FACTORY_GITHUB_TOKEN=ghp_...

Or use config file:
  factory init --config=factory.yaml --non-interactive

[Exit]
```

Technical Implementation

CLI Structure

```
cmd/factory/
├── main.go               # Entry point
├── commands/
│   ├── init.go           # Bootstrap command
│   ├── intake.go
│   ├── review.go
│   ├── change_order.go
│   ├── rescue.go
│   └── config.go         # Reconfiguration
└── tui/
    └── init/
        ├── model.go      # Bubble Tea model
        ├── welcome.go    # Welcome screen
        ├── env.go        # Environment detection
        ├── llm.go        # LLM configuration
        ├── github.go     # GitHub integration
        └── success.go    # Success screen
```

State Machine

```go
type InitState int

const (
    StateWelcome InitState = iota
    StateEnvDetection
    StateLLMConfig
    StateGitHubIntegration
    StateProjectInit
    StateSuccess
)

type InitModel struct {
    state      InitState
    env        *EnvironmentInfo
    llm        *LLMConfig
    github     *GitHubConfig
    project    *ProjectConfig
    err        error
}

func (m InitModel) Update(msg tea.Msg) (tea.Model, tea.Cmd) {
    switch msg := msg.(type) {
    case tea.KeyMsg:
        switch msg.String() {
        case "ctrl+c", "q":
            return m, tea.Quit
        case "enter":
            return m.advance()
        }
    }
    return m, nil
}

func (m InitModel) advance() (tea.Model, tea.Cmd) {
    switch m.state {
    case StateWelcome:
        m.state = StateEnvDetection
        return m, detectEnvironment
    case StateEnvDetection:
        m.state = StateLLMConfig
        return m, detectLLM
    // ... etc
    }
}
```

Configuration Schema

```go
// pkg/config/types.go
type Config struct {
    Version string         `yaml:"version"`
    LLM     LLMConfig      `yaml:"llm"`
    GitHub  *GitHubConfig  `yaml:"github,omitempty"`
    Project ProjectConfig  `yaml:"project"`
}

type LLMConfig struct {
    Provider  string                 `yaml:"provider"`  // ollama, openai, etc.
    Endpoint  string                 `yaml:"endpoint,omitempty"`
    Model     string                 `yaml:"model"`
    APIKeyRef string                 `yaml:"api_key_ref,omitempty"`
    Options   map[string]interface{} `yaml:"options,omitempty"`
}

type GitHubConfig struct {
    Enabled         bool   `yaml:"enabled"`
    Username        string `yaml:"username"`
    TokenRef        string `yaml:"token_ref"`
    AppInstalled    bool   `yaml:"app_installed"`
    InstallationID  int64  `yaml:"installation_id,omitempty"`
}

type ProjectConfig struct {
    Name         string `yaml:"name"`
    Path         string `yaml:"path"`
    ContractsDir string `yaml:"contracts_dir"`
    ReportsDir   string `yaml:"reports_dir"`
}
```

API Calls Summary

GitHub API:

1. `POST /login/device/code` - Initiate OAuth device flow
2. `POST /login/oauth/access_token` - Poll for access token
3. `GET /user` - Get authenticated user info
4. `GET /user/installations` - Check GitHub App installation
5. `GET /user/repos` - List user repositories (optional)

Ollama API:

1. `GET /api/tags` - List available models
2. `POST /api/generate` - Test generation (optional)

External LLM APIs:

1. OpenAI: `GET /v1/models` - Validate API key
2. Anthropic: `POST /v1/messages` - Test request
3. Google: Similar validation calls

Error Codes

```go
const (
    ErrOK = 0

    // Environment errors (1xx)
    ErrNoInternet = 101
    ErrNoKeyring = 102
    ErrNoWriteAccess = 103

    // LLM errors (2xx)
    ErrNoLLMProvider = 201
    ErrOllamaNotFound = 202
    ErrInvalidAPIKey = 203
    ErrLLMConnectionFailed = 204

    // GitHub errors (3xx)
    ErrGitHubOAuthFailed = 301
    ErrGitHubAppNotInstalled = 302
    ErrGitHubPermissionDenied = 303

    // Project errors (4xx)
    ErrInvalidProjectPath = 401
    ErrProjectAlreadyExists = 402
    ErrGitInitFailed = 403
)
```

Testing Requirements

Unit Tests

Coverage Requirements:
- Environment detection: 90%+
- OAuth flow: 85%+
- LLM configuration: 85%+
- Secret management: 95%+

Example Tests:

```go
// internal/bootstrap/env_test.go
func TestDetectEnvironment(t *testing.T) {
    tests := []struct {
        name     string
        setup    func()
        expected EnvironmentInfo
    }{
        {
            name: "codespaces detected",
            setup: func() {
                os.Setenv("CODESPACES", "true")
            },
            expected: EnvironmentInfo{
                IsCodespaces: true,
                IsOnline: true,
            },
        },
        // ... more tests
    }

    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            tt.setup()
            defer os.Clearenv()

            result, err := DetectEnvironment()
            assert.NoError(t, err)
            assert.Equal(t, tt.expected, result)
        })
    }
}
```

Integration Tests

Mock GitHub API:

```go
// Use httptest to mock GitHub API responses
func TestOAuthDeviceFlow(t *testing.T) {
    server := httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.
Request) {
        if r.URL.Path == "/login/device/code" {
            json.NewEncoder(w).Encode(DeviceCodeResponse{
                DeviceCode: "test-device-code",
                UserCode: "ABCD-1234",
                VerificationURI: "https://github.com/login/device",
            })
        }
    }))
    defer server.Close()

    // Test OAuth flow with mock server
}
```

Mock Ollama:

```
func TestOllamaDetection(t *testing.T) {
    server := httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.
Request) {
        json.NewEncoder(w).Encode(map[string]interface{}{
            "models": []map[string]string{
                {"name": "llama3.2:latest"},
            },
        })
    }))
    defer server.Close()

    // Override Ollama endpoint for test
    config, err := DetectOllama(server.URL)
    assert.NoError(t, err)
    assert.True(t, config.Available)
}
```

E2E Tests

Scenarios:

1. **Fresh install, Ollama available**
   - Should detect Ollama
   - Should offer model selection
   - Should complete successfully

2. **Fresh install, no Ollama, with API key env var**
   - Should detect API key from environment
   - Should validate connection
   - Should complete successfully

3. **Fresh install, offline**
   - Should detect offline mode
   - Should show appropriate message
   - Should guide user on next steps

4. **Re-initialization**
   - Should detect existing config
   - Should ask for confirmation
   - Should preserve selected settings

Manual Testing Checklist

- [ ] Fresh install on macOS
- [ ] Fresh install on Linux (Ubuntu)
- [ ] Fresh install on Windows
- [ ] Re-init with existing config
- [ ] Offline mode
- [ ] Ollama auto-detection
- [ ] Each BYOK provider (OpenAI, Anthropic, etc.)
- [ ] GitHub OAuth device flow
- [ ] GitHub App installation
- [ ] Error recovery flows
- [ ] Help dialogs

- [ ] Keyboard navigation
- [ ] Screen resize handling

Appendix

Sample Configuration Files

~/.config/factory/config.yaml:

```yaml
version: "1.0.0"

llm:
  provider: ollama
  endpoint: http://localhost:11434
  model: llama3.2:latest
  options:
    temperature: 0.7
    num_ctx: 8192

github:
  enabled: true
  username: johndoe
  token_ref: github.oauth.token
  app_installed: true
  installation_id: 12345678

project:
  name: my-awesome-app
  path: /home/user/projects/my-awesome-app
  contracts_dir: contracts
  reports_dir: reports

ui:
  color_scheme: auto  # auto, light, dark
  animations: true
```

Dependencies

go.mod:

```
module github.com/ssdajoker/Code-Factory

go 1.21

require (
    github.com/charmbracelet/bubbletea v0.25.0
    github.com/charmbracelet/bubbles v0.18.0
    github.com/charmbracelet/lipgloss v0.9.1
    github.com/zalando/go-keyring v0.2.3
    github.com/spf13/cobra v1.8.0
    gopkg.in/yaml.v3 v3.0.1
    github.com/google/go-github/v57 v57.0.0
    golang.org/x/oauth2 v0.15.0
)
```

References

- GitHub OAuth Device Flow (https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps#device-flow)
- GitHub App Installation (https://docs.github.com/en/apps/maintaining-github-apps/installing-github-apps)
- Ollama API Documentation (https://github.com/ollama/ollama/blob/main/docs/api.md)
- Charm.sh Bubble Tea Tutorial (https://github.com/charmbracelet/bubbletea/tree/master/tutorials)

Revision History

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