Helix CLI Specification v2.0

As the guide for examples of implementation of existing solutions we provide git submodules with source codes of OpenCode, Charm Crush, Ollama Code, Codename Goose, Gemini Code, Qwen Code. It will be done full analysis of the approaches that have been used in each of those projects and then used for our needs - architectural best ideas, whole feature implementations, or just partial on top of which we will be building further and do significant fixes and improvements.

Executive Summary

Helix CLI is an AI-powered coding agent designed for terminal environments that enables seamless interaction with LLMs for software development tasks. The system supports multiple LLM providers and offers comprehensive development capabilities including code generation, testing, debugging, refactoring, and deployment.

Technical Feasibility Analysis

Achievable Features:

- Multi-provider LLM integration (Llama.cpp, Ollama, OpenRouter, etc.)
- Decoupled Go-based architecture with bash orchestration
- Terminal UI and headless CLI modes
- Basic project management and session handling
- Standard testing and debugging workflows

Potential No-Go Areas (Require Significant R&D):

- **Real AI QA execution** - Requires sophisticated AI testing frameworks
- **Multi-machine distributed computing** - Complex synchronization and network architecture
- **Figma/Penpot bidirectional sync** - Complex API integrations and design system mapping
- **Automatic 100% test coverage** - May not be practical for all project types
- **Real-time collaborative editing** - Requires advanced conflict resolution

1. Core Architecture

1.1 System Design Principles

- **Decoupled Components**: Each module operates independently as a Go program
- **Bash Orchestration**: Components bound together via bash scripts for flexibility
- **Reusability**: Generics and interfaces for maximum code reuse
- **Multi-Platform**: Support for Linux, macOS, BSDs, Windows

1.2 Component Architecture

Core Components

1. **LLM Integration Layer**

    - Llama.cpp API wrapper
    - Ollama API integration
    - OpenRouter API client
    - Provider-agnostic interface

2. **User Interface Layer**

    - Terminal UI (TUI) with rich interaction
    - Headless CLI for automation
    - REST API server

3. **Data Management**

    - PostgreSQL with SQLCipher encryption
    - Session and project state management
    - Configuration persistence

4. **Processing Engines**

    - Planner: Project analysis and documentation
    - Builder: Code generation and modification
    - Refactorer: Code optimization and restructuring
    - Tester: Comprehensive testing execution
    - Debugger: Issue detection and resolution

1.3 Deployment Flavors

- **Terminal UI**: Interactive development environment
- **Headless CLI**: Scripting and automation
- **REST API**: Multi-client support
- **Docker Container**: Portable deployment

2. LLM Integration & Management

2.1 Supported Providers

- **Local**: Llama.cpp, Ollama
- **Remote APIs**: DeepSeek, Qwen, Claude, Gemini, Grok, Mistral
- **Aggregators**: OpenRouter, HuggingFace, NVIDIA

2.2 Model Management

- **Dynamic Hardware Analysis**: Automatic model selection based on system capabilities
- **Model Installation**: Automated download, conversion, and optimization
- **Fallback Strategy**: Configurable model fallback chains
- **Performance Monitoring**: Resource usage tracking and alerts

2.3 API Architecture

- **Unified Interface**: Provider-agnostic API layer
- **Capability Discovery**: Dynamic detection of model features

- **Context Management**: Efficient handling of conversation context
- **Tool Calling**: Support for LLM tool execution

3. User Interface & Experience

3.1 Terminal UI Features

- **ASCII Art Header**: Dynamic project branding
- **Mode Switching**: Seamless transitions between development modes
- **Interactive Confirmation**: Granular control over agent actions
- **Theme System**: Customizable color schemes
- **Session Management**: Pause, resume, detach capabilities

3.2 Navigation & Hierarchy

- **Project Structure**: Project → Module → Submodule hierarchy
- **Context Switching**: Quick navigation between project levels
- **History Management**: Request history with favorites and grouping
- **Rollback System**: Safe project state restoration

3.3 Configuration System

- **Global Configuration**: `~/.config/Helix/helix.json`
- **Project Configuration**: `Helix.md` files
- **Session Settings**: Dynamic configuration per session
- **Import/Export**: Configuration and database portability

4. Development Workflows

4.1 Core Development Modes

4.1.1 Planning Mode

- **Deep Analysis**: Comprehensive project assessment
- **Documentation Generation**: Structured planning documents
- **Workflow Design**: Dynamic development flow creation
- **Resource Estimation**: Hardware and time requirements

4.1.2 Building Mode

- **Parallel Code Generation**: Multiple worker coordination
- **Module-based Development**: Independent module construction
- **Conflict Resolution**: Automated merge and synchronization
- **Quality Assurance**: Built-in code quality checks

4.1.3 Testing Mode

- **Comprehensive Testing**: Unit, integration, automation, E2E
- **Quality Scanning**: SonarQube and Snyk integration

- **Issue Resolution**: Automatic problem detection and fixing
- **Report Generation**: Detailed test execution reports

4.1.4 Refactoring Mode

- **Code Optimization**: Performance and structure improvements
- **Pattern Application**: Best practice implementation
- **Dependency Management**: Library and API updates
- **Migration Support**: Framework and language transitions

4.1.5 Debugging Mode

- **Issue Detection**: Automated problem identification
- **Root Cause Analysis**: Systematic error investigation
- **Solution Generation**: Context-aware fix proposals
- **Validation**: Solution testing and verification

4.2 Advanced Modes

4.2.1 Design Mode

- **Figma Integration**: Design import/export via API
- **Penpot Support**: Open-source design tool integration
- **Design System Sync**: Bidirectional design-code synchronization
- **Prototype Generation**: Interactive prototype creation

4.2.2 Diagram Mode

- **UML Generation**: Automatic diagram creation
- **Multiple Formats**: Draw.io, Mermaid.js, PNG, JPEG, PDF
- **Project Visualization**: Architecture and flow diagrams
- **Documentation Integration**: Markdown-compatible outputs

4.2.3 Deployment Mode

- **Profile Management**: Configurable deployment targets
- **Cloud Integration**: Multi-provider deployment support
- **Environment Configuration**: Stage-specific settings
- **Rollback Capability**: Safe deployment reversal

4.2.4 Porting Mode

- **Technology Mapping**: Cross-platform code translation
- **Dependency Resolution**: Equivalent library identification
- **Pattern Adaptation**: Architecture pattern conversion
- **Validation Testing**: Ported code verification

5. Session & Collaboration

5.1 Session Management

- **Named Sessions**: Organized development contexts
- **Multi-User Access**: SSH and REST API participation
- **Session Monitoring**: Real-time progress tracking
- **Resource Control**: CPU, memory, and model usage

5.2 Collaboration Features

- **QR Code Joining**: Easy session access
- **Broadcast Discovery**: Automatic service detection
- **Real-time Updates**: WebSocket-based synchronization
- **Access Control**: Credential-based sharing

5.3 Distributed Computing

- **Multi-Machine Coordination**: Parallel execution across nodes
- **Load Balancing**: Optimal resource utilization
- **Fault Tolerance**: Graceful failure handling
- **Synchronization**: Consistent state management

6. Memory & State Management

6.1 Memory Architecture

- **Multi-layer Storage**: Project → Session → Request hierarchy
- **Context Preservation**: LLM conversation state management
- **Duplicate Detection**: Intelligent request recognition
- **Progressive Learning**: Continuous improvement from history

6.2 Database Design

- **PostgreSQL Backend**: Robust data persistence
- **Encrypted Storage**: SQLCipher protection
- **Efficient Querying**: Optimized data access patterns
- **Backup/Restore**: Data portability and recovery

7. Testing & Quality Assurance

7.1 Testing Strategy

- **100% Coverage Goal**: Comprehensive test implementation
- **Multi-level Testing**: Unit, integration, system, E2E
- **Automated Execution**: CI/CD pipeline integration
- **Quality Gates**: SonarQube and Snyk enforcement

7.2 Quality Tools Integration

- **SonarQube Scanning**: Code quality and security analysis
- **Snyk Security**: Vulnerability detection and remediation

- **Dockerized Execution**: Isolated testing environments
- **Report Generation**: Actionable quality insights

8. Security & Access Control

8.1 Security Features

- **Encrypted Configuration**: Secure credential storage
- **API Key Management**: Multiple key support with rotation
- **Access Control**: Granular permission system
- **Network Security**: HTTP/3 and secure protocols

8.2 Account Management (Optional)

- **User Authentication**: Credential-based access
- **Permission System**: Role-based access control
- **Session Security**: Secure multi-user sessions
- **Audit Logging**: Comprehensive activity tracking

9. Performance & Optimization

9.1 Hardware Optimization

- **Dynamic Resource Allocation**: Adaptive model selection
- **Performance Monitoring**: Real-time resource tracking
- **Bottleneck Detection**: System limitation identification
- **Optimization Suggestions**: Performance improvement recommendations

9.2 Scalability Features

- **Horizontal Scaling**: Multi-instance deployment
- **Load Distribution**: Efficient work allocation
- **Caching Strategy**: Performance optimization
- **Resource Limits**: Configurable usage boundaries

10. Implementation Roadmap

10.1 Phase 1: Core Foundation

- Basic LLM integration (Llama.cpp, Ollama)
- Terminal UI framework
- Project configuration system
- Basic building and testing modes

10.2 Phase 2: Advanced Features

- Multi-provider support
- Session management
- Advanced testing integration
- Refactoring and debugging modes

10.3 Phase 3: Collaboration & Scale

- Multi-user sessions
- Distributed computing
- Design and diagram modes
- Deployment capabilities

10.4 Phase 4: Enterprise Features

- Account management
- Advanced security
- Performance optimization
- Comprehensive documentation

11. Technical Considerations

11.1 Implementation Challenges

- **Model Compatibility**: Ensuring consistent behavior across providers
- **State Synchronization**: Maintaining consistency in distributed environments
- **Performance Optimization**: Efficient resource utilization
- **Error Handling**: Graceful failure recovery

11.2 Integration Points

- **Version Control**: Git integration for project management
- **CI/CD Systems**: Automated testing and deployment
- **Cloud Providers**: Multi-cloud deployment support
- **Design Tools**: Figma and Penpot API integration

12. Success Metrics

12.1 Performance Indicators

- **Response Time**: Sub-second command execution
- **Resource Efficiency**: Optimal hardware utilization
- **Test Coverage**: 100% automated test implementation
- **User Satisfaction**: High adoption and retention rates

12.2 Quality Standards

- **Code Quality**: SonarQube passing scores
- **Security**: Snyk vulnerability-free status
- **Reliability**: 99.9% uptime for core features
- **Documentation**: Comprehensive and up-to-date

This specification represents a comprehensive roadmap for Helix CLI development, balancing ambitious features with practical implementation considerations. The modular architecture allows for incremental development while maintaining a clear vision for the complete system.