Helix CLI Specs

The Helix CLI is the AI coding agent built for the terminal to access and work (code, test, debug, etc.) with LLMs. Primary provider is LLama CPP, however it supports all other options such as work with remote APIs - DeepSeek, Qwen, Claude, Gemini, Grok, Mistral, etc. It supports various other providers like OpenRouter, Ollama, Nvidia, HuggingFace, etc.

It is powerfull programming tool which intracts with end user and makes possible flawless code implementation on the projects, refactopring, debugging, etc.

Important power feature

The CLI makes possible for end users to build and run local LLMs and use them for code generation, testing, debugging, etc. However, most important is that Toolking, Thinking and other LLM capabilities may be flwalessly used on low end computer systems (by using 7B LLMs). On more capable machines, dynamically detrmined, more powerful LLMs may be used - 32b and similar.

Architecture

All project / application components are fully decoupled programs which are bound together with proper bash scripts. Each comonent can be used separately or as a part of larger application bound together by bash scripts. All programs are writtent in Go language.

Some of the components that we are mentioning now are:

- LLama CPP API for communication with LLMs executed locally by LLama CPP
- OpenRouter API for communication with OpenRouter
- Ollama API for communication with Ollama
- DeepSeek API for communication with DeepSeek
- Qwen API for communication with Qwen
- Claude API for communication with Claude
- Other APIs for communication with other LLMs
- ASII art generator - will generate the header for the project that will appear in CLI UI when user starts the application
- Pure CLI mode (Headless CLI mode)
- CLI UI mode
- REST API mode (run as server)
- Postgres database component (run as server) - will be used by CLI UI and REST API, storing memories, progres, etc.
- Debugger component - make sure that debugging of software for issues detection is possible using LLMs. It investigates and collects information about software execution for further fixes and improvements or new features development.
- Planner - Does the deep analysis and perfroms the planning with written documentation as result of this efforts - user inputs are required as always
- Builder - Builds the code using LLMs, extends the features, fixes issues, etc.
- Refactorer - Refactors the code using LLMs, extends the features, fixes issues, etc.

- Tester - Tests the code using LLMs, extends the features, fixes issues, etc. Tester introduces to the codebase of single or multiple modules of the projects the full 100% code coverage with several testing types: unit, integration, full automation, end-to-end, etc. Last two types will always be executed on real devices or emulators or apps ran by real AI QA! There are two more kind of tests that will always be performed: SoanrQube deep scanning and Snyk. Both performed with dockerized (docker compose) free versions of the two tools. Full reports will be generated and any discovered issues fully fixed.

LLama CPP and Ollama support

Under the hood API will contain its local instance (codebase) of LLama CPP and Ollama. It will be used for communication with LLMs via exposed API. Once it is ready it will be used for communication with LLMs. To be ready it has to download source code, do optimal build for current OS, do the configuration and optimization. API will expose downloadable LLMs for both LLama CPP and Ollama. It will download LLMs from all possible sources such as HuggingFace, OpenRouter, DeepSeek, Qwen, Claude, etc. Downloaded LLMs will be converted to proper format if that is needed so LLama CPP and Ollama will be able to use them. User will trigger the installation of the LLM and it will be ready for use. Installation will first make sure to detect hardware of host machine - all mandatory capabilites, then it will build local project instance of LLama CPP and Ollama and will download the LLM to be used. All optimization performed and LLM downloaded and started.

All utils have to be decoupled and other components for installation, configuration, download and running the LLMs. Make sure we have proper generics and interfaces which are widely reusable! Resusability principle has to be followed with every project component!

Others APIs - VLLM, Local LLM, etc

Same principle we described for LLama CPP and Ollama will be applied to other platforms and their LLMs.

CLI (Terminal UI) features

- Testing
- Refactoring
- Planning
- Builder (Code generation) - Creates the code using LLMs. It spwans multiple builders which will split the codebase into modules and generate the code for each module. Each worker (LLM builder) will generate the code for single module. All this will be done in parallel and synchornized. Default number of workers (builders) is one. User can set the number of workers to be used. There will be another builder running - the coordinator which will accept work done by workers and put it properly so no conflicts happen!
- Refactorer - Performs codebase refactoring
- Tester - Executes all available tests, brings up all required apps and emulators as precoditions, generates reports, fixes discovered issues and finall does the SonqrQube and Snyk scanning.
- Debugger
- Diagrams (creation of UML diagrams and others from the Project and its docs and source code in supported formats - drawio, png, jpeg, uml, pdf, etc.) Default is drawio.
- Deployment (deploying the project to the cloud or any other defined endpoint)
- Default project is the current directory

- If no Helix.md is available user will be asked to allow creation of it
    - Creation will scan the whole project up to the details and write to the file
    - Format and rules are same as for AGENTS.md or CLAUDE.md
    - If any existing file is found of other agents like CLaude or OpenCode it will be used to support the project's Helix.md generation
    - We will detect all sub-projects and modules. For each of it individual agent file (Helix.md) will be created
- User is able to switch between projects and modules (into deeper hierarchy and to go back)
- User is able to send command sequentially (requests) to the CLI agent which will execute them
- Using tab user can switch between the modes (Builder and others)
- User will be asked about every action that agent is going to do such as write into the file, execute the system command, util, etc. There will be option to skip, accept, accept for the whole session, accept forever for the current project
- All configuration and choices are going to be written in json configuration file: ~/.config/Helix/helix.json
- User will be able to switch modles by choosing provider (LLama CPP, DeepSeek, Qwen, Claude, etc.) and proper model (this will be stored in helix.json file as setting)
- All this operation will be dopne by proper set of commands sent to the agent, we will use a exact same command like Crush has with some differences: we will be able to paste from clipboard request that we want to send; we are going to be able to repeat all requests sent to agent. All requests will be stored in Postgres database using dagtabse component. The whole history of requests will be rememebred so we can scroll through it and repeat any request.
- User will be able to add requests from history to favorites and to group them! There will be command / section so user can pick some and execute regularly! All this is going to be sotred in the database
- Export and import of helix.json and the database will be possible - in sql and json formats
- Theming - Default theme will be based on default ascii code generated LOGO presented at the top of Terminal UI. It will be in green-ish colors. However we will have the following color themes which user can change too: warm red, blue, green, yellow, gold, grey, white, darcula, dark blue, violet, warm orange, etc. User will be able to clone the theme and change the value, save the theme under the new name. All this will be stored in the database and will be loaded when user starts the application.

CLI (Headless) features

- All above but without UI - executed interactively or non-interactively (non-interactive mode is useful for automation and default one)

REST API features

- All above but without UI - executed interactively or non-interactively (non-interactive mode is useful for automation and default one) through REST API

Session work

- All started work will define a session
- Session can have a name
- All active session within a Helix agent will be stored in the database
- It will be possible to join to existing session using proipe "join" command and interacti with it from multiple places - for example user is connected via SSH or REST API from various computers. All this

must work.
- It will be possible to kill session which is stuck or we do not want it anymore with command "kill" passed to Helix CLI from the terminal
- It will be possible to list all active sessions with command "sessions"
- Filtering sessions by active, stuck, completed, etc.
- Session monitoring - show commands executed by the agent and their results, cpu and memory usage, etc. ("monitor" command)

Memory

All work done will be stored in database in several layers as working memory so LLMs will be able to use it and continue where they left of! The structire of remembering will be Project -> All done so far -> Current sessions -> Current work in progress. Once user re-executes the request, or types similar one and sends it to execution, system will recognize it as already in progress and will not execute it again! It will update on current status after check and next steps, then continue with work. If there is no such work done on project at the moment fresh new operation will be done to achieve golas from the request sent.

Testing

Tbd

Documentation

Tbd

Diagrams

Tbd