Sqirvy-cli (Go Implementation)

This directory contains the Go implementation of `sqirvy-cli`, a command-line tool for interacting with various Large Language Models (LLMs).

Overview

The Go version of `sqirvy-cli` provides a native executable for querying LLMs from the terminal. It leverages several popular Go libraries to offer a robust and efficient command-line experience.

Key Features

- **Native Executable**: Compiles to a single binary for easy distribution and execution.
- **Multi-Provider Support**: Interacts with Anthropic, Google Gemini, OpenAI, and Llama models via the `langchaingo` library.
- **Structured Commands**: Uses the `cobra` library for a clear command structure:
  - `query`: Sends arbitrary prompts (default command).
  - `plan`: Requests the LLM to generate a plan.
  - `code`: Asks the LLM to generate source code.
  - `review`: Instructs the LLM to review code or text.
  - `models`: Lists supported models and their providers.
- **Flexible Input**: Reads prompts from:
  - Standard Input (stdin) for easy piping.
  - File paths.
  - URLs (content is scraped using the `colly` library).
- **Configuration**:
  - Command-line flags (`-m` for model, `-t` for temperature) managed by `cobra`.
  - Environment variables for API keys (`ANTHROPIC_API_KEY`, `GEMINI_API_KEY`, `OPENAI_API_KEY`, `LLAMA_API_KEY`) and base URLs (`OPENAI_BASE_URL`, `LLAMA_BASE_URL`).
  - Optional configuration file support via `viper` (default: `$HOME/.config/sqirvy-cli/config.yaml`).
- **System Prompts**: Uses embedded `.md` files for command-specific system prompts (`query.md`, `plan.md`, `code.md`, `review.md`).
- **Modular Design**:
  - `cmd/sqirvy-cli`: Contains the main application logic, command definitions (`cobra`), and prompt reading/processing.
  - `pkg/sqirvy`: Implements the core LLM interaction logic, defining the `Client` interface and provider-specific implementations (Anthropic, Gemini, OpenAI, Llama) using `langchaingo`. Manages model-provider mapping and token limits.
  - `pkg/util`: Provides utility functions for file reading (`files.go`) and web scraping (`scraper.go`).

Building

You can build the executable using the standard Go toolchain:

```
cd go/cmd/sqirvy-cli
go build -o sqirvy-cli main.go
```

Alternatively, use the provided Makefiles:

```
make build # Builds the binary in the project root
```

or

```
cd go
make build # Builds the binary in go/bin/
```

Running

Once built, you can run the tool from your terminal:

```
# Basic query using default model and temperature
echo "What is the capital of France?" | ./sqirvy-cli

# Specify model and temperature, providing a file
./sqirvy-cli -m claude-3-5-sonnet-latest -t 0.7 query my_prompt.txt

# Generate a plan from stdin
cat requirements.txt | ./sqirvy-cli plan -m gpt-4o

# Generate code based on a plan file and a URL
./sqirvy-cli code -m gemini-1.5-pro plan.md https://example.com/api-spec

# List available models
./sqirvy-cli models
```

Remember to set the required API key environment variables for the models you intend to use.

Testing

Unit tests are included in the `pkg/sqirvy` and `pkg/util` directories. Run tests using:

```
cd go
make test
```

Or run tests for specific packages:

```
cd go/pkg/sqirvy
go test ./...

cd go/pkg/util
go test ./...
```

Note: Some tests in `pkg/sqirvy` require API keys to be set in the environment and will be skipped otherwise.