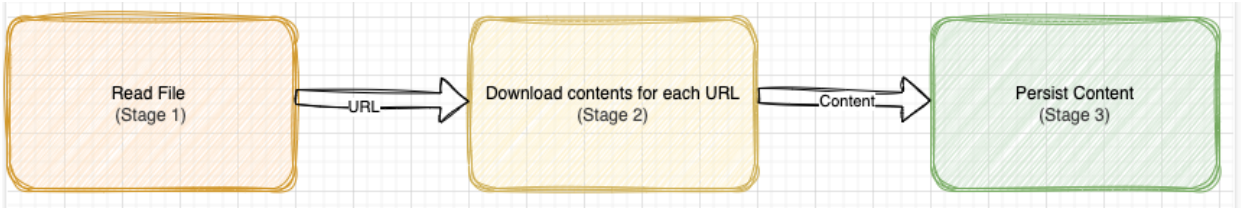
You need to create a command line application using Golang version 1.23 or newer. This application should accept a CSV file path as input. Your task is to access these URLs, download their content, and save it to the disk with files named randomly with the extension as ".txt."

The CSV file will contain a list of URLs, with one URL per line in a single column. The first line of the file is the header, and an example of the data is shown below.

Urls
www.example.com
www.someotherurl.com/api/v1
www.anotherone.com

Processing of the URLs should be managed via an in-process pipeline, which has three stages as follows.



Stage 1 - Read File.

This should be done from a goroutine. You should not read the entire file to memory to avoid excessive usage of resources.

Stage 2- Download contents

In this stage, you can have up to 50 go-routines, but not more than that. You should spin up a separate routine for each URL. The routines should be allocated only when URLs are available; if you have only 10 URLs to process, there should be only 10 goroutines' running; similarly, if you have 51 URLs available to process, there should be only 50 running. You should note that we are not using a pool of routines to batch-process the URL.

Stage 3 - Persist contents.

In this phase, there should be only 1 GoRoutine that persists in the file on disk.

Additional Requirements:

The pipeline must utilize channels for data transfer between stages. Ensure there are no deadlocks and enhance performance. The solution should scale effectively to handle files with hundreds of thousands of URLs.

Employ a logging library to track progress. Logs must document key milestones while omitting extraneous details. They can be sent to stdout. The logs should feature metrics such as the number of processed URLs, success and failure rates, and average download duration. Be cautious of excessive logging since the application is designed for large data sets.

Implement robust error handling. For example, if a URL cannot be downloaded, log the error and skip it. Ensure the application does not crash due to unexpected errors.

Ensure the application exits gracefully by allowing ongoing routines a deadline of 5 seconds to finish before quitting.

Include unit tests for key components, such as the CSV file reader, URL downloader, and file persister. Test edge cases, such as empty CSV files, invalid URLs, and network failures. Also, test the pipeline itself.

Include a README file that explains how to install and run the application, how to run unit tests, and any assumptions or design decisions you made. List any dependencies required to build and run the application.

Upload your code to a GitHub repository and share the link.

What We’re Looking For:

- Readability: Clean, well-documented code with meaningful variable and function names.
- Maintainability: Modular design, use of interfaces, and adherence to best practices.
- Extensibility: The solution should be easy to extend with additional features (e.g., adding new pipeline stages).

- Unit Tests: Comprehensive tests covering key functionality and edge cases.
- Production Quality Code