Local Network File Transfer Application (Go)

A peer-to-peer file transfer application that enables two devices on the same WiFi network to transfer files of any size.

Project Folder Structure

```
FileTransfer/
   cmd/
      sender/           # Entry point for sender mode
      receiver/         # Entry point for receiver mode
   internal/
      network/          # TCP/UDP connection handling
      transfer/         # File chunking, streaming, resume logic
      discovery/        # Device discovery on local network (mDNS/broadcast)
      protocol/         # Custom protocol definitions & message formats
      ui/               # Terminal UI or web UI handlers
   pkg/
      utils/            # Reusable utility functions (hashing, progress, etc.)
   configs/              # Configuration files (YAML/JSON)
   web/
      static/
          css/          # Stylesheets
          js/           # JavaScript files
          assets/       # Images, icons
      templates/        # HTML templates (if web UI)
   scripts/              # Build, deployment, helper scripts
   docs/                 # Documentation
   test/                 # Integration & end-to-end tests
   go.mod                # Go module definition (to be created)
   go.sum                # Dependency checksums (to be created)
   README.md             # Project readme (to be created)
```

Folder Descriptions

Folder	Purpose
`cmd/sender/`	Contains `main.go` for the sending device. This is the entry point when a user wants to send files.
`cmd/receiver/`	Contains `main.go` for the receiving device. This is the entry point when a user wants to receive files.
`internal/network/`	Handles low-level TCP/UDP socket connections, connection pooling, and keep-alive mechanisms.
`internal/transfer/`	Core file transfer logic - chunking large files, streaming data, checksum verification, and resume support for interrupted transfers.
`internal/discovery/`	Device discovery using mDNS (Bonjour/Avahi) or UDP broadcast to find other devices on the same network.
`internal/protocol/`	Defines the communication protocol - message types, headers, handshake procedures, and serialization.

Folder	Purpose
internal/ui/	User interface handlers - either terminal-based (TUI) or web-based UI logic.
pkg/utils/	Shared utility functions like file hashing (MD5/SHA256), progress bar helpers, file size formatting, etc.
configs/	Configuration files for ports, buffer sizes, default download paths, and other settings.
web/static/	Static web assets (CSS, JS, images) if building a web interface.
web/templates/	HTML templates for web UI rendering.
scripts/	Helper scripts for building, cross-compiling, or deployment automation.
docs/	Project documentation, API docs, architecture diagrams.
test/	Integration tests and end-to-end test scenarios.

Requirement Analysis

1. Functional Requirements

ID	Requirement	Priority
FR-01	Devices on the same WiFi network must discover each other automatically	High
FR-02	Users must be able to select single or multiple files for transfer	High
FR-03	Application must support files of any size (small KB to large GB files)	High
FR-04	Transfer progress must be displayed in real-time	High
FR-05	Transfers must be resumable if interrupted	Medium
FR-06	File integrity must be verified after transfer (checksum)	High
FR-07	Users should be able to cancel ongoing transfers	Medium
FR-08	Support folder/directory transfers	Low
FR-09	Show transfer speed and estimated time remaining	Medium
FR-10	Allow custom download directory selection	Medium

2. Non-Functional Requirements

ID	Requirement	Priority
NFR-01	Transfer speed should be optimized (utilize available bandwidth)	High
NFR-02	Application should work without internet (local network only)	High
NFR-03	Low memory footprint for large file transfers (streaming)	High
NFR-04	Cross-platform support (Linux, Windows, macOS)	Medium

ID	Requirement	Priority
NFR-05	Secure transfer option (encryption)	Low
NFR-06	Simple and intuitive user interface	High

3. Technical Requirements

Component	Technology/Approach
Language	Go (Golang)
Discovery	mDNS (using `github.com/hashicorp/mdns`) or UDP Broadcast
Transfer Protocol	TCP for reliable file transfer
File Handling	Streaming with configurable chunk sizes (e.g., 64KB-1MB)
Integrity	SHA-256 checksum verification
UI Options	Terminal UI (TUI) using `tview/bubbletea` OR Web UI
Configuration	YAML/JSON config files

4. System Architecture Overview

```
                        LOCAL NETWORK (WiFi)



       SENDER DEVICE               RECEIVER DEVICE


      1. Start App                     1. Start App
      2. Discover                 2. Broadcast
         Receivers                     Presence
      3. Select File
      4. Initiate                 3. Accept
         Transfer                      Request
      5. Stream                   4. Receive &
         Chunks                        Verify
      6. Confirm                  5. Send ACK
```

5. Data Flow

1. **Discovery Phase**: Receiver broadcasts presence via mDNS/UDP
2. **Connection Phase**: Sender discovers receiver and establishes TCP connection
3. **Handshake Phase**: Exchange metadata (filename, size, checksum)
4. **Transfer Phase**: Stream file in chunks with acknowledgments
5. **Verification Phase**: Receiver verifies checksum and sends final ACK

6. Key Go Packages to Use

Purpose	Package
mDNS Discovery	`github.com/hashicorp/mdns`
Terminal UI	`github.com/charmbracelet/bubbletea` or `github.com/rivo/tview`
Progress Bar	`github.com/schollz/progressbar`
Config	`github.com/spf13/viper`
CLI Flags	`github.com/spf13/cobra`
Logging	`github.com/rs/zerolog`

User Review Required

> [!IMPORTANT] **UI Choice Required**: Should the application have: - **Option A**: Terminal UI (TUI) - simpler, runs in terminal - **Option B**: Web UI - accessible via browser at `localhost:PORT` - **Option C**: Both options available

> [!IMPORTANT] **Transfer Mode**: Should both devices be able to send AND receive, or should the modes be separate (dedicated sender/receiver)?

Next Steps (After Approval)

1. Initialize Go module (`go mod init`)
2. Create base configuration structure
3. Implement device discovery
4. Implement transfer protocol
5. Build UI layer
6. Add tests
7. Cross-compile for different platforms

Verification Plan

Manual Testing

1. Run sender on Device A and receiver on Device B (same WiFi)
2. Transfer a small file (< 1MB) and verify content
3. Transfer a large file (> 1GB) and verify content/checksum
4. Interrupt a transfer mid-way and test resume
5. Test with multiple file types (text, binary, video)