PC 간 자동 폴더 동기화 어플리케이션

Progress report – 2

김정호

목차

PC 간 자동 폴더 동기화 어플리케이션 1

Progress report – 2 1

개발 내용 3

메인 페이지의 UI 개발 - 퍼블리싱 작업 3

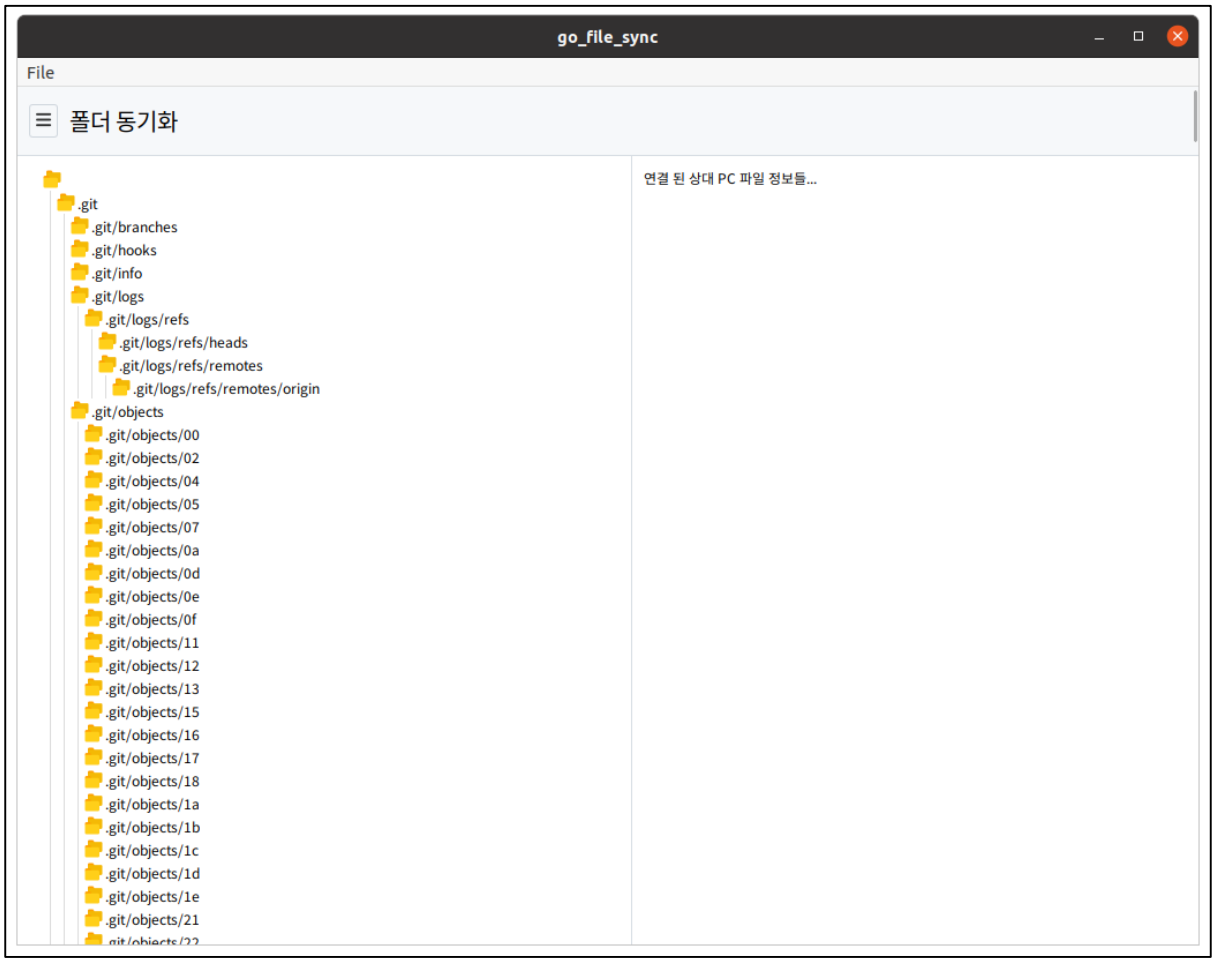
테스트 코드 작성 5

TCP/IP 통신 설정 6

개발 내용

메인 페이지의 UI 개발 - 퍼블리싱 작업

메인 페이지의 사용자 인터페이스(UI)를 최신화 하였습니다. 이 UI 업데이트는 동기화 기능을 향상시키기 위해 이루어졌으며, 특히 선택한 폴더 내부의 파일들과 하위 폴더에 포함된 모든 파일 목록을 개발하였습니다.



이 업데이트를 통해 사용자는 특정 폴더를 선택하면 해당 폴더 내부에 있는 모든 파일과 하위 폴더의 모든 파일 목록을 확인할 수 있습니다.

A. 사용자가 파일 및 폴더 간의 연결과 관련된 작업을 보다 쉽게 수행할 수 있게 해줍니다.

또한, 상위 폴더와 하위 폴더 간의 연관관계를 시각적으로 나타내기 위해 구분 선을 추가하였습니다.

B. 이 구분 선은 UI에서 사용자에게 각 폴더의 계층 구조를 명확하게 보여주어 파일 및 폴더 간의 관계를 시각적으로 이해하기 쉽게 합니다. 이로써 사용자는 어떤 파일이나

폴더가 어떤 상위 폴더에 속해 있는지 더욱 명확하게 파악할 수 있으며, 작업 중에 현재 위치를 빠르게 확인할 수 있게 됩니다.

테스트 코드 작성

서버와 클라이언트 부분에 대한 테스트 코드 작성을 시작하였습니다.
현재 기능이 예상대로 작동하는지 확인하기 위한 단계에 진입했습니다.

```go
type File struct {
    DirectoryPath string    `json:"directorypath"`
    FileName      string    `json:"filename"`
    FileSize      int64     `json:"filesize"`
    FileModTime   time.Time `json:"filemodtime"`
    Depth         int       `json:"depth"`
}

func NewFiles(directoryPath string, rootDepth int) ([]File, error) {
    var files []File
    err := filepath.Walk(directoryPath, func(path string, info os.FileInfo, err error) error {
        if err != nil {
            return err
        }
        if !info.IsDir() {
            dirPath := filepath.Dir(path)
            depth := strings.Count(dirPath, string(filepath.Separator)) - strings.Count(directoryPath, string(filepath.Separator))
            files = append(files, File{
                DirectoryPath: dirPath,
                FileName:      info.Name(),
                FileSize:      info.Size(),
                FileModTime:   info.ModTime(),
                Depth:         depth + rootDepth,
            })
        } else {
            dirPath := path
            depth := strings.Count(dirPath, string(filepath.Separator)) - strings.Count(directoryPath, string(filepath.Separator))
            files = append(files, File{
                DirectoryPath: dirPath,
                FileName:      "",
                FileSize:      0,
                FileModTime:   info.ModTime(),
                Depth:         depth + rootDepth,
            })
        }
        return nil
    })
    if err != nil {
        return nil, err
    }
    return files, nil
}
```

```go
func ParseDirectoryFiles(filesInfo []File) map[string][]File {
    var fileDir = map[string][]File{}

    for _, v := range filesInfo {
        fileDir[v.DirectoryPath] = append(fileDir[v.DirectoryPath], v)
    }

    return fileDir
}
```

사용자가 특정 폴더를 선택하면 해당 폴더 내의 모든 파일과 하위 폴더의 파일 목록을 확인할 수 있는 함수와 파일 및 디렉터리 정보를 디렉터리별로 그룹화하는 함수를 작성 했습니다.

서버는 위의 작업을 거친 이후에 클라이언트로 데이터를 보내주는데 위의 함수에 대한 테스트 코드를 작성했습니다.

TCP/IP 통신 설정

두 개의 PC를 TCP 연결하는 것이 중요한 과제입니다. 작업을 수행하기 위해 중요한 요소를 정리했습니다.

1. TCP/IP 설정: 두 개의 PC는 서로 서버인 상태로 설정되며, 어느 한 쪽이든 다른 PC로 연결하면 두 개의 PC가 맞물리는 형태여야 합니다.
2. 포트 설정: 두 PC는 하나의 랜선으로 연결되어 있으므로 포트 번호를 다르게 설정해야 합니다.
3. 에러 처리 및 재시도: 네트워크 환경에서는 항상 에러가 발생할 수 있으므로 에러 처리와 재시도 메커니즘을 구현해야 합니다.
4. 데이터 전송 및 동기화 프로토콜: 데이터 전송 및 동기화를 위한 프로토콜을 설계하고, 변경된 파일을 식별하고 전송하는 방법을 고려해야 합니다.
5. 사용자 편의성: 사용자가 연결 설정 및 동기화를 관리할 수 있도록 사용자 인터페이스를 제공하는 것이 중요합니다.

동기화 방법을 결정해야 합니다. 변경된 파일을 실시간으로 동기화할 것인지 아니면 주기적으로 동기화할 것인지 고려해야 합니다.

변경된 파일을 감지하는 방법과 충돌을 관리하는 방법도 고려해야 합니다.