endurox-connect - Feature #606

Multi-part file download

11/10/2020 10:45 PM - Madars

Status:	Closed	**Start date:**	11/10/2020
Priority:	Normal (Code 4)	**Due date:**	
Assignee:		**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Streaming interface shall be created for file download

History

#1 - 11/10/2020 10:45 PM - Madars

```
################################################################################
# Fields:
################################################################################
EX_STREAM_FILENAME    string # file name
EX_STREAM_FILENO    long # file number in download
EX_STREAM_HEADER    string # file heaer
EX_STREAM_PARNO    long # part number in fileno
EX_STREAM_COMMAND        # stream command number
EX_STREAM_PARAM1        # streaming parameter
EX_STREAM_DATA    carray        # data block from file
################################################################################

################################################################################
# commands:
################################################################################
0 - normal request, control to destination
1 - receive form files, argument found in "EX_STREAM_PARAM1", gives control to other party,
2 - sending data, control not switched
3 - acq, send next, control to other party
4 - acqrsp, control to other party
5 - all files downloaded, control to other party
99 - error occurred, control given
################################################################################

/upload_files {stream:"y", ack:"100"}
```

Sample source:

```
package main

import (
    "crypto/sha256"
    "fmt"
    "io"
    "io/ioutil"
    "log"
    "mime/multipart"
    "net/http"
    "os"
)

var (
    indexPage = `<html>
    <body>
        <form enctype="multipart/form-data" action="http://localhost:8080/upload" method="post">
            <input type="file" name="files" multiple />
            <input type="submit" value="upload" />
```

```
        </form>
    </body>
</html>`
)

func doSomethingWithFile(f *os.File) {
    if n, err := f.Seek(0, 0); err != nil || n != 0 {
        log.Printf("unable to seek to beginning of file '%s'", f.Name())
    }
    h := sha256.New()
    if _, err := io.Copy(h, f); err != nil {
        log.Printf("unable to hash '%s': %s", f.Name(), err.Error())
    }
    log.Printf("SHA256 sum of '%s': %x", f.Name(), h.Sum(nil))
}

func serveIndex(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(200)
    w.Write([]byte(indexPage))
}

func readParts(w http.ResponseWriter, r *http.Request) {
    // define some variables used throughout the function
    // n: for keeping track of bytes read and written
    // err: for storing errors that need checking
    var n int
    var err error

    // define pointers for the multipart reader and its parts
    var mr *multipart.Reader
    var part *multipart.Part

    log.Println("File Upload Endpoint Hit")

    if mr, err = r.MultipartReader(); err != nil {
        log.Printf("Hit error while opening multipart reader: %s", err.Error())
        w.WriteHeader(500)
        fmt.Fprintf(w, "Error occured during upload")
        return
    }

    // buffer to be used for reading bytes from files
    chunk := make([]byte, 4096)

    // continue looping through all parts, *multipart.Reader.NextPart() will
    // return an End of File when all parts have been read.
    for {
        // variables used in this loop only
        // tempfile: filehandler for the temporary file
        // filesize: how many bytes where written to the tempfile
        // uploaded: boolean to flip when the end of a part is reached
        var tempfile *os.File
        var filesize int
        var uploaded bool

        if part, err = mr.NextPart(); err != nil {
            if err != io.EOF {
                log.Printf("Hit error while fetching next part: %s", err.Error())
                w.WriteHeader(500)
                fmt.Fprintf(w, "Error occured during upload")
            } else {
                log.Printf("Hit last part of multipart upload")
                w.WriteHeader(200)
                fmt.Fprintf(w, "Upload completed")
            }
            return
        }
        // at this point the filename and the mimetype is known
        log.Printf("Uploaded filename: %s", part.FileName())
        log.Printf("Uploaded mimetype: %s", part.Header)

        tempfile, err = ioutil.TempFile(os.TempDir(), "example-upload-*.tmp")
        if err != nil {
            log.Printf("Hit error while creating temp file: %s", err.Error())
            w.WriteHeader(500)
```

```
        fmt.Fprintf(w, "Error occured during upload")
        return
    }
    defer tempfile.Close()

    // defer the removal of the tempfile as well, something can be done
    // with it before the function is over (as long as you have the filehandle)
    defer os.Remove(tempfile.Name())

    // here the temporary filename is known
    log.Printf("Temporary filename: %s\n", tempfile.Name())

    // continue reading until the whole file is upload or an error is reached
    for !uploaded {
        if n, err = part.Read(chunk); err != nil {
            if err != io.EOF {
                log.Printf("Hit error while reading chunk: %s", err.Error())
                w.WriteHeader(500)
                fmt.Fprintf(w, "Error occured during upload")
                return
            }
            uploaded = true
        }

        if n, err = tempfile.Write(chunk[:n]); err != nil {
            log.Printf("Hit error while writing chunk: %s", err.Error())
            w.WriteHeader(500)
            fmt.Fprintf(w, "Error occured during upload")
            return
        }
        filesize += n
    }
    log.Printf("Uploaded filesize: %d bytes", filesize)

    // once uploaded something can be done with the file, the last defer
    // statement will remove the file after the function returns so any
    // errors during upload won't hit this, but at least the tempfile is
    // cleaned up
    doSomethingWithFile(tempfile)
  }
}

func main() {
  log.Println("Gopher upload service started")
  http.HandleFunc("/upload", readParts)
  http.HandleFunc("/", serveIndex)
  log.Fatalf("Exited: %s", http.ListenAndServe(":8080", nil))
}
```

#2 - 03/06/2021 09:38 PM - Madars

- Status changed from New to Resolved

- % Done changed from 0 to 100

#3 - 03/06/2021 09:38 PM - Madars

- Status changed from Resolved to Closed