**Topics in Computer Security project: code documentation**

Submitted by: Omri Basch: 203536917. Email: [omrib15@gmail.com](mailto:omrib15@gmail.com)

This document provides an overview of the project’s source code, which resides in the directory **FileManager/src**.

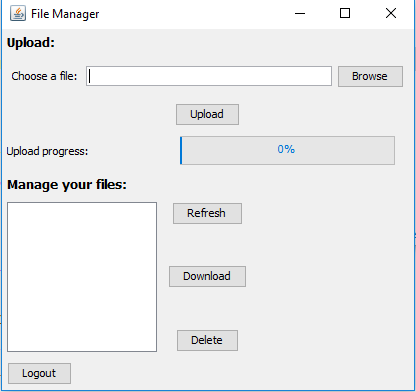
**Table of contents**

**Client side………………………………………………………………………………………..2  
 package: client…………………………………………………………………….2  
 package: client.encryption……………………………………………………4  
 package: client.mac……………………………………………………………..4  
 package: client.login…………………………………………………….………4**

**Server side……………………………………………………………………………………….6**

**Client side**

Under the client package at **FileManager/src/client** you will find the client code. We will first go over the important classes within the client package, and then the packages nested within the client package.



**Package: client**

UserFrame.java

This class extends swings’ JFrame class.  
It is the frame from which the users can  
interact with their files. Let’s go over   
some of the **UserFrame** class important   
features:

**Note:** each http request made the client to any of the servers’ resources is attached a “Authorization” header for Basic Auth with the server (more information about that in the server part of the document).

* **Uploading files:** to upload a file to the server, the client does the following:  
  1. Encryptes the selected file name and the file itself via the **CryptoUtils** class static encryption methods which use the AES cipher and the users’ specific encryption key (further explained later).   
  2. Creates a new UploadTask object (which abstracts file upload), providing it with the URL for the uploads resource on the server and the **encrypted** file.  
  3. Executes the UploadTask.
* **Downloading files:** to download a file from the server, the client does the following:  
  1. Gets the user selected file name from the list of files on server.  
  2. Downloads the encrypted file from the server via the **HttpDownloadUtility** class static method appropriately called **downloadFile.**3. Decrypts the file via **CryptoUtils** class.  
  4. Authenticates the file by computing its tag (using the **authUtil** class and the users’ authentication key). and comparing it to the tag received along with the downloaded file.
* **Deleting files:** to delete a file from the server, the client does the following:  
  1. Gets the user selected file name from the list of files on server.  
  2. Encrypts the files name via **CryptoUtils** class and the users’ encryption key.  
  3. Sends an http DELETE request to the FilesResource, with the encrypted file name.
* **Refreshing file list:** this is accomplished by sending an http GET request to the FilesResource. To which the server responds with a list of the current users’ file names on server. This action might cause a warning to popup in case the server detected unauthorized changes to the users’ files.

UploadTask.java

This class is used to execute the file upload in a background thread and updates progress to the progress bar listener (a nice little GUI feature).  
It uses an instance of the **MultipartUploadUtility**.

MultipartUploadUtility.java

This utility class provides an abstraction layer for sending multipart (files, in our case) POST requests to a web server.

HttpDownloadUtility.java

A utility that downloads a file from a given URL (via http GET request), and saves it to specified location locally.

UserInfo.java

An instance of this class Is created after successful user login. Its constructor receives the user name the master secret that the user typed, and derives the users’ password, encryption key and authentication key. The Created instance is then used by the UserFrame object in order to perform user specific encryption/decryption, file authentication and more.

**Package: client.encryption**

This package contains cryptography utilities for the client to use.

CryptoUtils.java

This class provides the client with static methods that use the AES cipher to encrypt/decrypt files and Strings (for the encryption of file names).

**Package: client.mac**

This package contains only the following file:

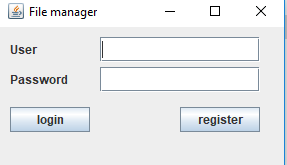
AuthUtil.java

This class provides the client with the a static method (getAuthTag) which given a file and an authentication key, computes the file tag according to the HMACSHA256 algorithm.

**Package: client.login**

This package contains the LoginFrame class, which is the entry point of the client.

LoginFrame.java



This class extends Swings’ JFrame class.   
It contains a **main** method which is the   
entry point of the client.   
Its’ main features are:

* **User registration:** to register a new user, the client does the following:  
  1. Makes sure that the user typed legitimate name and password (longer than 6 characters).  
  2. Uses SHA256 to hash the master secret typed by the user.  
  3. Sends an http POST request to the registrationResource with the hashed user name and password, in order to persist them on the server for future logins.
* **User login:** to login, the client:  
  1. Makes sure that the user typed legitimate name and password (longer than 6 characters).  
  2. Uses SHA256 to hash the master secret typed by the user.  
  3. Sends an http GET request to the LoginResource. (this is frther explained in the server part of this document).  
  4. If the server responds with login confirmation, a new UserFrame object is created (with the users’ specific UserInfo object) and displayed. The LoginFrame is disposed.

**Server side**

Under the server package at **FileManager /src/server** there are 5 classes:

SecurityFilter.java

This class implements ContainerRequestFilter interface, which basically means that whenever a request is made to any of the resources (FilesResources, LoginResource etc.), the request is first processed by the **filter** method. My implementation of the **filter** method in SecurityFilter is such that it performs http Basic Auth of the request (meaning it confirms the request was made by a registered user). Note that my implementation **does not** perform Basic Auth on requests made to the registration resource, since users need to able to register before their credentials can be authenticated.   
If the authentication of the requesting user fails, the filter aborts the request and sends back a response with UNAUTHORIZED status code, thus not allowing unauthorized access to any of the project’s resources.

UploadServlet.java

This class is an extension of the basic HttpServlet. It is in charge of handling the file uploads to server due to clients http POST requests, and file downloads from server due to clients http GET requests.  
When an http POST request is made to the UploadServlet URL, the **doPost** method is invoked to process the request. Its main objectives are:

1. Persist the file in the request’s body to the user directory on the server
2. Persist the file’s tag and length in the special file within the user directory (user\_dir/auth/auth.txt. these names are not encrypted in order to make testing the project easy for both of us).
3. Return a proper http response to the client, which informs whether the upload was successful or not.

Similarly, when an http Get request is made to UploadServlet URL, the **doGet** method is invoked to process the request. Its main objectives are:

1. Get the user name and requested file name (encrypted) via the “fileName” and “username” request parameters.
2. Use the user and file names to locate the file the client wants to download, and add the file to the http response body.
3. Add the file tag to the response, for the client to authenticate the file.
4. Send back the response to the client.

RegistrationResource.java

This class uses the [Jersey](https://jersey.github.io/) library to provide a REST api for the client to interact with the server via http requests. specifically for user registration.

**Note:** You might ask yourself why is UploadServlet implemented as a basic HttpServlet, and the registrationResource (and other resources as you will see) implemented with the **Jersey** library. The reason is that prior to writing this project I did not know a lot about client-server communication, so I did some research and decided it will be nice to gain hands-on practice in both older ways – the basic HttpServlet, and the newer ways- Jersey, which is an implementation of the JAX-RS (Java API for RESTful Web Services) specification.

Whenever an http request is made to the “%projectURL%/registration” URL by the client, the registrationResource class handles it.

When the client wishes to register a new user, It sends an http **POST** request to “%projectURL%/webapi/registration”, which invokes the registrationResource method **registerUser** (this is accomplished via Jerseys @POST annotation above the method).

The **registerUser** methods main objectives are:

1. Get the user name and password from the request.
2. Generate a new random salt for the user.
3. Use sha256 to hash the (password,salt), get h(pass,salt).
4. Persist the triplet: <username, salt, h(pass,salt)> to the special passwords file.

FilesResource.java

This class also uses the [Jersey](https://jersey.github.io/) library to provide a REST api for the client to interact with the server via http requests. specifically, to get a list of the user files names on server or delete some of them.

When the client wishes to get the list of its files names on server, it sends an http **GET** request to “%projectURL%/webapi/Files/%userName%”, which invokes the FileResource method **getFileNames** (this is accomplished via Jerseys @GET and @Path annotation above the method).

The **getFileName** methods main objectives are:

1. Get list of file names from the user directory on server (specified by the requests “userName” parameter).
2. Make sure no files names or size were changed on the server, by comparing the file name and size to the ones persisted in the special file when the files were uploaded (see UploadServlet).
3. If no file name or size were changed on the server, send back to the client a response with the list of file names the user has on the server. Otherwise, send a response which notifies the user that its files on server were changed.

When the client wishes to delete a file it has on the server, it sends an http **DELETE** request to “%projectURL%/webapi/Files/%userName%/%fileName%”, which invokes the FileResource method **deleteFile** (this is accomplished via Jerseys @DELETE and @Path annotation above the method).

The **deleteFile** methods main objectives are:

1. Get the user name and file name from the requests “userName” and “fileName” parameters.
2. Delete the specified users file from server
3. Delete the specified users file information from the special file that has file names, tags and sizes.

LoginResource.java

This class also uses [Jersey](https://jersey.github.io/). When the client wishes to login to its encrypted file system, it send an http GET to “%projectURL%/webapi/login”, which invokes the LoginRource method **login**. Notice that this method body has no complex logic in its body, since as seen before, the **SecurityFilter** makes sure every request made to server (other than registration of course) is by an authorized user. Therefore, if the client sends a GET request to this LoginResource in order to login, but with unauthorized (not registered) credentials, the **SecurityFilter** will catch that request and the user would not be able to login.