Birzeit University
Software Engineering Master Program
Software Construction SWEN6301
2016-2017

Course Project Report - D1

Prepared By:
Citreen Shihadeh
Hanan Namrouti
Saad Hazem
Instructor:
Dr. Ahmad Afaneh

Revision Control

Date	Version	Description	Author
18/Dec/2016	0.1	Initial draft- testing Download manager ver 1.1	Citreen & Hanan & Saad
24/Dec/2016	1.0	Deliverable 1- design document	Citreen & Hanan & Saad

1. Table of Contents

2. Table of Figures

3. Basic Components

This application will be implemented using Java, the basic components to deliver the requested functionality are the following:

1. Multipart player	The module that provides an API for the download manager client apps. It is responsible to parse, validate, stream and assemble the requests of downloading files.
2. Manifest file	The object the holds the reference for the needed segments to be downloaded to create the final file. It also may contain a reference to other Manifest files, and to create a nested Manifest file.
3. Segment	A small part of the whole file to be downloaded, each segment will contain at least one downloading mirror
4. Parser	The module responsible of tokenizing the Manifest file to get the downloading links for each requested file; and then convert it to a downloading object.
5. Downloader	The client app, basically provides a GUI for the user to select the options to start downloading the needed file, it can be a desktop, web or mobile downloader.

3.1. Functionalities provided by the multipart player:

1. Stream	The functionality of executing the download, performed by connecting to the server that hosts the downloading link
2. Assemble	The functionality provided by the multipart player, in which all the downloaded segments are being put in order to form the final file
3. Validate	The functionality of checking the health of the Manifest file, with valid segment separators, as well as validating the Manifest file type, and each downloading mirror formatting. This is important to decrease the load on the downloading machines from excessive or malicious downloading links.

4. Introduction

The purpose of this software design document is to provide a low-level description of the Multipart downloader , providing insight into the structure and design of each component. topics covered include its architecture and design explanation, in terms of UML diagrams, machine state diagram, sequence diagram and test cases and expected results

4.1. Project overview and scope

Multipart downloader in general is going to download file parts from different machines and assemble them in one huge file incrementally according to predefined addresses of the multipart files in a manifest file. This document will elaborate in describing the high level architecture, processes, activities, and other detailed design constraints that all will comprise a driver for implementation and an insight for later readers.

4.2. Goals and objectives

The objectives of this application is solving the problem of downloading files with large sizes, where they are being broken into smaller downloading parts to make the process faster and more reliable. Our application is able to handle a file with multiple segments; each is hosted on a different server, and assemble them as one file representable to the user.

5. Team Members and Roles

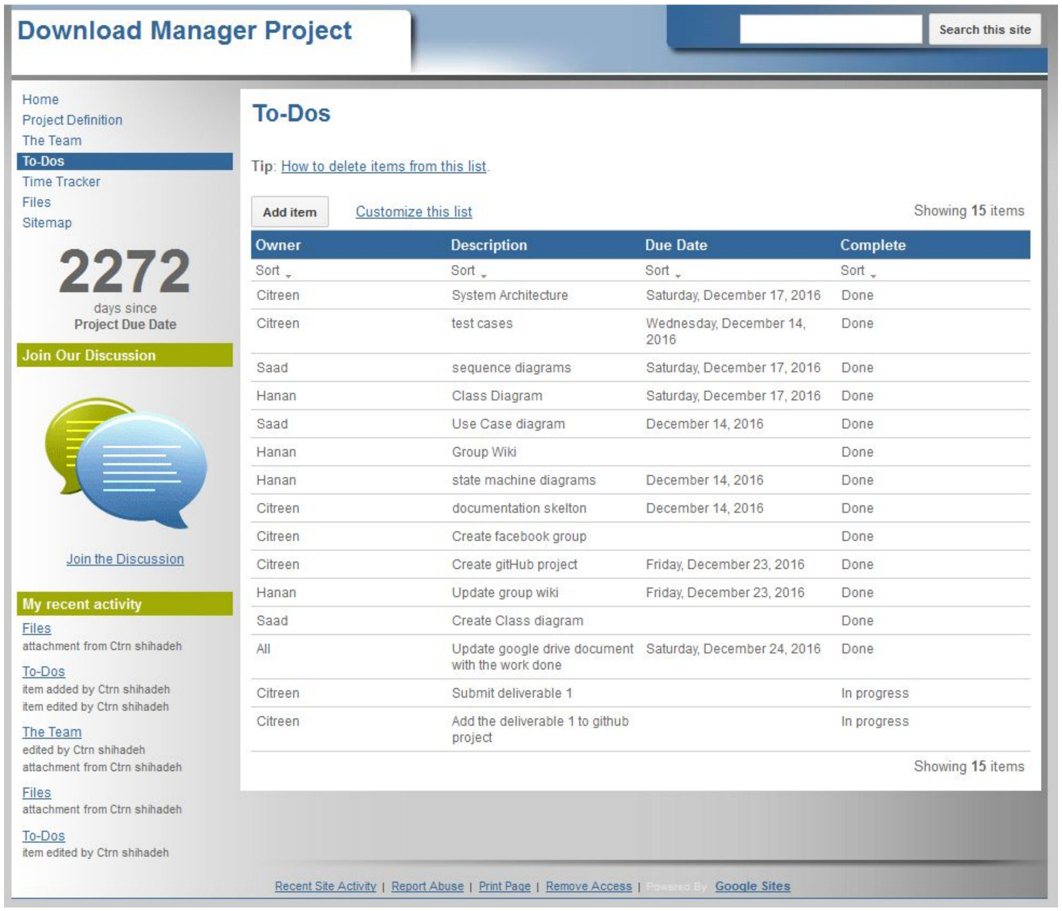
We would like to state that all team members were present in daily meetings and have participated in the tasks effectively.

Each member was assigned to deliver a specific task, and the other 2 members did review for the work and posted comments.
Each member task was added to the group wiki; please refer to this detailed list for our tasks (https://sites.google.com/site/downloadmanagerproject/to-dos) and we updated it continuously.

During this deliverable the roles were as the following:

1. Citreen: Testing and Team Leader
2. Hanan: Design and Website Administrator
3. Saad: System Architect.

Figure 1 shows the task assignment during deliverable 1:



My recent activity

Files
attachment from Ctrn shihadeh

To-Dos
item added by Ctrn shihadeh
item edited by Ctrn shihadeh

The Team
edited by Ctrn shihadeh
attachment from Ctrn shihadeh

Files
attachment from Ctrn shihadeh

To-Dos
item edited by Ctrn shihadeh

To-Dos

Tip: How to delete items from this list.

Add item Customize this list Showing 15 items

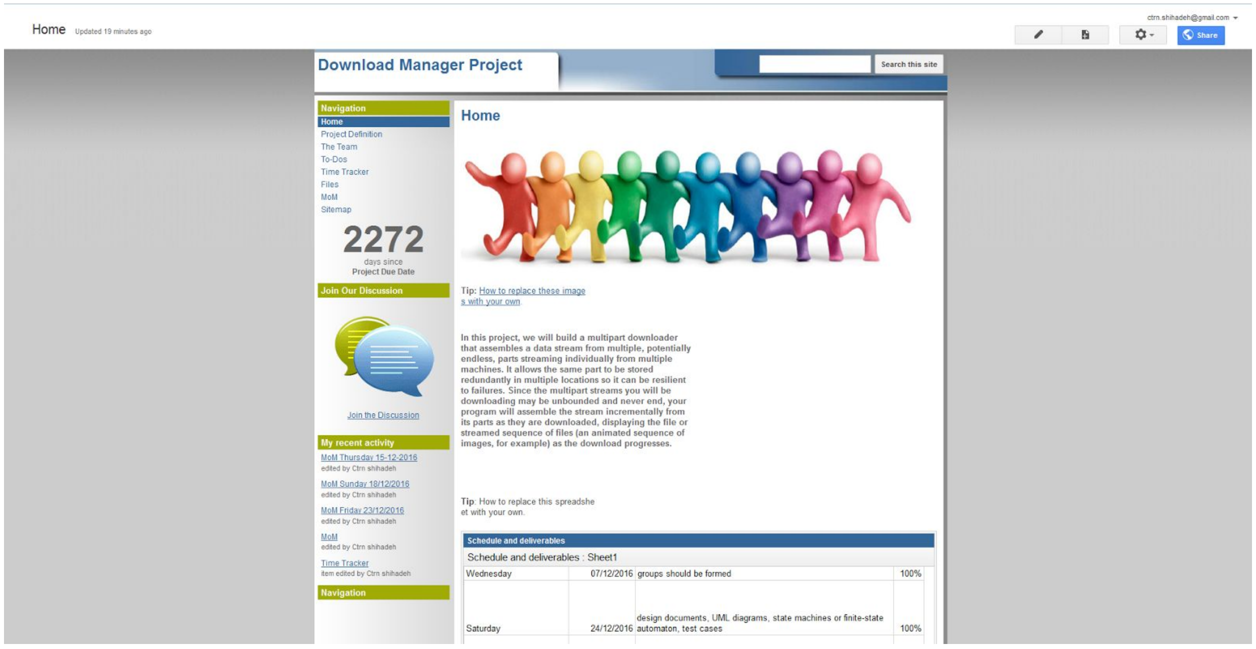
Owner	Description	Due Date	Complete
Sort	Sort	Sort	Sort
Citreen	System Architecture	Saturday, December 17, 2016	Done
Citreen	test cases	Wednesday, December 14, 2016	Done
Saad	sequence diagrams	Saturday, December 17, 2016	Done
Hanan	Class Diagram	Saturday, December 17, 2016	Done
Saad	Use Case diagram	December 14, 2016	Done
Hanan	Group Wiki		Done
Hanan	state machine diagrams	December 14, 2016	Done
Citreen	documentation skelton	December 14, 2016	Done
Citreen	Create facebook group		Done
Citreen	Create gitHub project	Friday, December 23, 2016	Done
Hanan	Update group wiki	Friday, December 23, 2016	Done
Saad	Create Class diagram		Done
All	Update google drive document with the work done	Saturday, December 24, 2016	Done
Citreen	Submit deliverable 1		In progress
Citreen	Add the deliverable 1 to github project		In progress

Showing 15 items

Figure 1: To Do List for Deliverable 1

6. Collaboration Tools Used

1. **GitHub**: we created a repository that contains all the project resources.
    a. URL: https://github.com/Ctrn/ConstDownloadManager
2. **Google Drive**: the initial drafted document and all the diagrams used were created and shared using google drive. All history changes are viewable and we can post comments offline, and do collaborative editing in real time.
3. **Google Hangouts**: daily meetings and collaborative sharing.
4. **Facebook group**: for coordinating the meetings.
5. **Google Sites**: we created a group wiki that includes all the project resources, including MoM and files, tasks lists, and others. (https://sites.google.com/site/downloadmanagerproject/)



In this project, we will build a multipart downloader that assembles a data stream from multiple, potentially endless, parts streaming individually from multiple machines. It allows the same part to be stored redundantly in multiple locations so it can be resilient to failures. Since the multipart streams you will be downloading may be unbounded and never end, your program will assemble the stream incrementally from its parts as they are downloaded, displaying the file or streamed sequence of files (an animated sequence of images, for example) as the download progresses.

Tip: How to replace this spreadsheet with your own.

Schedule and deliverables

Schedule and deliverables : Sheet1

Wednesday	07/12/2016	groups should be formed	100%
Saturday	24/12/2016	design documents, UML diagrams, state machines or finite-state automaton, test cases	100%

Figure 2: Wiki Home page

7. System Architecture

In this section we will introduce a brief review for the system architecture.
The download manager is a client server application, where the client requests to download a specific file from a given URL, this request is represented in the form of the Manifest file, and will be manipulated to validate the given segments to start download and get the file download (response).

Each client will connect to the multipart download via the download manager. The multipart downloader will provide the needed API for the clients; either web, or desktop apps; to download files. The multipart downloader is responsible for parsing and validating the Manifest file, and to get the segments needed to stream the requested file.

Figure 2 represents the block diagram for the basic components listed in the previous section

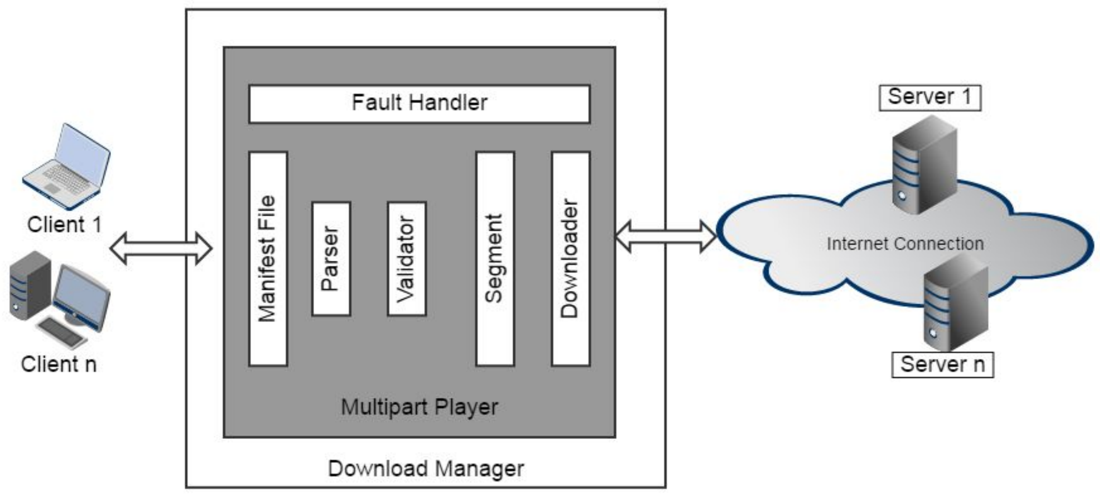


Figure 3: System Architecture

8. UML Diagram

1. 8.1. Use case Diagram

1. **8.2. UC-1 : Stream Segments**

   Actors:
   1. Primary Actor: User.
   2. Secondary Actor: N/A

   Description: The User Want to view a file or a set files from the servers.
2. **UC-2:** Break File to Segments

   Actors:
   1. Primary Actor: Machine "Servers".
   2. Secondary Actor: N/A

   Description: The Servers segments the files to a smaller segments and store them on multiple machine that can be accessed to download them.
3. **UC-3:** Download Segments.

   Actors:
   1. Primary Actor: User.
   2. Secondary Actor: "Multi-Part Downloader"

   Description: The user wants initiate the download of the needed file to be stream, while the multi-part downloader organizes the download of the segments and their orders.

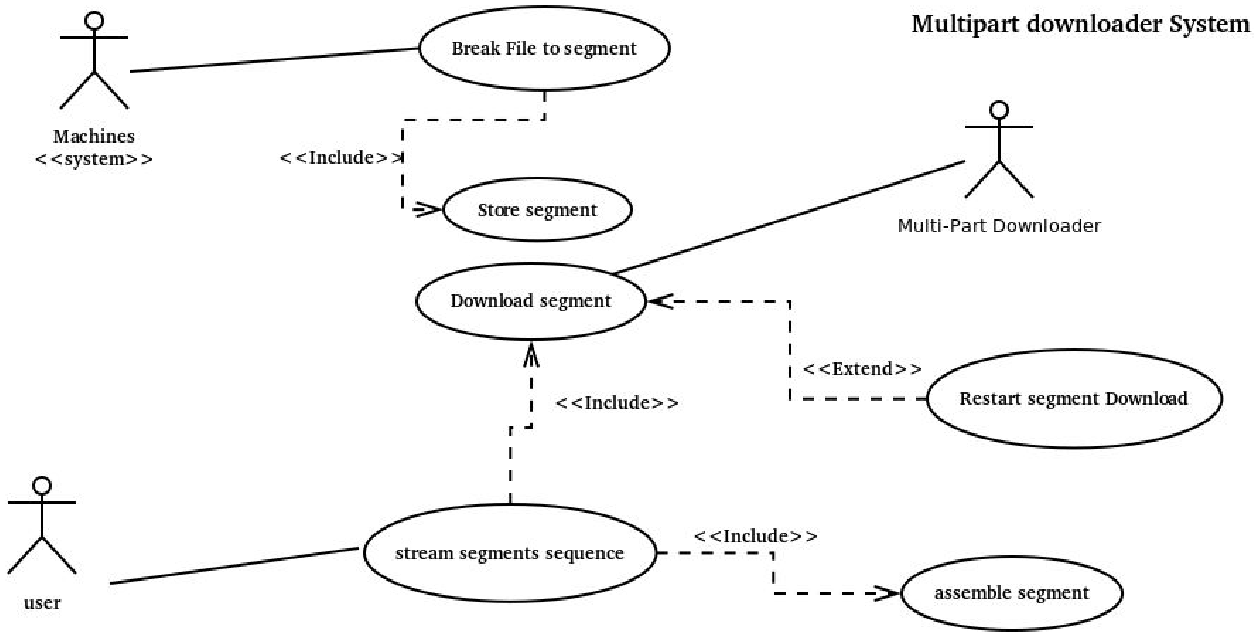


Figure 4: Use Case Diagram

2. 8.3. Sequence Diagram

We are show casing the interaction between the following objects in order to stream a segment on the end-user side:

- User.
- Multi-Part Player.
- File Parser.
- Tokens Validator.
- Segment Finder.
- Segment Downloader.

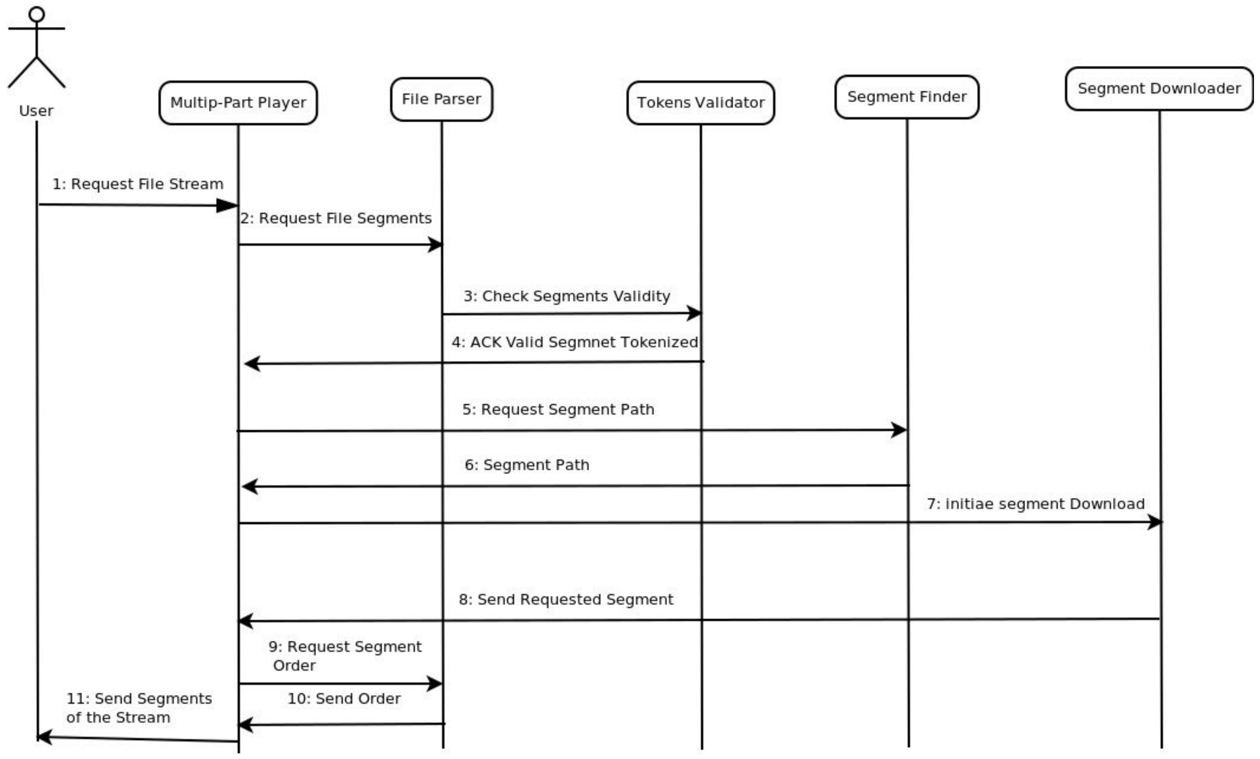


Figure 5: Sequence Diagram

3. 8.4. Class Diagram

In order for our structural representation to download and play a segment located on the servers "Machines" we have taken the following assumptions and preconditions:

1. The **Streamer** will check the Network **Connection.**
2. The **Streamer** will be able to **validate** the location, status and other attributes associated with the desired segment.
3. The **File** a set of of Segments located on **Multiple Server**.
4. After the Successful Download and the validation of the Segment the **Stream Player** will be able to show the **File** that was segmented in order.

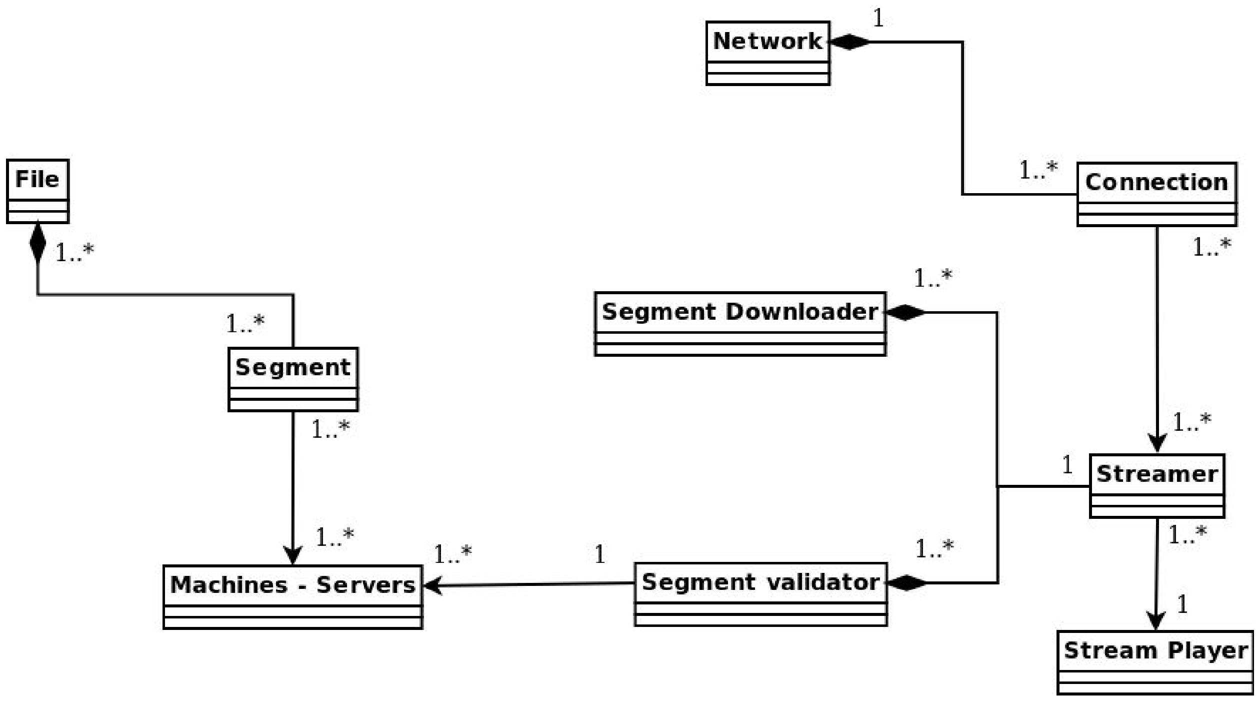


Figure 6: Class Diagram

4. 8.5. State machine diagram

The state machine diagram represent the main activities include:

- ➔ Reading the file
  - input: file
  - action read
  - next step parsing
- ➔ Parsing the file
  - input:url
  - action :parse url
  - next step: validate
- ➔ Validating
  - input: parsed url,
  - action: check validity
  - next step: if it's file parse again If it's segm downloading
- ➔ Downloading file
  - input: url of segment in specific machine
  - action downloading
  - next step: assembling
- ➔ Assembling
  - input: downloaded file
  - action assembling
  - next step :end

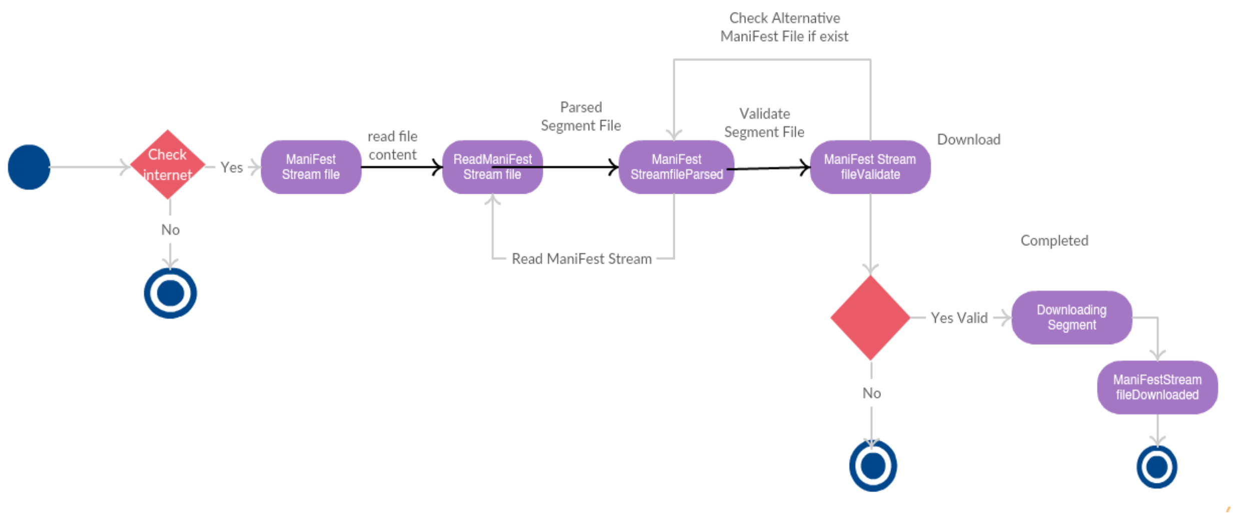


Figure 7: State machine diagram

9. Test Cases

In this section we will state the proposed test cases for the SW. We divided them into black and white testing, both good and bad flows.

Those test cases we will be executed later and will be used to catch issues in the SW.

Black Box Testing				
Good Flow test cases				
Case ID	Description	Steps	Expected result	Actual result
1.	Download URL is valid	1. Run manifest file 2. Download link is valid and so download starts	Success	
2.	Download correct file from clicked url	1. Run manifest file 2. Download starts 3. Download ends successfully 4. Check the downloaded file, you should get the one you requested	Success	
3.	Downloaded file is not corrupted	1. Run manifest file 2. Download starts 3. Download ends successfully 4. Check the file if it is in the right order, example:a music file should be sequentially arranged	Success	
4.	Download link is accessible	1. Run manifest file 2. Download starts successfully	Success	
5.	Progress bar is shown while download is in progress	1. Run manifest file 2. Download starts 3. While downloading you should a progress bar that notifies you on the progress of the download	Success	

6.	Informative error message received upon download failure	1. Run manifest file 2. In cases an error occurred while downloading (example: no internet connection) you should see a descriptive error message	Success	
7.	Informative message received upon download complete	1. Run manifest file 2. Download starts and ends successfully 3. Descriptive msg should be displayed on download complete	Success	
8.	Create valid manifest file to download a file	1. Create a new file with correct type 2. Add segments of download 3. Add at least one mirror for each segment 4. Start download 5. Successful Download message displayed. 6. File is downloaded correctly.	Success	
9.	Create manifest file with duplicated segments and download the file	1. Create a file with the correct extension 2. Add segments for download 3. Duplicate 1 or more segments in the file 4. Start download 5. Download should end successfully	Success	
10.	Create a nested manifest file and download the file	1. Create Manifest file with at least 2 segments 2. One of the segments should be pointing to another manifest file that has 3. Download should start and ends successfully 4. Informative error message displayed upon successful.	Success	

Bad flow test cases				
11.	Download file from inaccessible URL	1. Create Manifest file with inaccessible URL 2. Download should not start 3. Informative error message should be received.	Fail	
12.	Download from invalid URL (error in the format)	1. Create Manifest file with invalid URL 4. Download should not start 5. Informative error message should be received.	Fail	
13.	Download file while network disconnected	1. Disconnected internet connection 2. Start download 3. Download should not start and an 4. informative error message should be displayed	Fail	
14.	Create manifest file with 2 download mirrors, with 2 mirrors is invalid/not accessible	1. Create Manifest file with 2 download mirrors that are invalid 2. Download should start 3. Download should terminate 4. Error message is displayed	Fail	
15.	Create manifest file with invalid file extension (type)	1. Create Manifest file type not as expected 2. Download should not start 3. Error msg is displayed	Fail	
White-box testing				
Good Flow test cases				
16.	Valid URL formats for download mirrors	1. Get a Manifest file 2. Check URL validity: a. Format b. Language	Success	

		c. Special characters		
17.	Valid segments separator	1. Get a Manifest file 2. Check segment separator validity with ** between each segment and the other one	Success	
18.	Download from Manifest file and check performance time	1. Get Manifest file with segments with not more than 50 kbyte size 2. Downloading time should not exceed 5 minutes	Success	
19.	Disconnect internet while download is in progress	1. Get a Manifest file 2. Start Download 3. Progress bar is shown while downloading 4. Disconnect internet. 5. Download should stop with an informative error message	Success	
20.	Download from Manifest file with Inaccessible mirrors	1. Get a manifest file with more than 1 mirror for each segment; and 1 mirror is not valid 2. Start download 3. Download is started and progress bar appears 4. Download stops successfully and informative message is shown	Success	
21.	Check sequence of segments downloaded for different types of files	1. Get a valid Manifest file 2. Download starts and stops successfully with informative messages 3. Check the downloaded file if it is corrupted or not	Success	
Bad Flow test cases				
22.	Download from invalid manifest file	1. Get invalid manifest file 2. Check segments invalidity 3. Dwonload should not start	FAIL	

		4. Informative error message is displayed to the use it		