

Birzeit University

Software Engineering Master Program

Software Construction SWEN6301

2016-2017

“Course Project Final Report

Multipart downloader”

Team Members

|  |  |  |
| --- | --- | --- |
| Name | ID | E-Mail |
| Citreen Shihadeh | 1145342 | ctrn.shihadeh@gmail.com |
| Hanan Namrouti | 1165217 | hanan.namrouti@gmail.com |
| Saad Hazem | 1165293 | saadajaj35@gmail.com |

Instructor

Dr. Ahmad Afaneh

# **Revision Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 20/Jan/2017 | 0.1 | Initial draft- Final Report | Citreen |
| 25/Jan/2017 | 0.2 | Document Structure and updates | Citreen |
| 28/jan/2017 | 1.0 | Final version | Citreen, Hanan & Saad |

# 

# **Table of contents**

[**Revision Control**](#_4jmmkgnn228k) **1**

[**Table of contents**](#_hh129l2s5yi1) **2**

[**Table of Figures**](#_5r1uruk502db) **4**

[**Table of Tables**](#_srxcb6v0jbj6) **5**

[**1.Project Links**](#_55bqkwj1uyir) **6**

[Source Code on GitHub](#_dq3dc6p95s6u) 6

[Group Wiki](#_zi9d9spvu1xo) 6

[Deliverable 1: Design Document](#_40wptenjoqk7) 6

[Deliverable 2:](#_bvn9155z0fy) 6

[JavaDoc](#_stady6tmtshh) 6

[Bugzilla and BugList](#_2bizxdixezkw) 6

[**2. Scope**](#_ft9vnnixeyga) **7**

[**3. Introduction**](#_jltgl2ez39rf) **7**

[**4. Goals and objectives**](#_6459sf70t77n) **7**

[**6. JavaDoc**](#_rme816fqlnox) **8**

[Why JavaDoc:](#_8rvf2cvlehlx) 8

[**7. Team Members and Roles**](#_8rftf8o1l2nt) **9**

[7.1 Detailed Task List](#_sc25phlqwyw6) 10

[Deliverable 1:](#_joediqf21ttq) 11

[**8. Team Management Activities and tools**](#_gbis2pq8ji1o) **14**

[Shared git workspace using GitHuB. A repository that contains all the project resources. Each member pushed and pulled his/other code.](#_rnuahcn0jhsh) 14

[Code Reviews: For each patch pushed, at least one developer did the code review. Sessions were conducted between the author and the reviewer; and fixes pushed accordingly.](#_pwdrmhk2lp4a) 14

[FaceBook Group: for coordinating the meetings and syncing between members.](#_dlparv8sanr6) 14

[**9. Challenges**](#_sttcbh9zeeo7) **17**

[**10. Basic Components**](#_pxsrs3wih15l) **18**

[10.1 Functionalities provided by the multipart player as the following:](#_8qusitek724a) 20

[10.2 System Screenshots:](#_yrtrvxqouv6y) 21

[**11. System Design and Architecture**](#_wff5goed7t17) **23**

[11.1 System Architecture](#_50uvohnlsx3u) 23

[11.2 Updated UML Diagrams](#_n98uhcrez7fu) 24

[11.2.1 Class Diagram](#_yj8n74wttkxd) 24

[11.2.2 Use Case Diagrams](#_9eio2crggv5) 25

[Main Use Cases Description:](#_ncp2lbmetw1g) 25

[11.3 Sequence Diagram](#_j06p4pfr86j6) 28

[11.3 State machine diagram](#_j8hb3bkici3e) 29

[**12. System Testing: Bugs Tracking system and Bugs List**](#_p0mcj8cj7ykn) **31**

[12.1 Unit tests](#_2o7gapf1n3ep) 31

[12.2 Test coverage](#_2sg5kobz15z3) 32

[**13. Bug Tracking System: Bugzilla**](#_2dtifjxo5v4o) **33**

[**14. Bugs List**](#_ur25qh69phjo) **34**

[**15. Test Cases Results and Logs**](#_ff5nt75w8xce) **35**

[Logs](#_b9wweknh0ouw) 40

# 

# 

# **Table of Figures**

Figure [1](#mmgosikv1sxu) JavaDoc Example  
Figure [2](#a93o1jlemh78) Deliverable 1 To-Do List  
Figure [3](#bltw1guufw06) Deliverable 2 To-Do List  
Figure [4](#ul74u5he7g24) Deliverable 3 To-Do List  
Figure [5](#xif6wovas2ww) Group Wiki Home Page  
Figure [6](#fyshtb9hffr1) Group Wiki site map  
Figure [7](#d33evogkwaq9) MoM example  
Figure [8](#b4mmpfsu70o2) System Architecture  
Figure [8](#bndijn4whvjs) Class Diagram  
Figure [9](#athwvwd5kir8) Use Case Diagram   
Figure [10](#rmxj9n1qoxrh) Sequence Diagram  
Figure [11](#7hc8p0lq79vc) Finite State Machine  
Figure [12](#i5671eklrfkv) Project hierarchy

# 

# **Table of Tables**

Table 1: Bugheaven-bugzilla credentials

Table 2: Project Deliverables and roles

Table 3: MultiPart Downloader basic components

Table 4: Multipart Downloader functionalities

Table 5: Test Cases

# **1.Project Links**

## [Source Code on GitHub](https://github.com/Ctrn/ConstDownloadManager/tree/master/MultiPartDownloader)

## [Group Wiki](https://sites.google.com/site/downloadmanagerproject/)

## 

## Deliverable 1: [Design Document](https://docs.google.com/document/d/1Y3S1iacV6dy4Qdjz_n8_cosU5j1_2lCNLdO3E9m4zKs/edit)

## Deliverable 2:

* + [Documentation](https://docs.google.com/document/d/1HOAzapKmK8Q5iJPv2utbCISX1AQBorOlDLUyMfkkbYE/edit)
  + [Source Code on GitHub](https://github.com/Ctrn/ConstDownloadManager)
  + [Server Instances](http://ec2-54-186-190-13.us-west-2.compute.amazonaws.com:8080/manager/html/)
* Project Update: Compression and GUI
  + [Source Code on GitHub](https://github.com/Ctrn/ConstDownloadManager/tree/master/MultiPartDownloader)

## [JavaDoc](https://hanannamrouti.github.io/JavaDoc/)

## 

## [Bugzilla](http://www.bugheaven.com/bugzilla/ConstructionGroupBZU/) and [BugList](http://www.bugheaven.com/bugzilla/ConstructionGroupBZU/buglist.cgi?bug_status=__open__&list_id=15&order=relevance%20desc&query_format=specific)

Credentials for logging to Bug Heaven- BugZilla:

Table 1: Bugheaven-bugzilla credentials

|  |  |  |
| --- | --- | --- |
| Link | User Name | Password |
| <http://www.bugheaven.com/bugzilla/ConstructionGroupBZU/> | aafaneh@birzeit.edu | aafaneh@birzeit.edu |

# 

# **2. Scope**

Multipart downloader works as a medium between the hosting domain, servers of a file and the requester how want to stream the file, this approach is provided by dividing the file into a smaller segments located in different server to be accessed and downloaded in cases of needed the whole file, the file is represented as a set of segments that can be merged together to provide the requested file, the multipart downloader is responsible for assembling and comprising the downloaded segments, in this document we will be overviewing the elicited requirements and the approached design, and its associated implementation to achieve a reliable file download and viewing.

# **3. Introduction**

The purpose of this software design and implementation 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, in addition to implementation phase which includes the technologies used to achieve the objective intended by multipart downloader.

# **4. Goals and objectives**

The objectives of this application is solving the problem of downloading files with a relatively large sizes, where they are being segmented 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 in order for the requester of the file to view the provided stream.

5. Project Details

Development Methodology

Iterative development

Technology Used

Java

Hosting for data

Amazon Web Service (AWS): micro instance

Tomcat

Open bugs

BugHeaven: Free Bugzilla Hosting

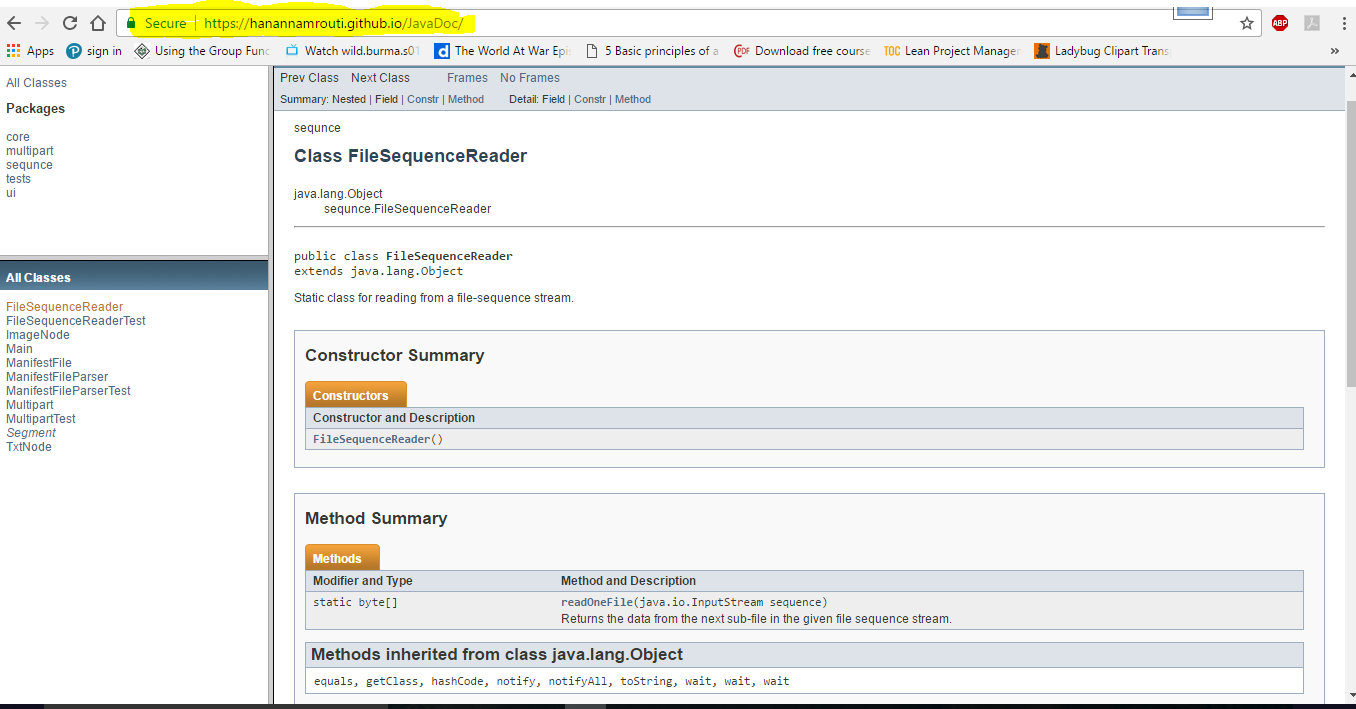
# **6. JavaDoc**

JavaDoc is an Eclipse free plugin to generate headers to source code , it designed to offer an HTML website will all packages classes and their methods, give as any information about them like return type,we used a free jar from eclipse market [JavaDoc tool.jar](http://www.java2s.com/Code/Jar/c/Downloadcomsunjavadocjar.htm)

## Why JavaDoc:

* Tell us what classes do and how it does now.
* Understandable by non programmer user and clients.
* Easy to update and maintain.
* Good for Authors history.

The link for our project is <https://hanannamrouti.github.io/JavaDoc/>



[Figure 1:](https://hanannamrouti.github.io/JavaDoc/)  JavaDoc Example

# 

# **7. Team Members and Roles**

Software Development Process we followed and the roles of team members in each phase as shown in Table 2 below:

Table 2: Project Deliverables and roles

|  |  |
| --- | --- |
| Initial Investigation and requirement understanding | All |
| Team Collaborating Tools | Source control (GitHub) repository initialization: Citreen  Group Google Wiki construction: Hanan  Google Wiki management: Hanan, Citreen, Saad |
| Design | 1. All: Documentation 2. Saad    1. Class Diagram.    2. Sequence Diagrams    3. Use case Diagram. 3. Citreen    1. Documentation Initialization.    2. System Architecture.    3. Classes Definition. 4. Hanan    1. State Machine Diagram |
| Development | 1. Documentation: Saad, Citreen. 2. Saad:    1. Build Project and initial commit on GitHub    2. Node    3. TextNode    4. ImageNode    5. Sequence 3. Citreen    1. Build Project and initial commit on GitHub    2. Logging integration    3. Main Class and simple GUI implementation    4. Segment class 4. Hanan:    1. JavaDoc creation and integration with GitHub |
| Project Update: Compression and GUI | 1. All:    1. Initial Requirement Analysis 2. Saad:    1. GUI integration    2. Compression Integration 3. Citreen:    1. Slideshow |
| Test Data Hosting | 1. Citreen:    1. Investigation and server selection (aws).    2. AWS micro instance and Tomcat setup.    3. Creating Web project and segments, text and image.    4. Exporting and hosting data. |
| Testing | 1. Citreen:    1. Test Plan and test cases design 2. Hanan:    1. Unit Tests    2. Test cases execution    3. Code Coverage    4. Upload bugs to Bugzilla after testing |
| Bug Tracking System setup | 1. Citreen:    1. Investigation and tool selection 2. Saad:    1. Bug Heaven configuration and Users creation |
| Final Report and documentation | 1. All:    1. Review, update specific parts 2. Citreen, Saad:    1. Report Structure, review, styling |

## 7.1 **Detailed Task List**

We documented the tasks for each team member, in each deliverable, on google wiki task list, each task has the status of: new, in progress, done, as the following:

### Deliverable 1:



Figure 2: Deliverable 1 To-Do List

Figure 3: Deliverable 2 To-Do List

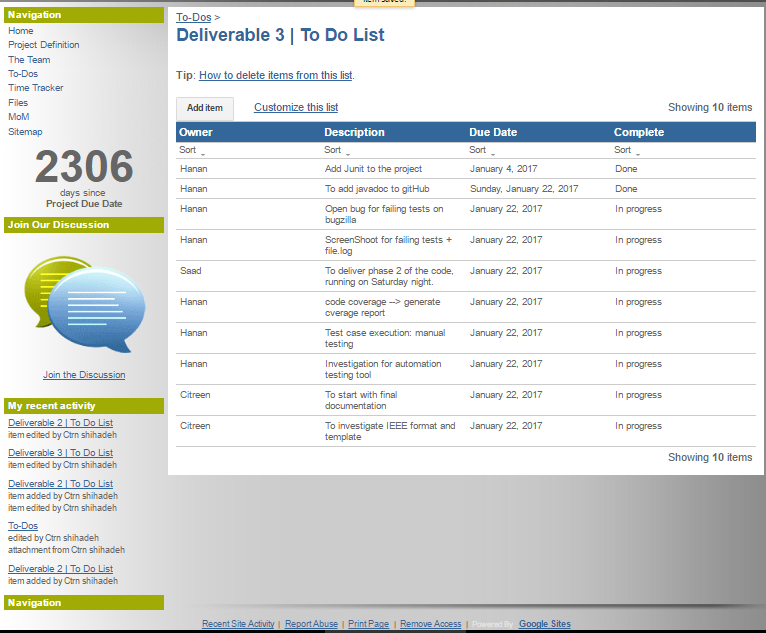
Figure 3: 

Figure 4: Deliverable 3 To-Do List

# **8. Team Management Activities and tools**

## [**Shared git workspace**](https://github.com/Ctrn/ConstDownloadManager) using GitHuB. A repository that contains all the project resources. Each member pushed and pulled his/other code.

## Code Reviews: For each patch pushed, at least one developer did the code review. Sessions were conducted between the author and the reviewer; and fixes pushed accordingly.

## [**FaceBook Group**](https://www.facebook.com/groups/249302272156886/?ref=aymt_homepage_panel): for coordinating the meetings and syncing between members.

1. Google Collaboration Tools:
   1. [Google Sites](https://sites.google.com/site/downloadmanagerproject/): we created a group wiki that includes all the project resources, including MoM and files, tasks lists, and others. Figure 5 shows the site home page, and figure 5 shows the site map.



Figure 5: Group Wiki Home Page

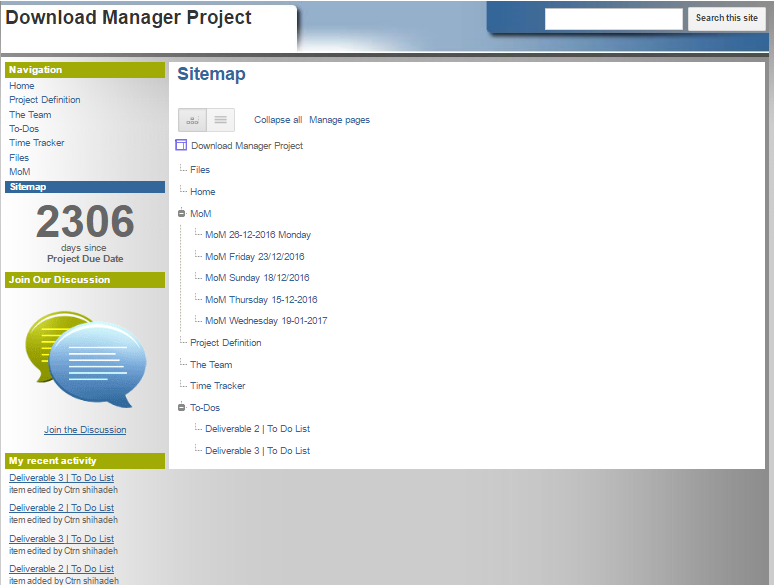


Figure 6: Group Wiki site map

* + 1. [Meeting Minutes](https://sites.google.com/site/downloadmanagerproject/mom). Each meeting was documented. Example of MOM, is shown in figure 7. The structure of MoM includes Agenda and Action items.

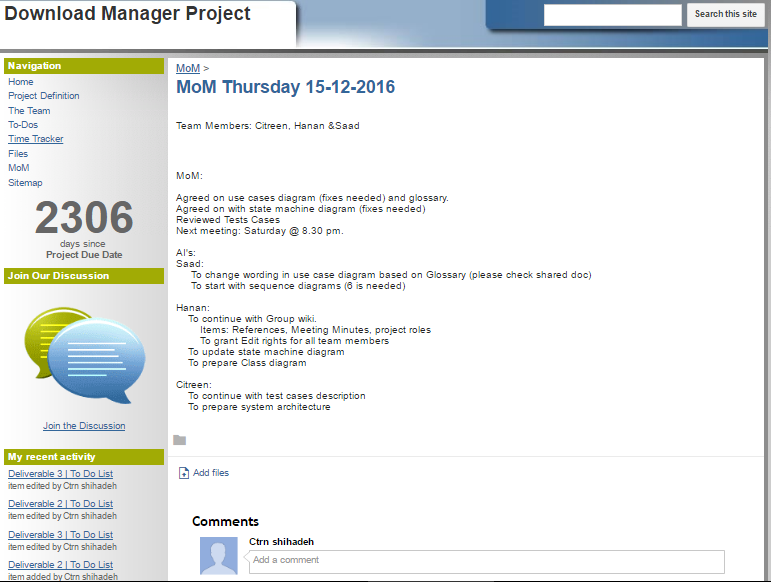


Figure 7: MoM example

* + 1. [Task List](https://sites.google.com/site/downloadmanagerproject/to-dos): please refer to [Team Members and roles](#_8rftf8o1l2nt) section for detailed info.
  1. Google Drive: sharing documents and real time collaboration. All history changes are viewable and we can post comments offline.
  2. Google Hangouts: meetings and syncing.

# 

# **9. Challenges**

1. No test data, we spent effort on creating server and instances for covering test cases, which were conducted from problem definition and description.
2. Dealing with image data (non textual) for creating test data.
3. Implementing slideshow part of the code.

# 10. Basic Components

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

Table 3: MultiPart Downloader basic components

|  |  |
| --- | --- |
| 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. |
| 1. Manifest file | The object the holds the references for the needed segments to be downloaded to create the complete file. It also may contain a reference to other Manifest files, and to create a nested Manifest file. |
| 1. Segment | A small part of the whole file to be downloaded, each segment will contain at least one downloading mirror. |
| 1. Parser | The module responsible of tokenizing the Manifest file content of URL to get the downloading links for each requested file segment; and then convert it to a downloading object. |
| 1. 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. |
| 1. Sequence | Its Responsible for Sequencing the fetched stream of the associated URL. |
| 1. Logging | We used<http://logging.apache.org/log4j/1.2/> We used different Logging levels in the project classes in instead of Java Logging API as it is more usable in defining Logger levels. This library is included as a jar file within the project. We also have referred to the official documentation to learn and use it. This framework is facade and have direct calls to Java Util Logging, which provides us a variety of logging implementations.  Construction principle: We implemented an abstract framework for Logging so we do not refactor the code when we are changing the logger classes.  including:  1. INFO: informative messages for the user  2. ERROR: inform the user with errors occurred while execution  3. DEBUG: example usage on function entering and printing some variables, not needed to be showed to the user. |

## 

## 

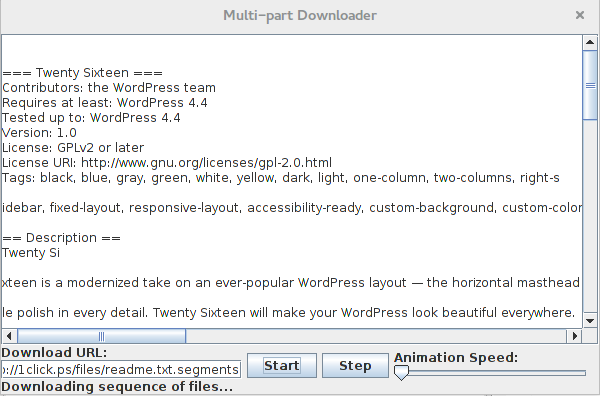
## 

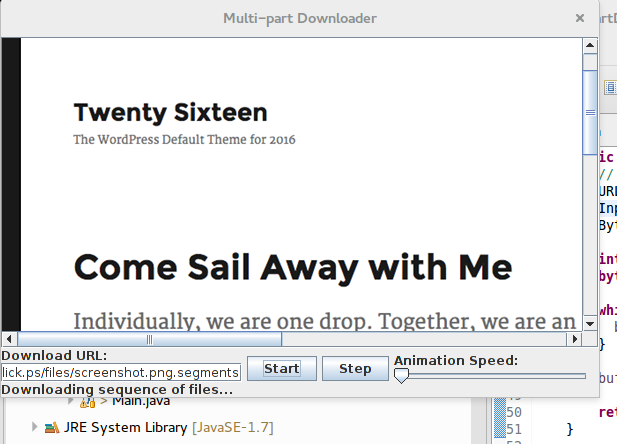
## 10.1 Functionalities provided by the multipart player as the following:

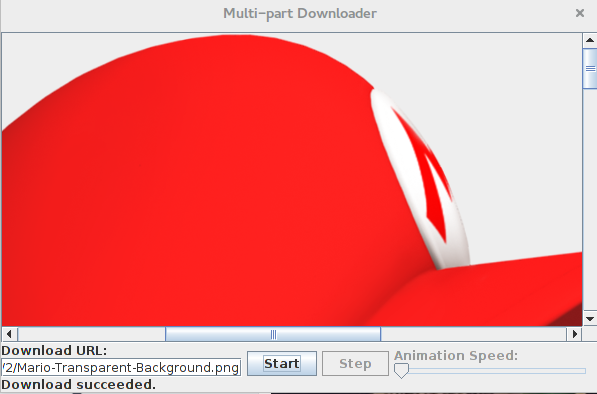
Table 4: Multipart Downloader functionalities

|  |  |
| --- | --- |
| 1. Stream | The functionality of executing the download, performed by connecting to the server that hosts the downloading link. |
| 1. Assemble | The functionality provided by the multipart player, in which all the downloaded segments are being put in order to form the final file. |
| 1. Validate | The functionality of checking the health of the Manifest file, with valid segment separators, as well as validating the Manifest file type, and validating the URL located in the the manifest file and checking the condition of the reference segment “File is Found, Redeunt of Files”. |
| 1. Download Text files | Which holds the responsibility for download text files, and processing the fetched data to the appropriate Data type for the user view. |
| 1. Download Image files | Which holds the responsibility for download image files, and processing the fetched data to the appropriate Data type for the user view. |
| 1. Slide Show | Which is associated with the viewing of the downloaded segments of the file to be viewed in a sequential manner |
| 1. Compression | Which associated with comprising the downloaded stream. |

## 10.2 System Screenshots:







# **11. System Design and Architecture**

## 11.1 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 8 represents the block diagram for the basic components listed in the previous section

# system arch.png

Figure 8: System Architecture

## 

## 11.2 Updated UML Diagrams

## 11.2.1 Class Diagram

A static aspects representation of the solution that highlights the relation and the operation, and the attributes of the communicating classes.

Overview view of the main classes:

1. Main: the class is associated with the previewing the download and merged segments which includes:
   1. Single File.
   2. Segments.
2. ManifestFile: responsible for determination of the implementation of the segment based on the file type associated with the requested URL.
3. ManifestFileParser: checks the the validity of the URLs fetched from the content of the manifest file:
   1. File not found response of the server 404 code.
   2. Remove Redundant URL based after check availability.
4. Multipart.
5. FileSequnceReader.
6. Segment.
7. TxtNode: which is responsible for opening the URL and processing, the data fetched, in converting the stream data to its appropriate viewing data type .
8. ImageNode: resembles the TxtNode but with different implementation to match the image.

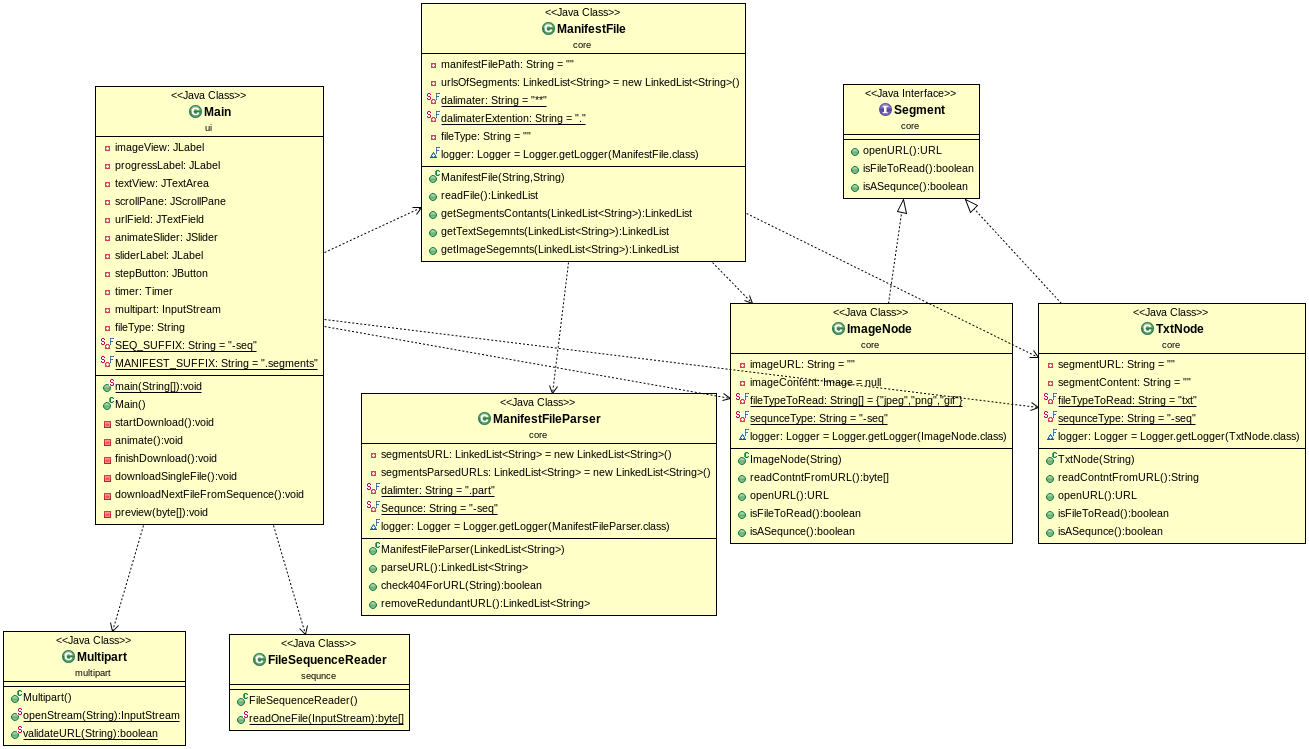


Figure 8: Class Diagram

## 11.2.2 ***Use Case Diagrams***

Use case view describes the set of behavioral aspects between the actors in the boundary of the application domain, and highlights the type of involvement of the actors with a high-level view establish the main goal of reliable file streaming.

Actors

1. Streamer

Initiate the Streaming of file from the hosted Service

1. System

Triggers the Operation of Tokenizing and partitioning of files to smaller segments and the operation involves storing on multiple hosting machines.

1. Multi Part Downloader

Intermediates “Secondary Actor” the File stream action, which is selected is selected by the Streamer, and hosted on Machines.

### Main Use Cases Description:

1. Choose File: Triggered by the Streamer, which identifies the need for the stream to view a certain file.
2. Parse File: Cared out by the MultiPart Download Manager, which find the needed segments “Tokens” that make up the file.
3. Find URL: identifies the URL associated with the tokenized part of the file.
4. Check URL: Validates the URL eligibility for the download of the required segment.
5. Download Segment: which involves the download of the parsed segment from a valid URL, with the an extension of restraining the download operation of segment due to failure state.
6. Sort Stream: involves the ordering the downloaded segments based on the parsed location of the file.
7. Stream File: viewing the sorted downloaded Segments in the designated view location for the Streamer.
8. Compress Segment:

|  |  |
| --- | --- |
| Term | Description |
| MultiPart Download Manager | Solution used for reliable file stream for distributed segmentation of the file. |
| Streamer | A stakeholder, a user of the Solution for the file streaming |
| Failure state | The status which can occur due:   1. Download Failure. 2. Unreachable host. 3. Corrupted file/segment. 4. Unresolvable URL. |

updated-UseCase.png

Figure 9: Use Case Diagram

## 11.3 ***Sequence Diagram***

The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. sequence diagram can be used as a requirements document to communicate requirements for a future system implementation. During the requirements phase of a project, analysts can take use cases to the next level by providing a more formal level of refinement. When that occurs, use cases are often refined into one or more sequence diagrams.

Object included Are as Follows:

1. Streamer: Reposivel of triggering the event by requesting the file using the URL.
2. Main: Main Object is the UI associated with the Streamer to navigate in, and provides a view platform of the acquired data.
3. Manifest File: an object responsible for calling other object for parsing and validating the requested URL
4. Manifest File Parser: holds the responsibility for parsing the accrued URL passed by the Manifest.
5. Segment: used as a container for the parsed URL and the fetched data from the associated Link
6. Comprise: used to comprise the segment content.

## SequnceDigram.png

Figure 10: Sequence Diagram

## 11.3 ***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

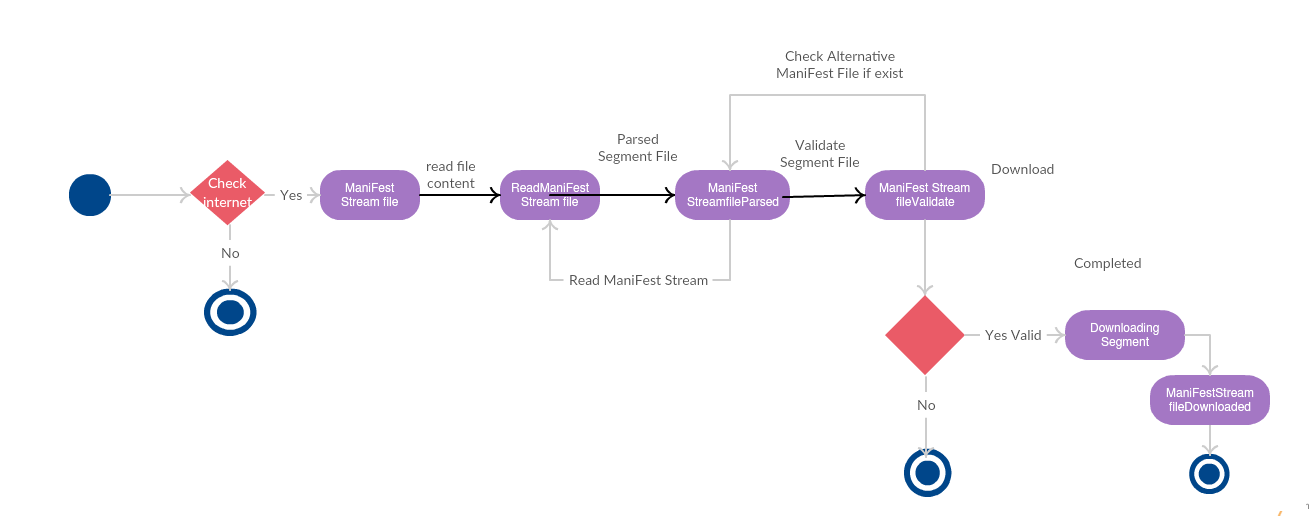


Figure 11: Finite State Machine

# **12. System Testing: Bugs Tracking system and Bugs List**

## 12.1 **Unit tests**

We include jar file for [Junit4-12](http://junit.org/junit4/) it’s free plugin provided by eclipse market ,Junit helps more in manual testing and in debugging, for efficiency we attach another jar [hamcrest-all-1.3](http://www.java2s.com/Code/Jar/h/Downloadhamcrestall13jar.htm).jar

it is a framework that provide writing software tests in Java. It give ability for creating and customized matchers ,we include jar file used to give us more matcher for Junit (unit test) Like assertThat ().

Our unit test under package test covered most of project code

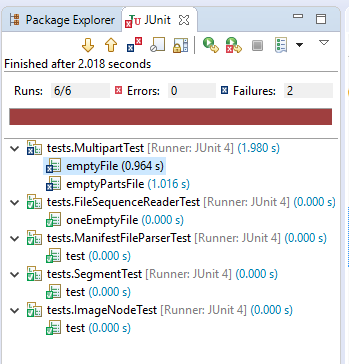
1. FileSequenceReaderTest.
2. ImageNodeTest
3. ManifestFileParserTest
4. MultipartTest
5. SegmentTest 

Figure 12: All UnitTests

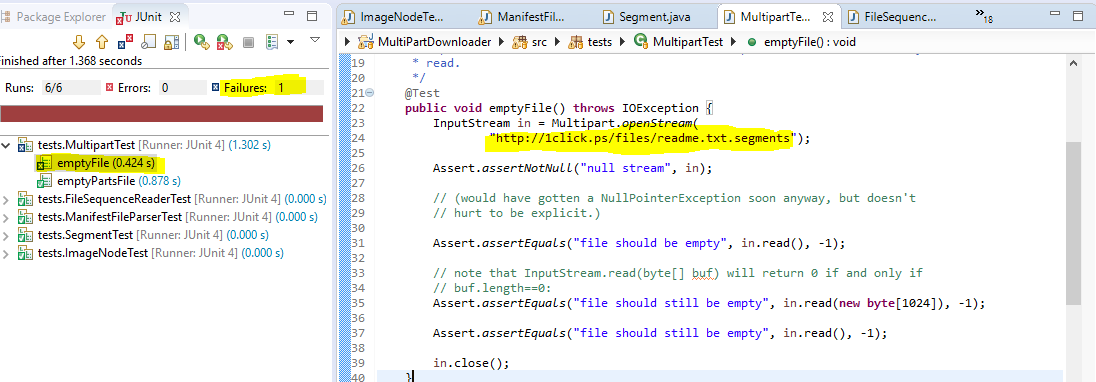


Figure 13: Fail unit test

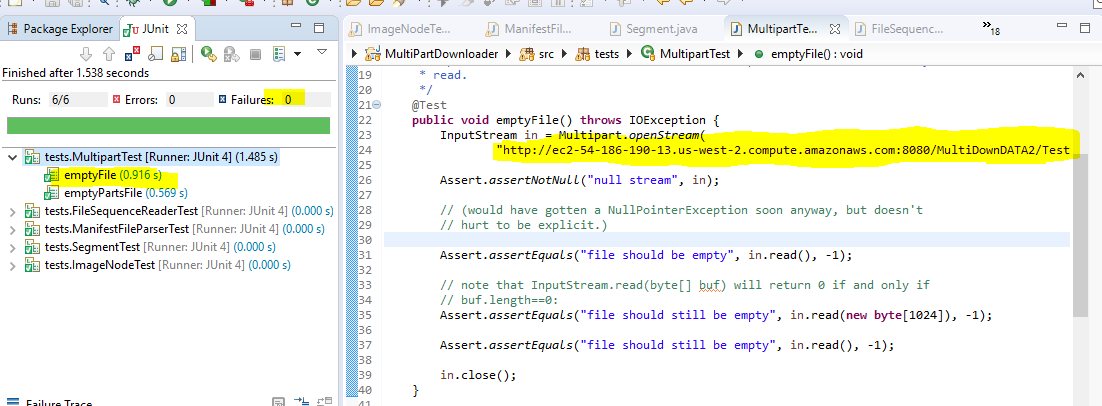


Figure 14:Pass unit test

## 12.2 **Test coverage**

To specify which methods are tested and covered using test explorer ,it’s show table with percentage give us which classes and methods are run and which code has been tested.

We use EclEmma is a free Java code coverage tool for Eclipse, available under the Eclipse Public License. It brings code coverage analysis directly into the Eclipse

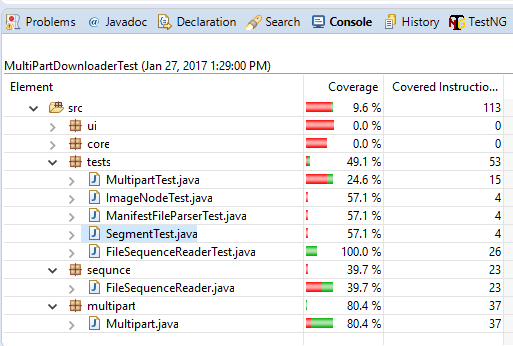


Figure 15: Test coverage result

# **13. Bug Tracking System: Bugzilla**

We have created a project on [BugHeaven](http://www.bugheaven.com/); that offers free hosting for Bugzilla up to 5 users per project. From there, we documented all bugs found during development and testing process. We kept this list updated with the status of the bugs, and each tester was responsible on testing closing the bug after the developer had fixed it.

# **14. Bugs List**

# bugs.PNG

Figure 16: BugZella All bugs

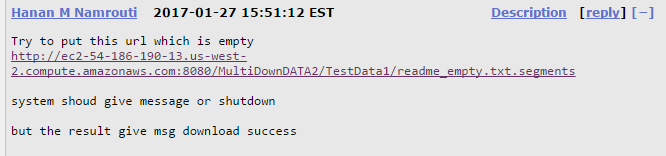


Figure 17: bug example

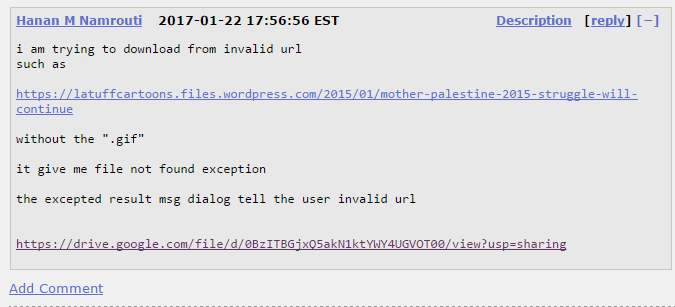


Figure 18: bug example

# **15. Test Cases Results and Logs**

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.

Table 5: Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Black Box Testing | | |  | |
| Good Flow test cases | | | | |
| Case ID | Description | Steps | Expected result | Actual result |
|  | Download URL is valid | 1. Run manifest file 2. Download link is valid and so download starts | Success | Success |
|  | 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 | Success |
|  | 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 | Success |
|  | Download link is accessible | 1. Run manifest file 2. Download starts successfully | Success | Success |
|  | 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 | Failed |
|  | 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 | Failed |
|  | 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 | Success |
|  | 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 | Success |
|  | 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 | Failed |
|  | 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 | Success |
| Bad flow test cases | | | | |
|  | 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 | Success |
|  | Download from invalid URL (error in the format) | 1. Create Manifest file with invalid URL 2. Download should not start 3. Informative error message should be received. | Fail | Success |
|  | 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 | Success |
|  | 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 | Failed |
|  | 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 | Failed |
| White-box testing | | | | |
| Good Flow test cases | | | | |
|  | Valid URL formats for download mirrors | 1. Get a Manifest file 2. Check URL validity:    1. Format    2. Language    3. Special characters | Success | Success |
|  | Valid segments separator | 1. Get a Manifest file 2. Check segment separator validity with \*\* between each segment and the other one | Success | Success |
|  | 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 | not-Executed |
|  | 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 | Failed |
|  | 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 | Failed |
|  | 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 | Success |
| Bad Flow test cases | | | | |
|  | Download from invalid manifest file | 1. Get invalid manifest file 2. Check segments invalidity 3. Dwonload should not start 4. Informative error message is displayed to the use it | FAIL | Success |

## Logs



Figure 19:Log File screenshots

Attached

[Logs file URL](https://github.com/HananNamrouti/JavaDoc/blob/master/logs.txt)