Streaming Using JavaScript

Technical Project – ETN5

POLYTECH' NANTES

2016

Authored by: Bilong HUANG, Jing KE

Contents

Figures

Streaming Using JavaScript

Technical Project – ETN5

1. Introduction

Our project is to implement multimedia streaming using C-S (client-server) pattern with JavaScript.

1.1. Context

Streaming data such as binary data, audio, video and files has been playing an important role among modern sophisticated technologies in era of network information. As one of the most popular technology, JavaScript has no reason to deny implementing streaming for multimedia.

Furthermore, since JavaScript has not only been applied in front ends of websites for browsers but also been utilized in both server-side and client-side applications, it would be meaningful to implement media streaming using JavaScript crossing over different platforms.

1.2. Objectives

Implement a basic light-weight framework using JavaScript by media streaming protocols to send, stream and pipe binary data in a bidirectional way between server-side (Node.js) applications and client-side applications.

2. Organization

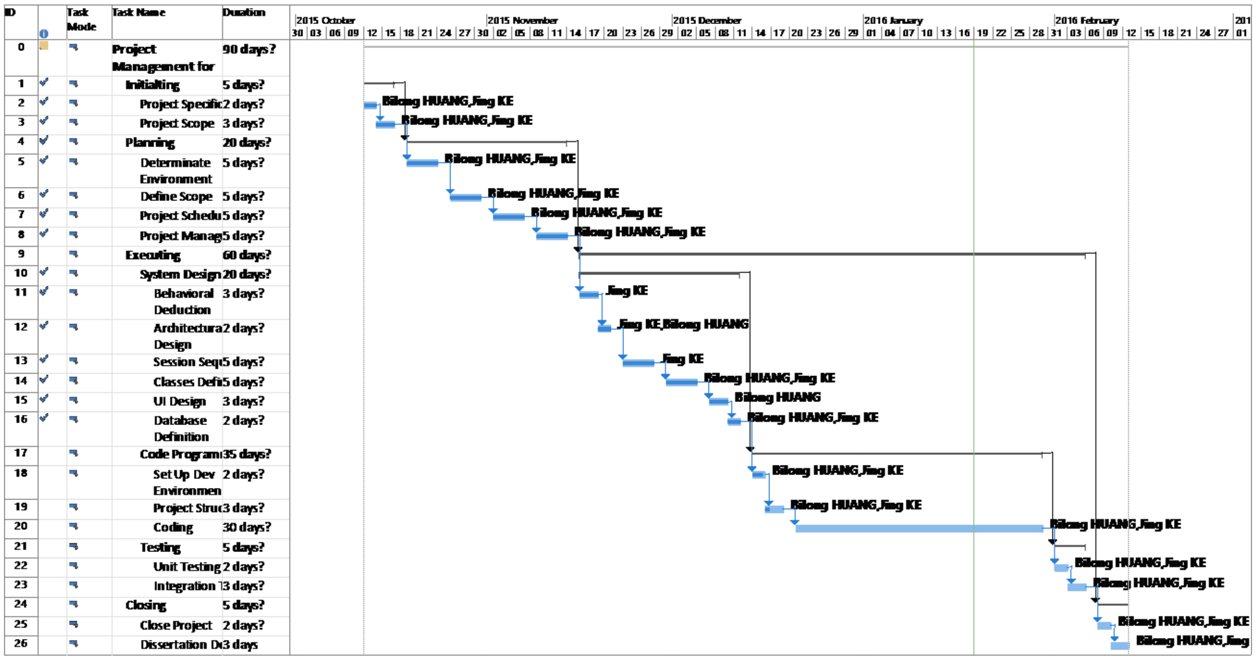
In this project, we have 2 developers. We should work as a team so we need to organize our works. To reach our goals, we divided the project into several parts and used some software to facilitate our cooperation.

2.1. Project Management

We divided the project into 4 parts and made a schedule to manage our tasks.

1. **Initiating:** determinates project specification and project scope.
2. **Planning:** represents preliminary conceive plan.
3. **Executing:** specifies implementation details.
4. **Closing:** goes into the final period of our project.

The figure following is the Gantt chart:



ID		Task Mode	Task Name	Duration
0			Project Management for	90 days?
1	✓		Initialting	5 days?
2	✓		Project Specific	2 days?
3	✓		Project Scope	3 days?
4	✓		Planning	20 days?
5	✓		Determinate Environment	5 days?
6	✓		Define Scope	5 days?
7	✓		Project Schedu	5 days?
8	✓		Project Manag	5 days?
9			Executing	60 days?
10	✓		System Design	20 days?
11	✓		Behavioral Deduction	3 days?
12	✓		Architectura Design	2 days?
13	✓		Session Seq	5 days?
14	✓		Classes Defi	5 days?
15	✓		UI Design	3 days?
16	✓		Database Definition	2 days?
17			Code Program	35 days?
18			Set Up Dev Environmen	2 days?
19			Project Struc	3 days?
20			Coding	30 days?
21			Testing	5 days?
22			Unit Testing	2 days?
23			Integration T	3 days?
24			Closing	5 days?
25			Close Project	2 days?
26			Dissertation D	3 days

FIGURE 1 - GANTT CHART

We divided the coding part to many small tasks:

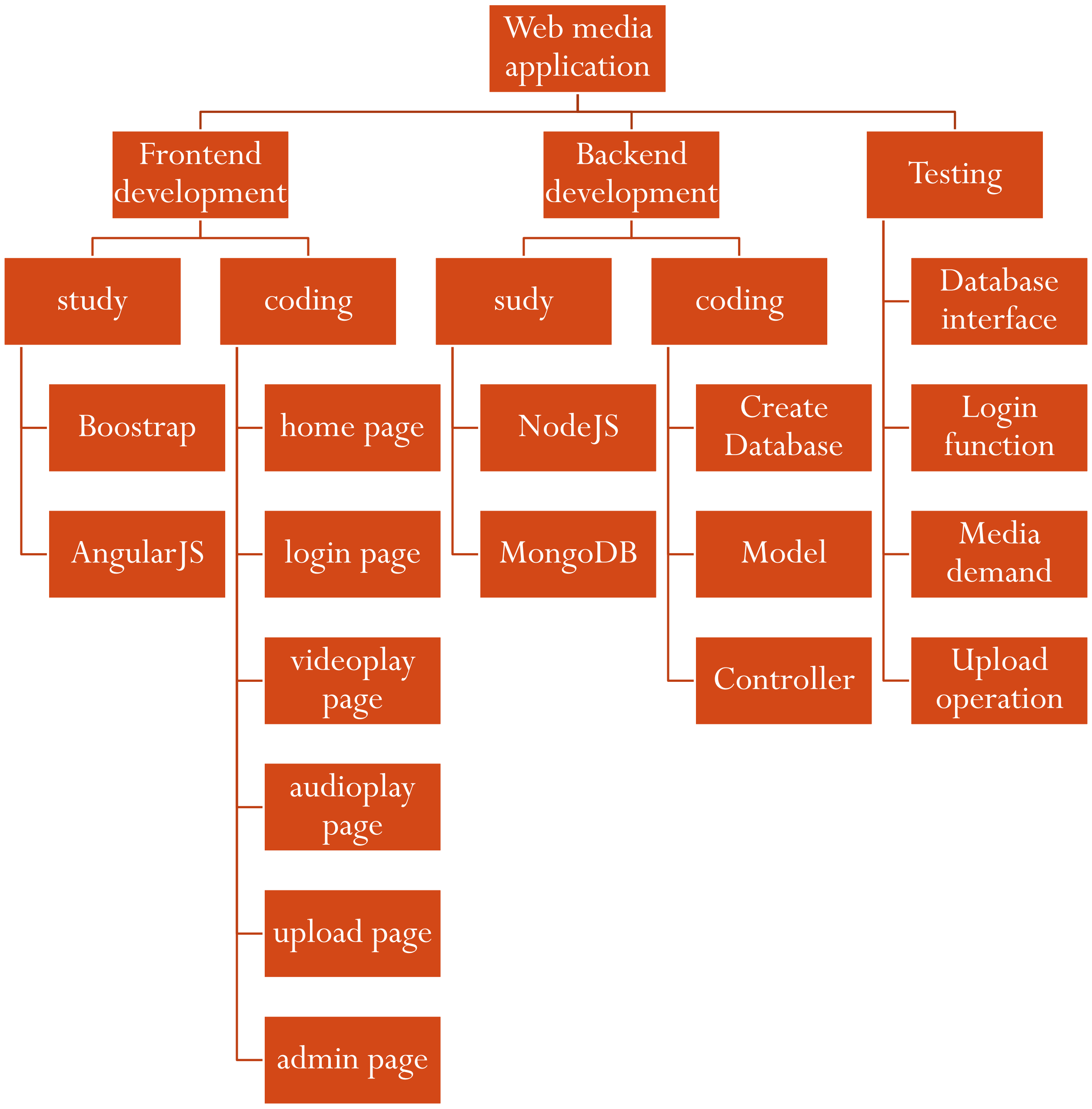


Figure 2 – Coding Tasks

The following is the divided tasks of model and controller:

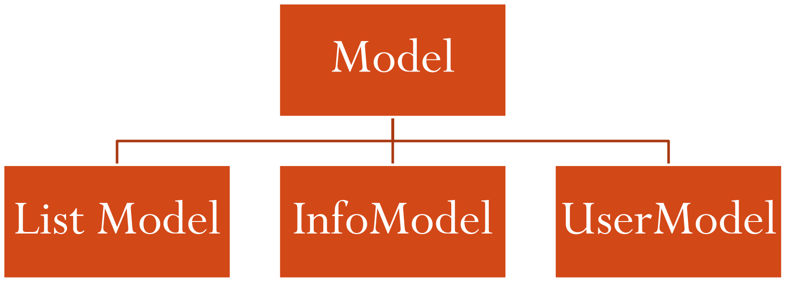


FIGURE 3 – MODELS TASKS

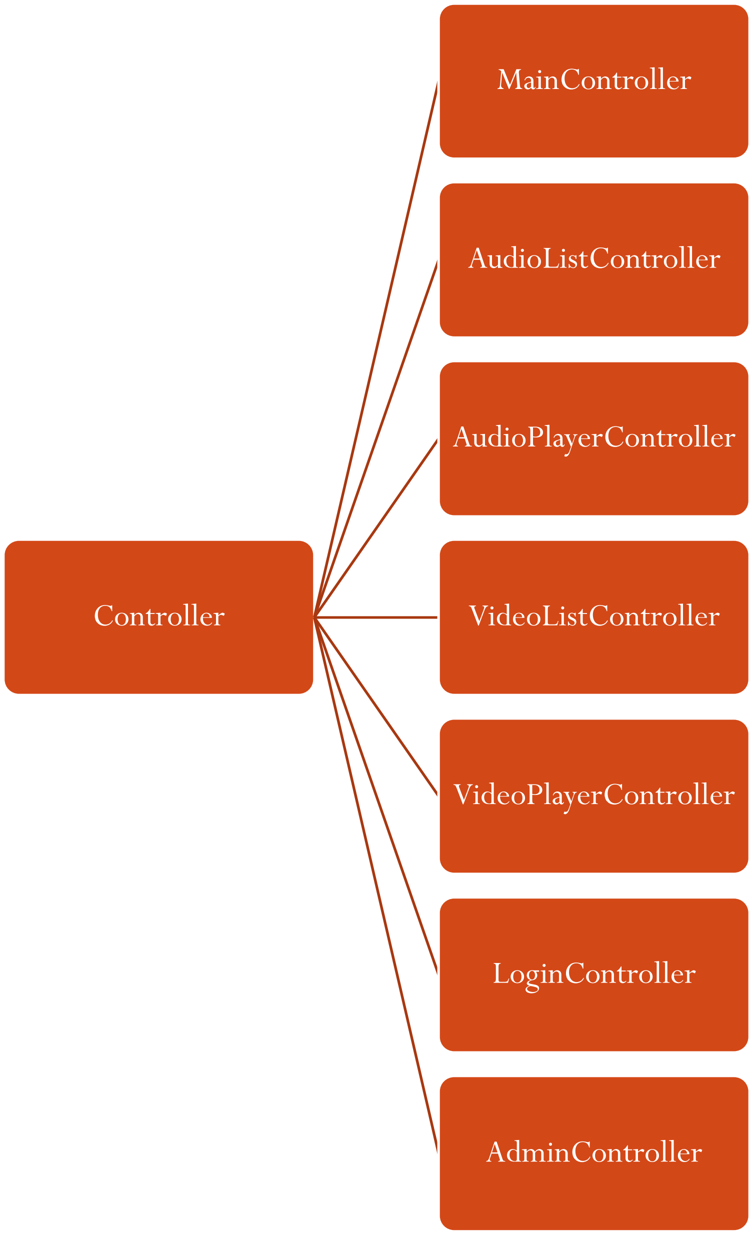


FIGURE 4 – CONTROLLER TASKS

After dividing the tasks, we made 3 dead line forms:

Frontend Development

Tasks	Wait	Doing	Done	Dead Line
Boostrap			√	01.17
AngularJS			√	01.17
Home Page			√	01.22
Login Page			√	01.22
Video Play Page			√	01.26
Audio Play Page			√	01.26
Upload Page			√	01.29
Admin Page			√	01.29

Backend Development

Tasks	Wait	Doing	Done	Dead Line
NodeJS			√	01.17
MongoDB			√	01.17
Create Database			√	01.17
Model			√	01.29
Controller			√	01.31

Testing

Tasks	Wait	Doing	Done	Dead Line
Database Interface			√	02.02
Login Function			√	02.04
Media Demand			√	02.07
Upload Operation			√	02.09

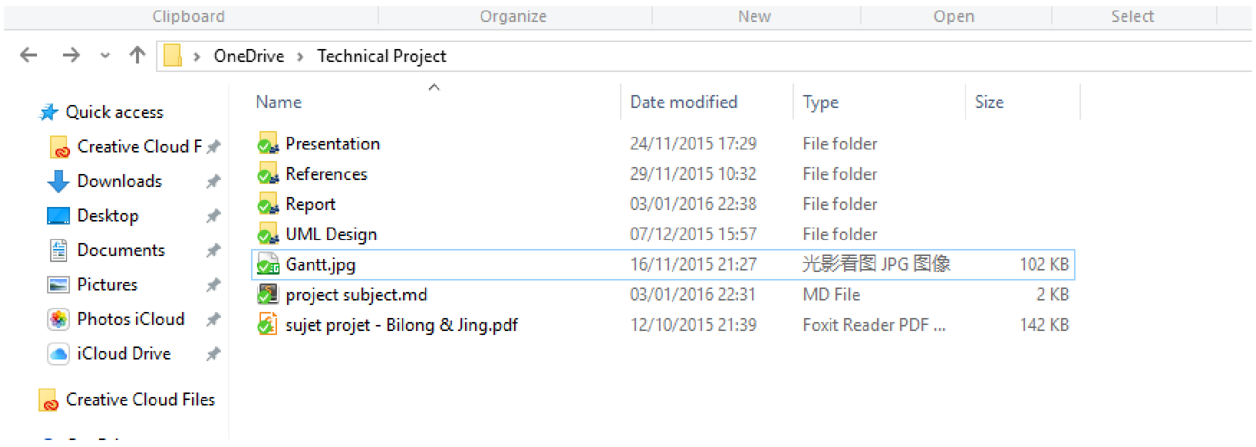
2.2. Cooperation Method

For sharing our files and code about this project, we use OneDrive and GitHub:

2.2.1. OneDrive

OneDrive is a file hosting service that allows users to sync files and later access them from a web browser or mobile device. Users can share files publicly or with their contacts.

The figure following is the organization of the files:



Name	Date modified	Type	Size
Presentation	24/11/2015 17:29	File folder	
References	29/11/2015 10:32	File folder	
Report	03/01/2016 22:38	File folder	
UML Design	07/12/2015 15:57	File folder	
Gantt.jpg	16/11/2015 21:27	光影看图 JPG 图像	102 KB
project subject.md	03/01/2016 22:31	MD File	2 KB
sujet projet - Bilong & Jing.pdf	12/10/2015 21:39	Foxit Reader PDF ...	142 KB

FIGURE 2 – ONEDRIVE FILES

2.2.2. GitHub

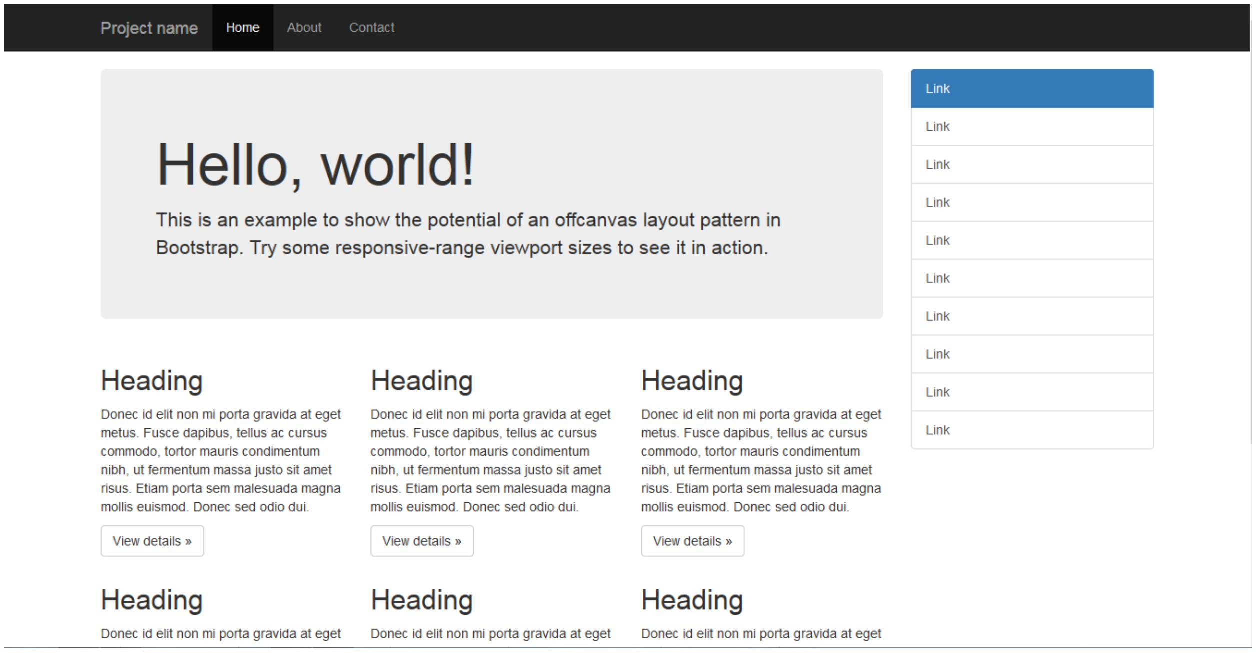
GitHub is a Web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features as bug tracking, feature requests, task management, and wikis for every project.

3. Technologies

In our project, we use some auxiliary technologies to facilitate our development.

3.1. Bootstrap

In order to ease the development of dynamic web application, we use Bootstrap that contains HTML-based and CSS-based design templates for typography, forms, buttons, navigation and other interface components.



Hello, world!

This is an example to show the potential of an offcanvas layout pattern in Bootstrap. Try some responsive-range viewport sizes to see it in action.

Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

Donec id elit non mi porta gravida at eget

Heading

Donec id elit non mi porta gravida at eget

Heading

Donec id elit non mi porta gravida at eget

FIGURE 2 - EXAMPLE OF A WEBPAGE USING BOOTSTRAP FRAMEWORK

More information about Bootstrap can be found here: http://getbootstrap.com/ .

3.2. AngularJS

In order to simplify both development and testing of SPA (single-page application), AngularJS can provide us client-side MVC (model-view-controller) and MVVM (model-view-viewmodel) architectures.

ANGULARJS by Google

FIGURE 3 - ANGULARJS LOGO

More information about AngularJS can be found here: https://angularjs.org/ .

3.3. Node.js

In order to develop server-side web application, we use Node.js as a cross-platform runtime environment with JavaScript.

FIGURE 4 - NODE.JS LOGO

More information about Node.js can be found here: https://nodejs.org .

3.4. MongoDB

In order to make the integration of data easier and faster, we use MongoDB as database to develop JSON-like documents with dynamic schemas.

FIGURE 5 - MONGODB LOGO

More information about MongoDB can be found here: https://www.mongodb.org/ .

3.5. MEAN.JS

MEAN.JS is a full-stack JavaScript solution that helps you build fast, robust, and maintainable production web applications using MongoDB, Express, AngularJS, and Node.js.

More information about MEAN.JS can be found here: http://meanjs.org/ .

3.6. Grunt

We use the JavaScript task runner – Grunt to easier our development when performing repetitive tasks like compilation, unit testing, etc.

FIGURE 6 - GRUNT LOGO

More information about Grunt can be found here: http://gruntjs.com/ .

3.7. Postman

Postman helps us test our API.

FIGURE 7 - POSTMAN LOGO

More information about Postman can be found here: https://www.getpostman.com/ .

4. System Design

We use UML method to design our system.

4.1. Use Case Description

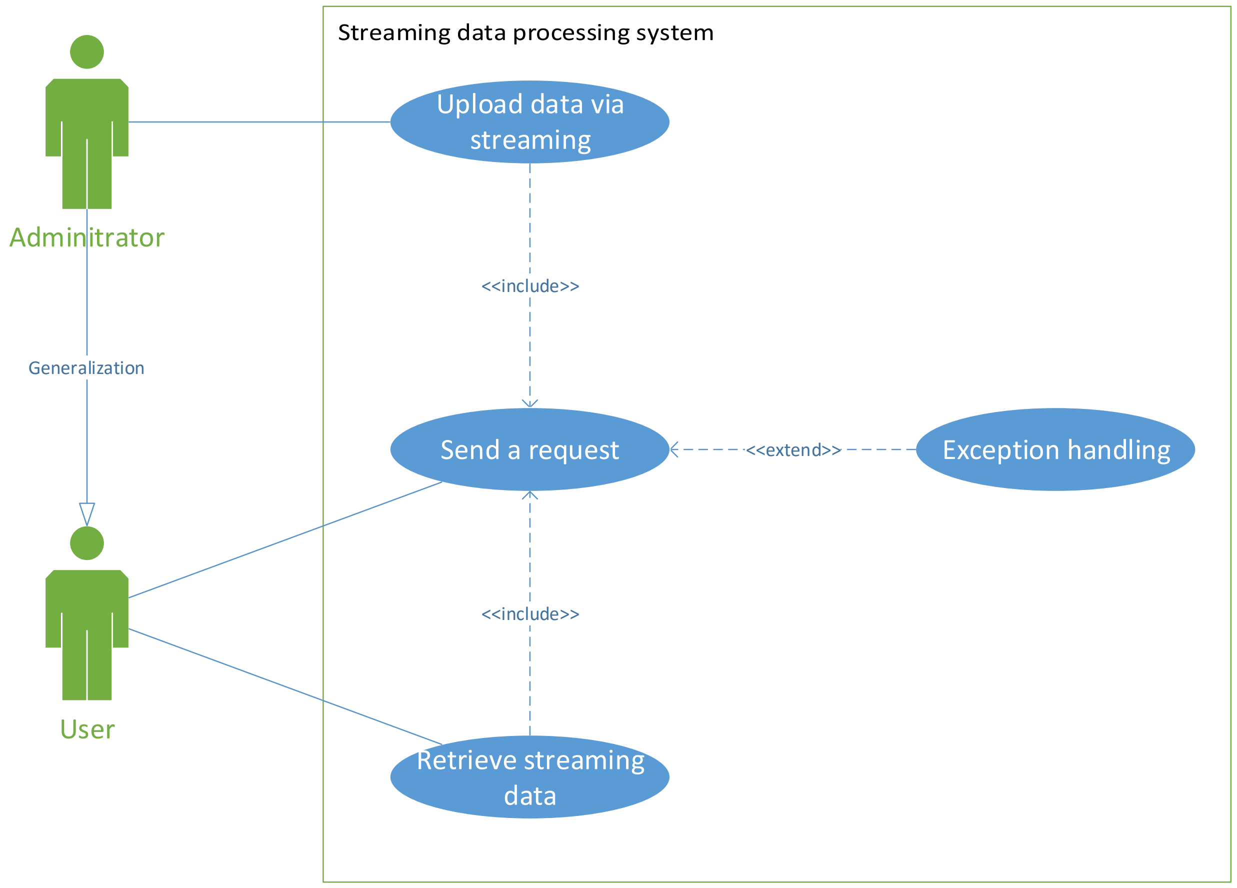


FIGURE 8 - USE CASE

- **Use Case:** Streaming data between client(s) and server(s)
- **Primary actor:** User, administrator
- **Scope:** A data stream processing system
- **Goal Level:** Basic system implementation
- **Brief:** The users upload multimedia data (audio/video) from clients onto servers and can see/retrieve from their client terminal the data uploaded while streaming.

- **Success Guarantees:**
    - The data is saved on the server through a streaming method and an updated viewed is shown
    - The data can be distributed from the server to all compatible devices via a urn apt
- **Preconditions:** Connected to the Internet
- **Triggers:** The user sends an upload (POST)/retrieve (GET) request to the server
- **Basic Flow:**
    1. The user sends a request to the server from their client terminals
    2. The server responds with streaming data
    3. The clients retrieve streaming data and present them to the user
- **Extensions:**
    a. Requests:
        1. A request sent by user should indicate an upload or a query operation
        2. Rerun step 1 (in the basic flow) if a request fails to reach the server and inform the user the failure operation
    b. Responds:
        1. A client should understand a respond status code and react with corresponding treatment
        2. Retrieve the required data from WS (WebSocket) content
    c. Presentation:
        1. Indicate the progress of operations
        2. Apply a functionality of cancelation if necessary

4.2. Activity modeling

4.2.1. Activity modeling (Client side)

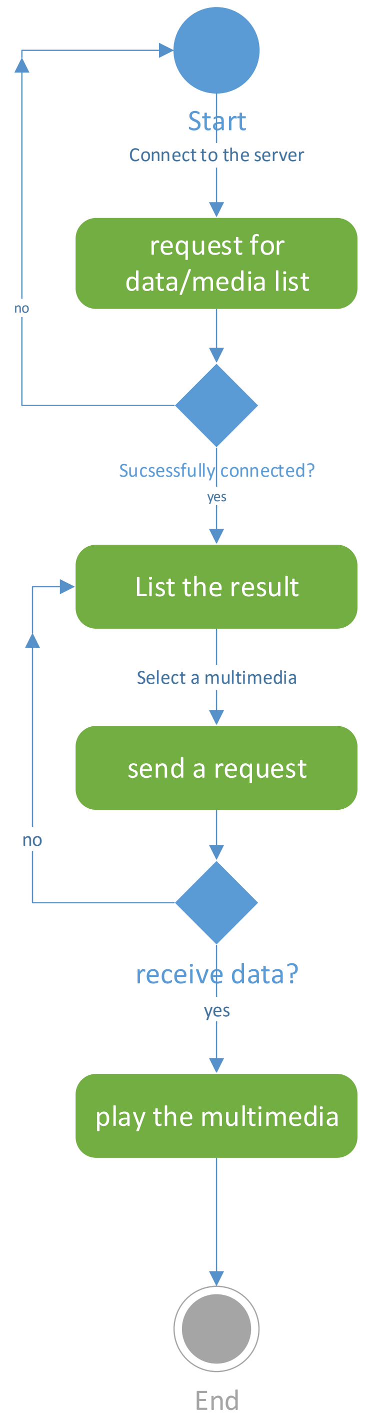


FIGURE 9 – CLIENT SIDE ACTIVITY ANALYSIS

4.2.2. Activity modeling (Server side)

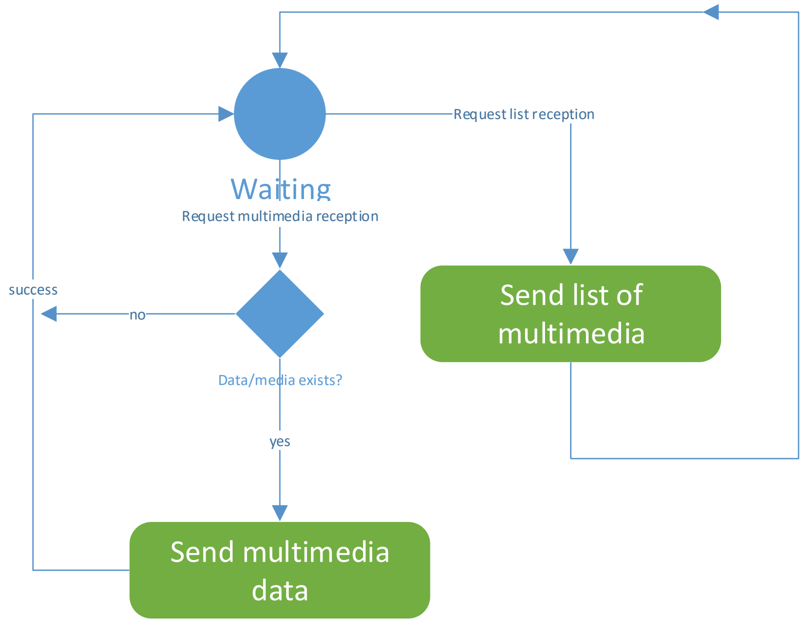


FIGURE 10 - SERVER SIDE ACTIVITY ANALYSIS

4.3. Architecture Design

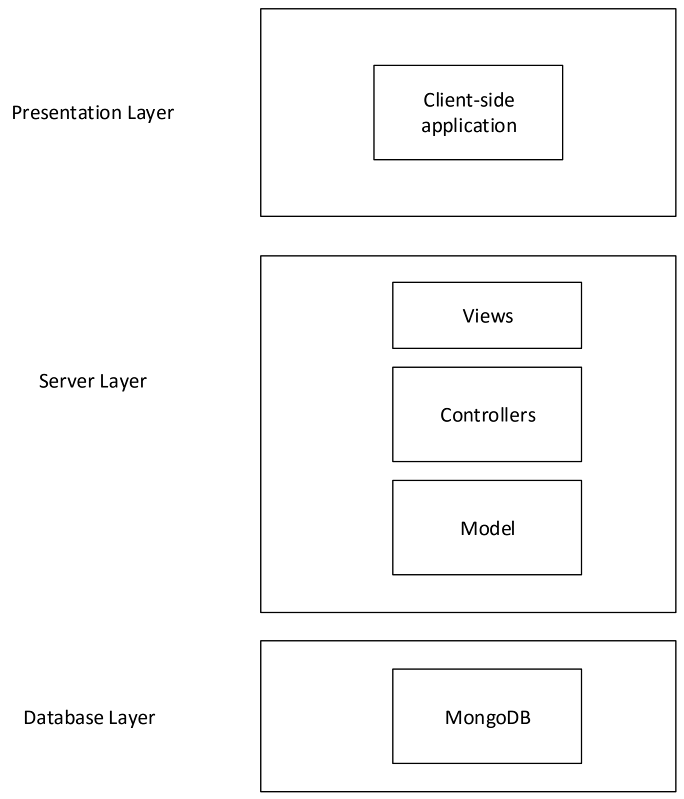


Figure 11 - First Abstract Of Architecture Design

- **Summary:** the entire architecture is divided into 3 layers as is shown above.
- **Presentation layer:** the presentation layer represents the client-side applications such as browsers, mobile apps and so on. It's mainly the user interface for rendering the result of streaming data.
- **Server layer:** the core of back-end processing program. It is principally modularized from MVC (Model-View-Controller) pattern.
- **Database layer:** it stores all the necessary information (user data, media data, etc.) for the server layer.

4.4. API/Router Design

We need to format the server-side requests (API) and the client-side requests (UI Router).

4.4.1. Server-side – API

For Audio module:

URI	Method	Result
/api/audios	GET	Retrieve audios' list
	POST	Create/Upload an audio profile
/api/audios/{audioId}	GET	Read an audio's profile
	PUT	Update an audio's profile
	DELETE	Delete an audio's profile
/api/audios/stream/{audioId}	GET	Stream an audio file to a player

For Video module:

URI	Method	Result
/api/videos	GET	Retrieve videos' list
	POST	Create/Upload a video profile
/api/videos/{videoId}	GET	Read a video's profile
	PUT	Update a video's profile
	DELETE	Delete a video's profile
/api/videos/stream/{videoId}	GET	Stream a video file to a player

4.4.2. Client-side – UI Router

For Audio module:

URI	Result
/audios	Show the audios' list
/audios/upload	Show audios' uploader UI
/audios/{audioId}	Play a specific audio and show its information
/audios/{audioId}/update	Show a specific audio's updater UI

For Video module:

URI	Result
/videos	Show the videos' list
/videos/upload	Show videos' uploader UI
/videos/{videoId}	Play a specific video and show its information
/videos/{videoId}/update	Show a specific video's updater UI

4.5. Database Design

As for the database tables, we refined the table structure with users as below:

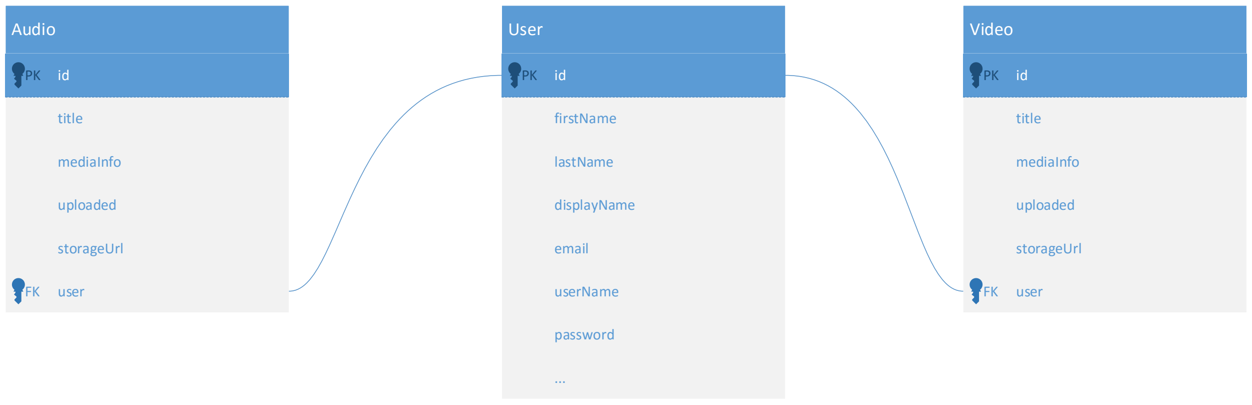


FIGURE 12 - DATABASE TABLE STRUCTURE

4.5.1. User table

The User table contains all the necessary information like username and password of a user and is identified by a unique id.

4.5.2. Audio/Video table

The Audio/Video table represents the Audio/Video data structure identified by a unique id:

- **Title:** is usually the track name of Audio/Video, or simply the filename.
- **MediaInfo:** contains all the media information such as the duration, performer, etc.
- **Uploaded:** represents a timestamp of an upload action.
- **StorageUrl:** represents the storage path of the media (on the server).
- **User:** the owner/uploader of the media, reference to the User table.

4.6. Sequence Diagram

Typically associated with use case realizations, we built the event scenarios with a sequence diagram:

This UML sequence diagram includes 6 parts of the actors and the modules:

- **User:** the primary actor who interacts with the system
- **Controller objects:** derived from the abstract controller, the main entrance of the system, to whom the user directly communicate
- **View objects:** derived from the abstract view, the main output presentation of the system
- **Model objects:** derived from the abstract model, in charge of the data structures
- **Database:** the information data storage
- **Media file storage:** where the media files are storage

Part I. Login, media presentations

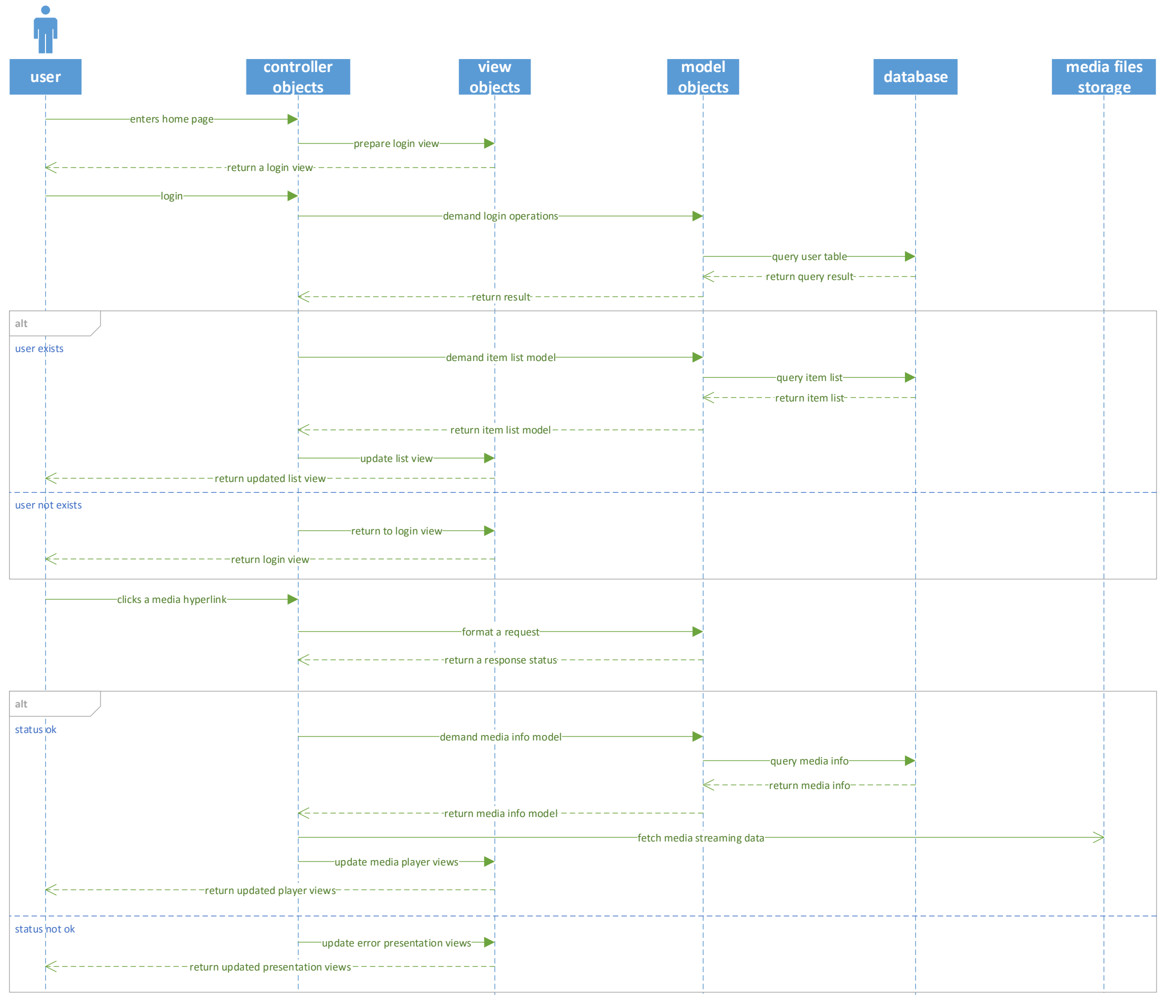


FIGURE 13 - SEQUENCE DIAGRAM (PART I)

- **Introduction:** this part of the sequence diagram represents 2 operations: user login and media requests.
- **User login:** all users are required to login at the very first time they enter the system. Once the login operation begins, the corresponding controller object will demand the view to prepare the login interface and

the corresponding view will be updated and returned to the user. Then after the user submits all necessary authentication to the controller, the controller invokes a login request from the model that will query the user data from the database and then responds with appropriate result to make the controller update the view to the user.

- **Media demand:** usually a user will click a hyperlink to demand (playing) a specific media file, and the controller formats a request and pass it to the model. The model checks if the file exists or not, then gives the corresponding response to the user.

Part II. The upload operation

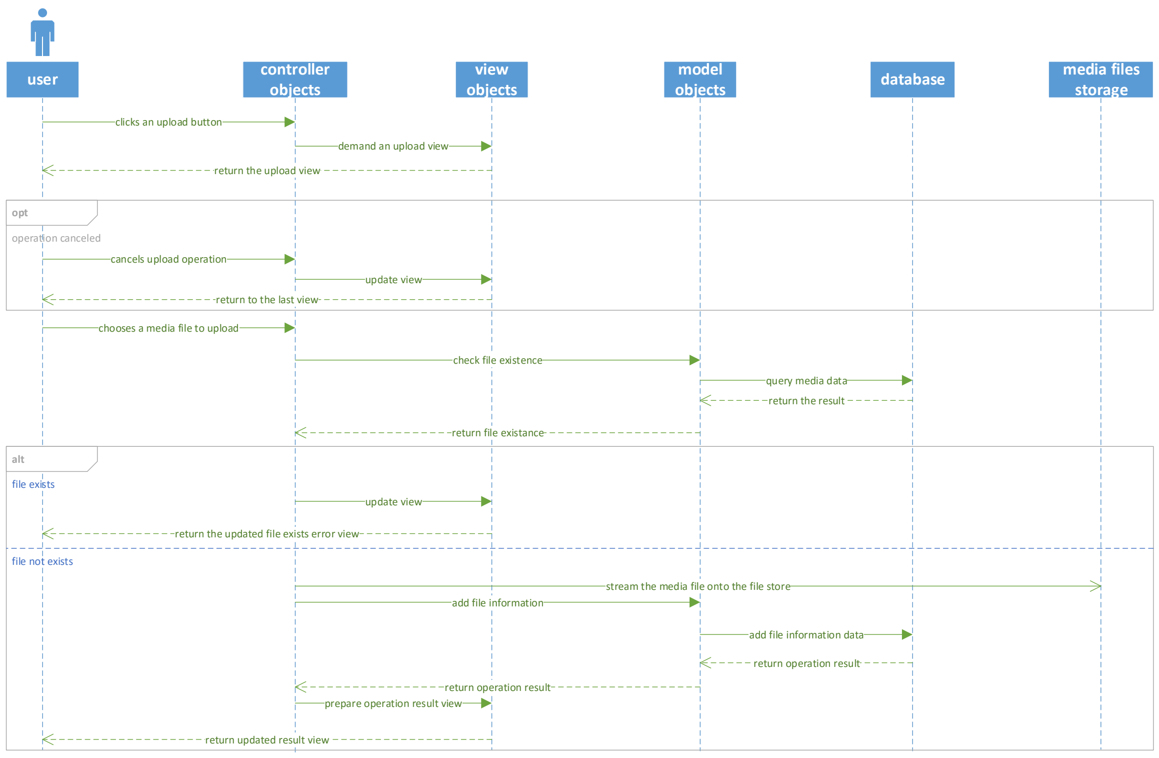


Figure 14 - Sequence Diagram (Part II)

- **Introduction:** this part of the UML sequence diagram represents the upload operation.
- **Upload operation:** when a user clicks the “upload” button, the controller demands an upload UI from the view, as usual, the view returns an updated view to user directly. Of course the user can optionally cancel the operation, otherwise when the user selects a media file to upload, the controller send a file existence check request to the model, and if the media file selected doesn’t exists, the controller will stream it onto the sever, meanwhile the controller will add the relevant information to the model and the database. Finally, the user will get an updated view indicating the success of the operation.

4.7. Class Diagram

The class diagram represents the main building block of object oriented modeling (mainly the MVC modeling in our case). It describes the structure of our application by showing its classes, their attributes, operations (or methods) and the relationships among objects.

4.7.1. The MVC modeling

First of all, we modified our MVC structure as we reconsidered it in a different way by further comprehension:

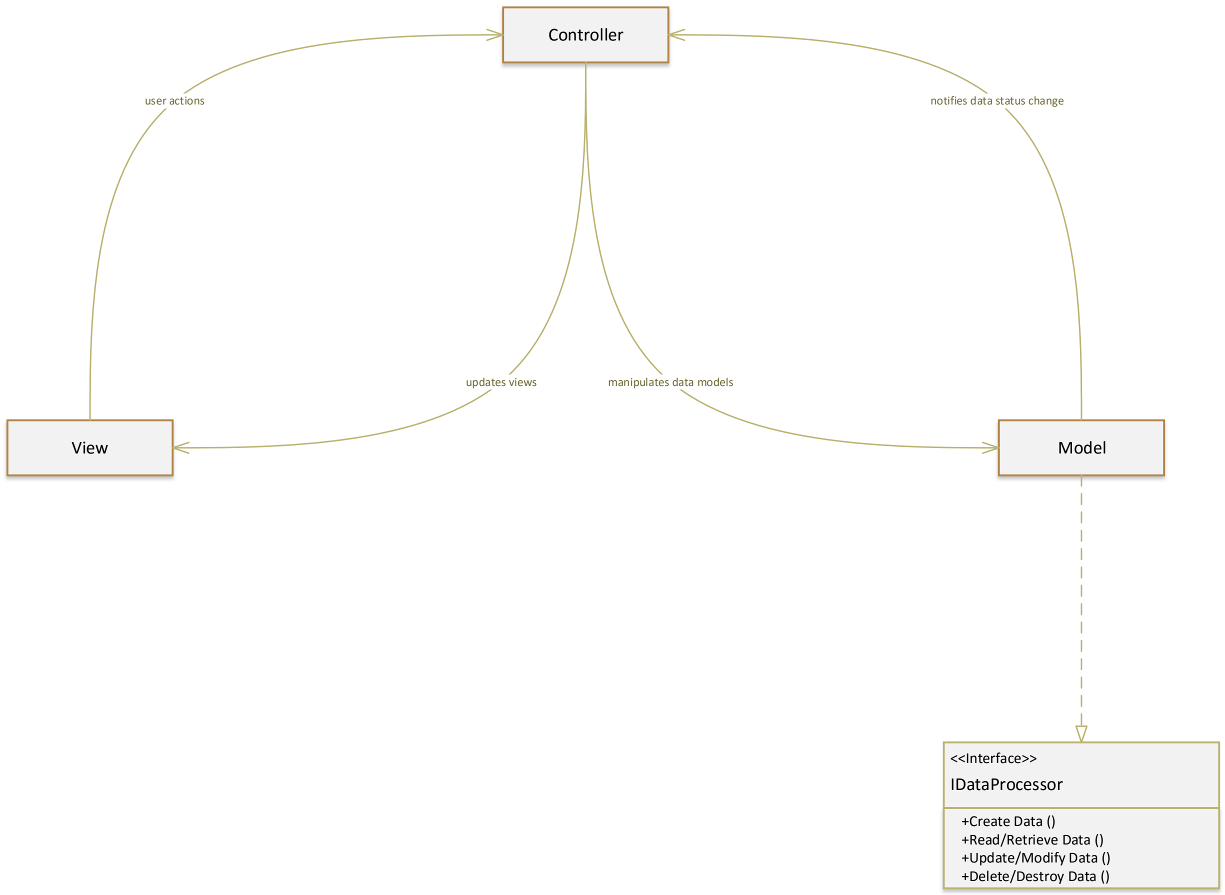


FIGURE 15 - MVC MODELING (MODIFIED)

- **Model:** the models are where the application's data objects are stored, they don't know anything about views and controllers. When a model changes, typically it will notify its observers that a change has occurred.
- **View:** the views is what's presented to the users and how users interact with our application, they are made with HTML, CSS, JavaScript and often templates. for example, in our case, we can create a view that nicely presents the list of audio/video items to our users. Users can also enter an/a audio/video item through some clicks. However, a view should not know how to update the model because that's the controllers' job.
- **Controller:** the controllers are the decision makers and the glues between the model and view, they update the views when the model changes, it also adds event listeners to the views and updates the model when the user manipulates the view. In our application, when the user uploads an item, the upload action is forwarded to the controllers, the controller modifies the model to mark a new item as added.

4.7.2. Models

We begin with the model part – the model objects.

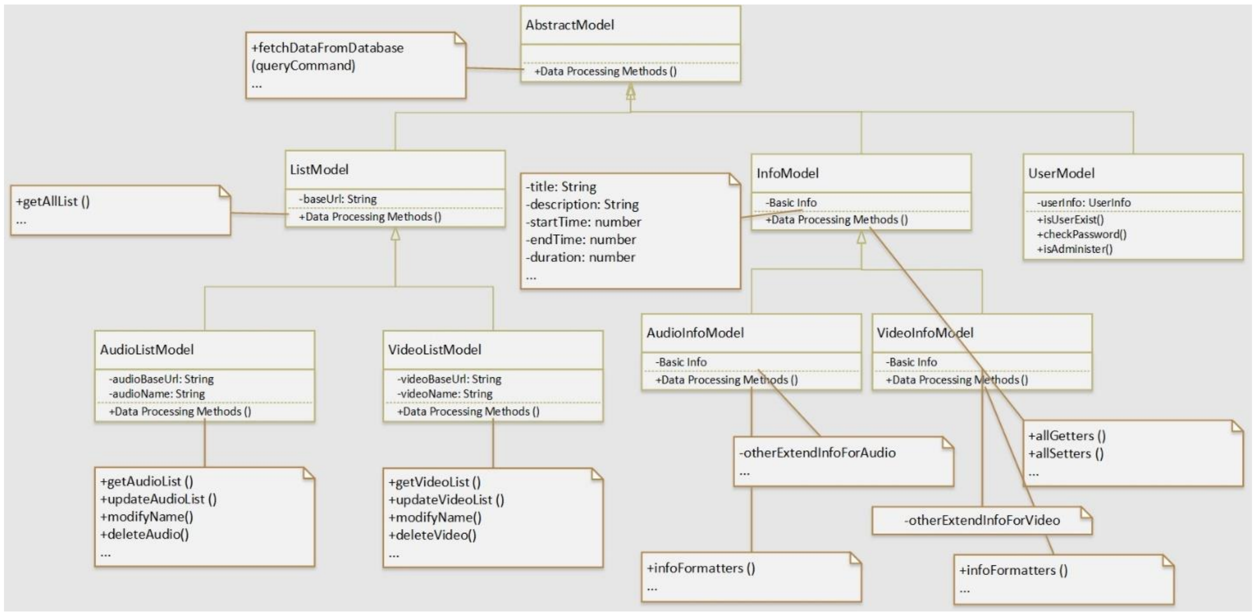


Figure 16 - UML Class Diagram Of The Model Part

We divided data models into three types: the list and the information of audios/videos, user data. All the attributes and the methods are not completely defined but we have figured out the majority of them as above.

- **ListModel**: we added the actions to modify or delete the multimedia
- **InfoModel**: storage and operations of all media information
- **UserModel**: this class contain the methods of the users, for example, to check if the user account exists or if the password is right

4.7.3. Views

In order to implement a SPA (Single Page Application) like presentation, we conceived the view part with the UML class as below:

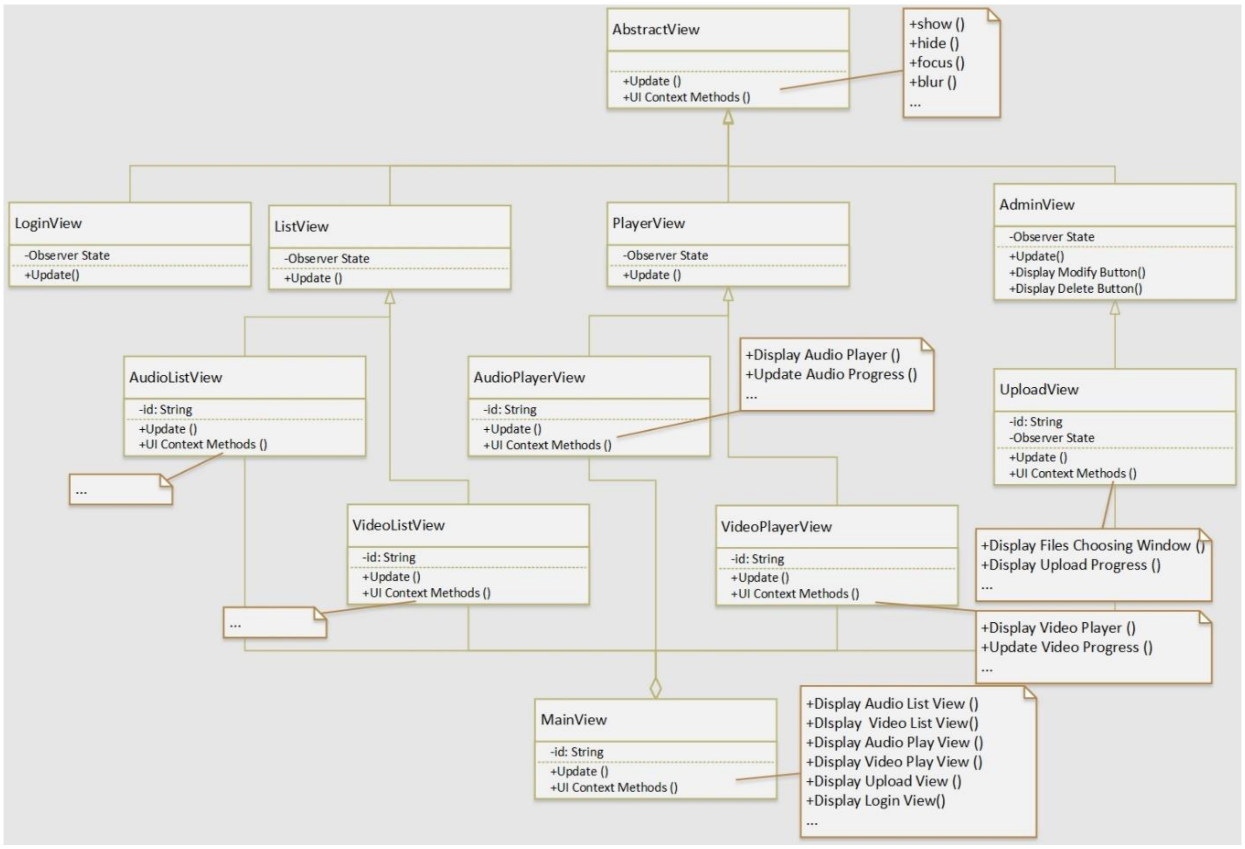


FIGURE 17 - UML CLASS DIAGRAM OF THE VIEW PART

- **LoginView**: to display the login form
- **ListView**: represent list UI
- **PlayerView**: represent the player UI
- **AdminView**: to display the button to modify, delete and upload the multimedia

The figures following are the basic interfaces design:

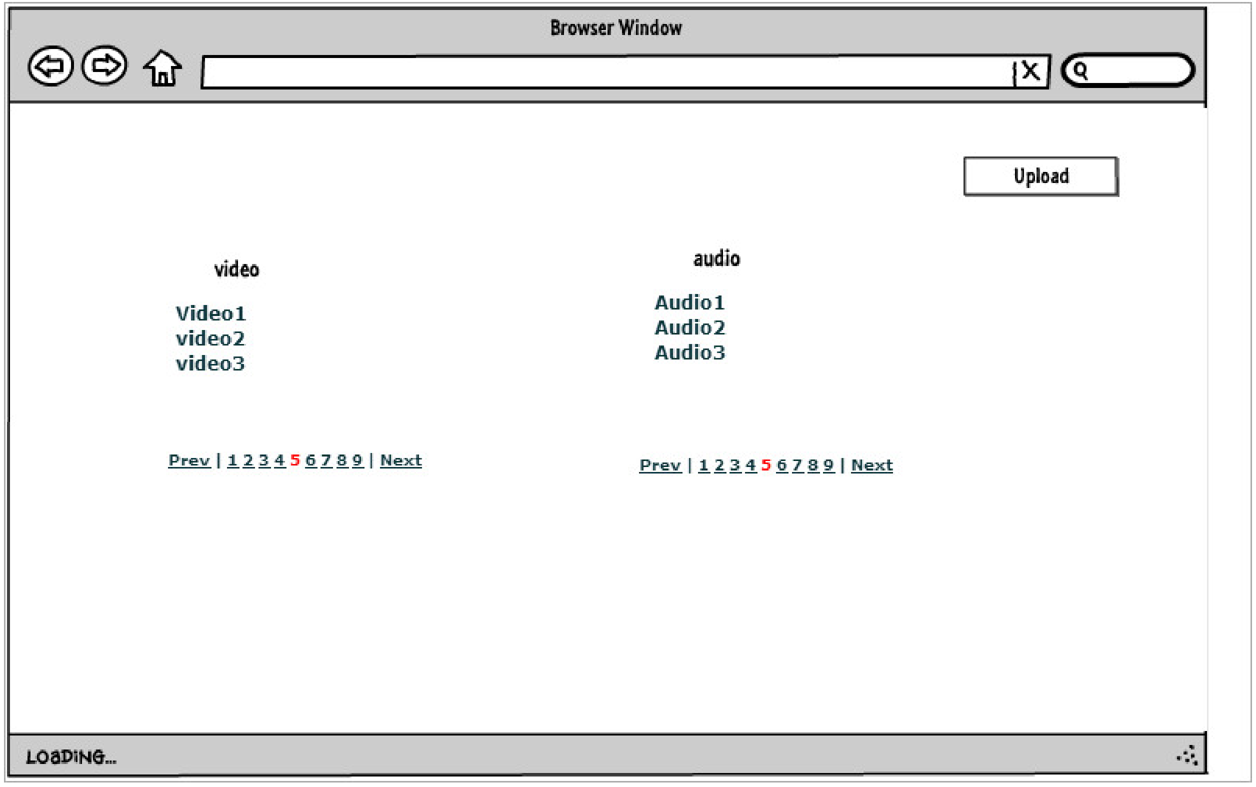


FIGURE 18 - THE MAIN VIEW

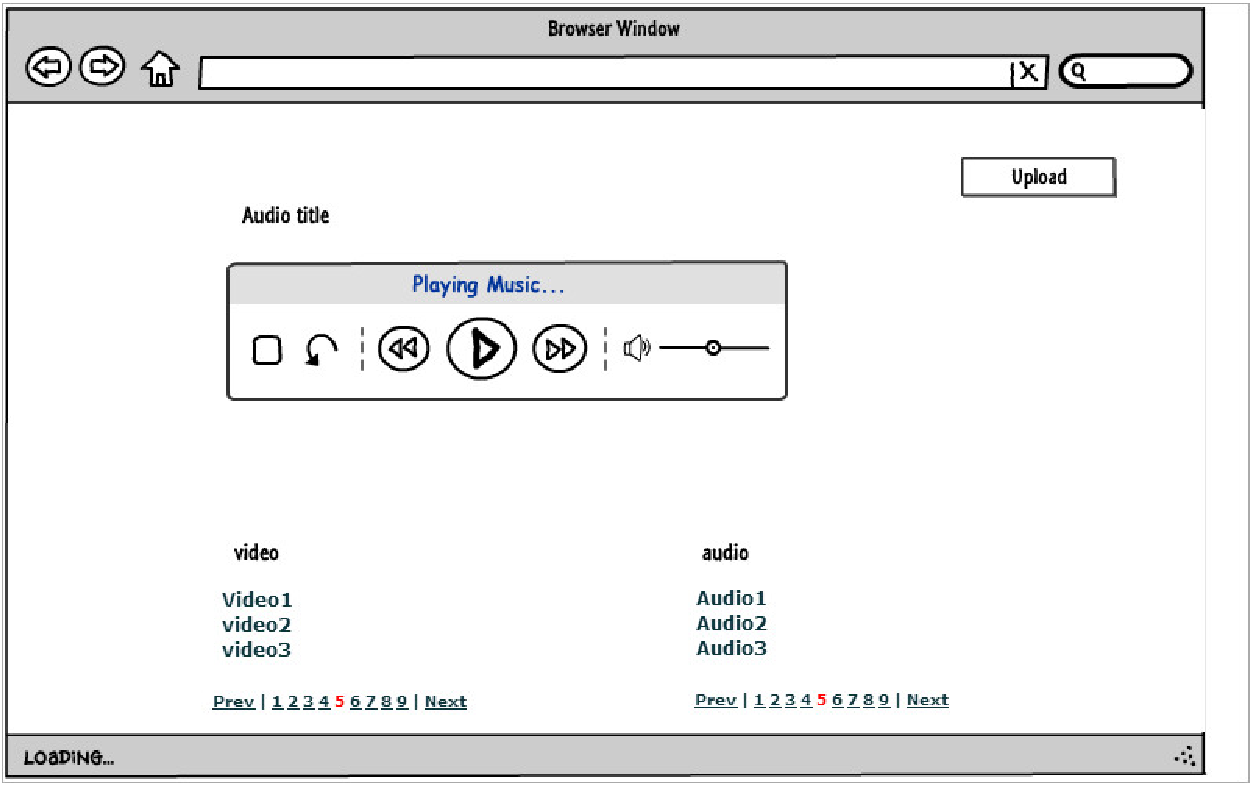


FIGURE 19 - THE AUDIO PLAYER VIEW

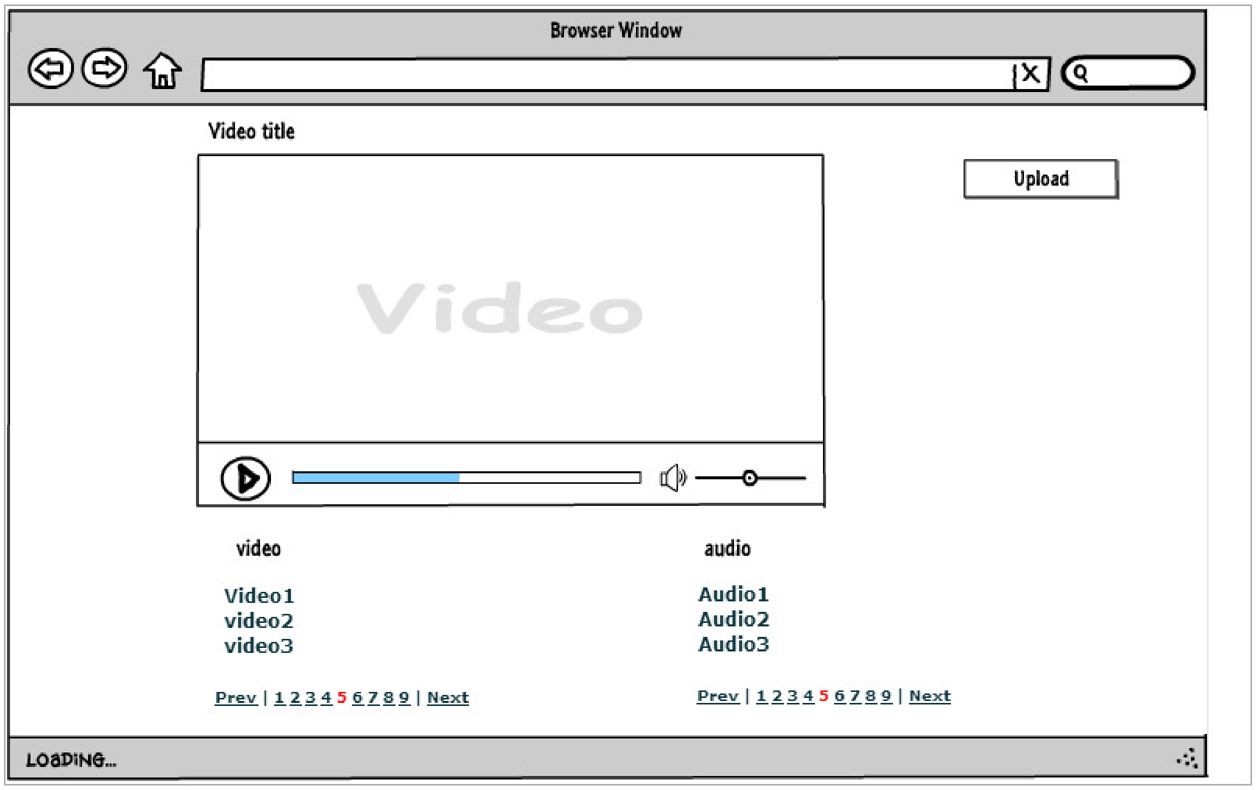


Figure 20 - - The Video Player View

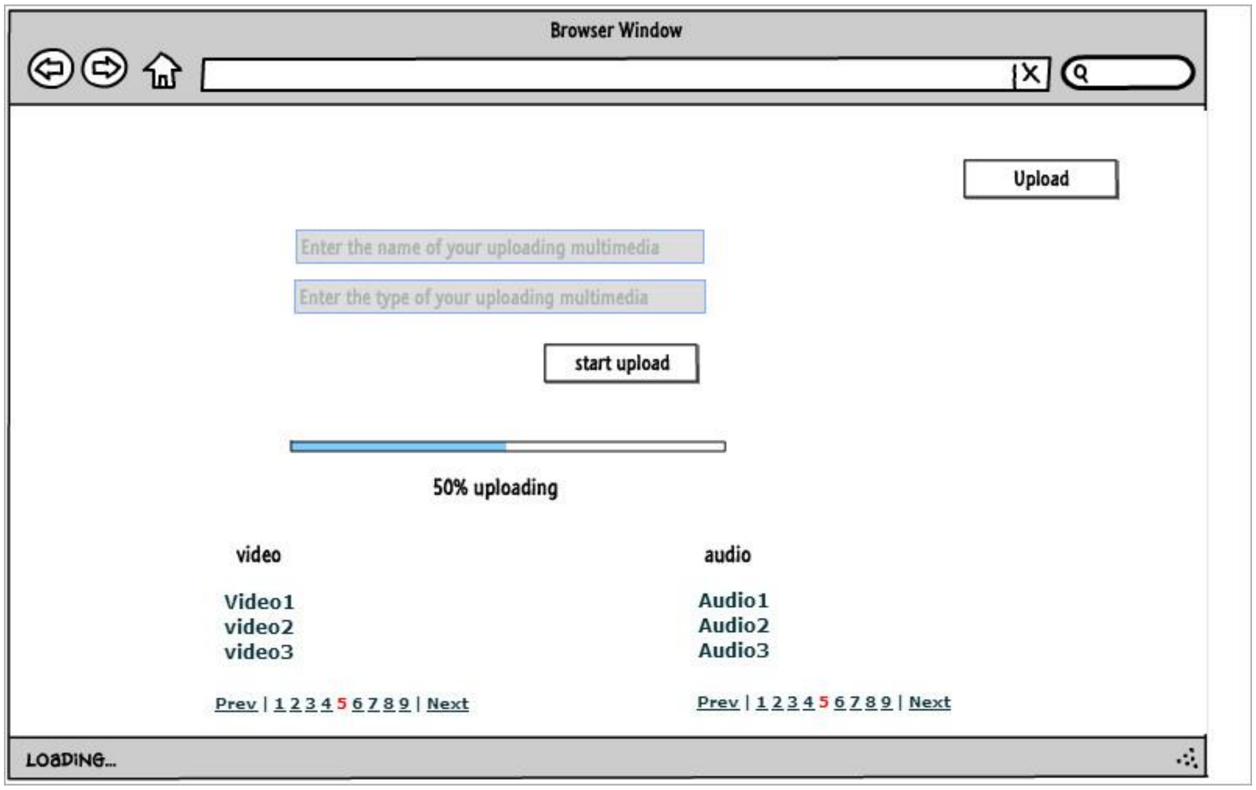

Browser Window
Upload
Enter the name of your uploading multimedia
Enter the type of your uploading multimedia
start upload
50% uploading
video
audio
Video1
video2
video3
Audio1
Audio2
Audio3
Prev | 1 2 3 4 5 6 7 8 9 | Next
Prev | 1 2 3 4 5 6 7 8 9 | Next
LOADING...

Figure 21 - The Upload View

4.7.4. Controllers

As glues between the views and the models, the controllers play an important role in the MVC pattern. We refined the controller part as below:

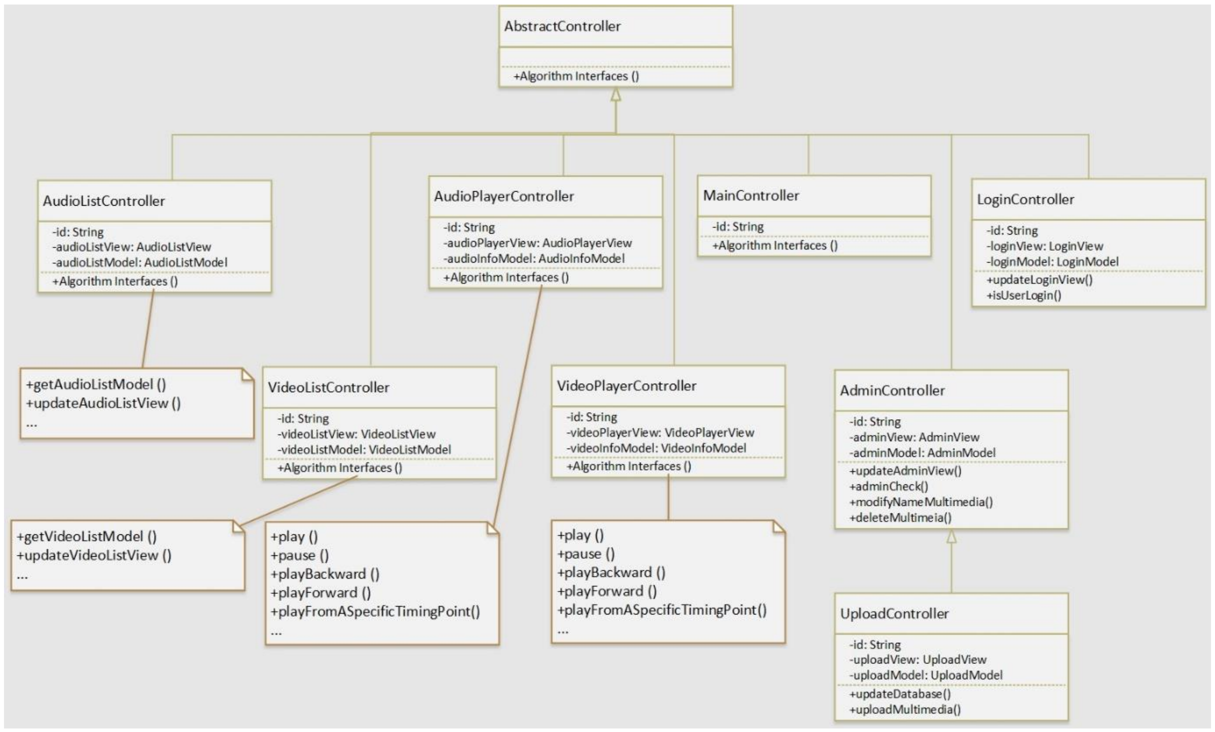


Figure 22 - UML Class Of The Controller Part

To reduce the complexity of development, we attach a controller to each view to handle the associated data model.

- **MainController**: controls the main page
- **AudioListController**: controls the audio lists
- **AudioPlayerController**: controls the audio player
- **VideoListController**: controls the video lists
- **VideoPlayerController**: controls the video player
- **LoginController**: the controller to manage the log in part of the user
- **AdminController**: to manage all the actions of the administer such as modifying the name of multimedia, deleting the files and uploading multimedia.

5. Development with MEAN Stack

After conceiving the structure of our project, we can start developing with MEAN stack

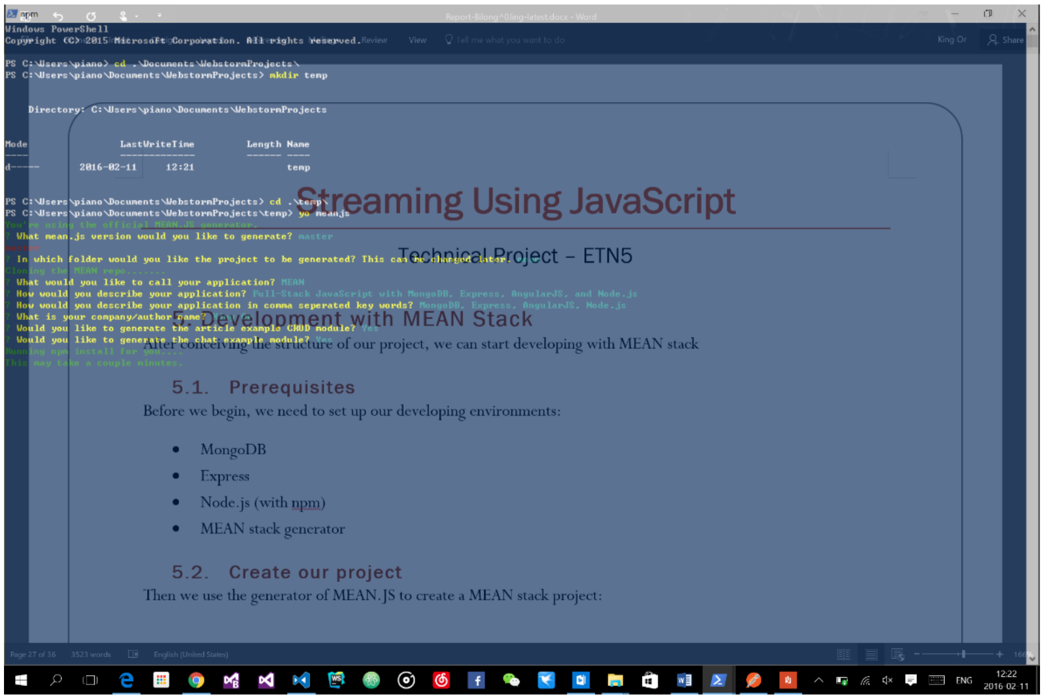
5.1. Prerequisites

Before we begin, we need to set up our developing environments:

- MongoDB
- Express
- Node.js (with npm)
- MEAN stack generator

5.2. Create our project

Then we use the generator of MEAN.JS to create a MEAN stack project:



```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\piano> cd .\Documents\WebstormProjects\
PS C:\Users\piano\Documents\WebstormProjects> mkdir temp


    Directory: C:\Users\piano\Documents\WebstormProjects


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       2016-02-11     12:21                temp


PS C:\Users\piano\Documents\WebstormProjects> cd .\temp\
PS C:\Users\piano\Documents\WebstormProjects\temp> yo meanjs
You're using the official MEAN.JS generator.
? What mean.js version would you like to generate? master
master
? In which folder would you like the project to be generated? This can be changed later.
Cloning the MEAN repo.......
? What would you like to call your application? MEAN
? How would you describe your application? Full-Stack JavaScript with MongoDB, Express, AngularJS, and Node.js
? How would you describe your application in comma seperated key words? MongoDB, Express, AngularJS, Node.js
? What is your company/author name? King Or
? Would you like to generate the article example CRUD module? Yes
? Would you like to generate the chat example module? Yes
Running npm install for you....
This may take a couple minutes.
```

FIGURE 23 - USE THE GENERATOR OF MEAN.JS TO CREATE A PROJECT

5.3. Create Modules

After creating our empty project, we can use the sub-generator of MEAN.JS to generate the <audios> and <videos> modules:

```
yo meanjs:mean-module
```

6. Testing

After working out the use cases and the sequence diagram, we can come of the tests as presented below:

6.1. Database interface

The communications between the "model" and database (the common CRUD).

6.1.1. Unit test

These unit testing cases will be implemented during or after development.

Test ID	Description	Expected Result	User actions	Result
UTDI-1	**Model.create:** Create or add new entries	Successful query	Use PostMan to test the request	Successful query
UTDI-2	**Model.read:** Read, retrieve, search, or view existing entries	Successful query	Use PostMan to test the request	Successful query
UTDI-3	**Model.update:** Update or edit existing entries	Successful query	Use PostMan to test the request	Successful query
UTDI-4	**Model.delete:** Delete/deactivate existing entries	Successful query	Use PostMan to test the request	Successful query

6.2. Login function

All users are required to login at the very first time they enter the system.

6.2.1. Unit tests

These unit testing cases will be implemented during or after development.

Test ID	Description	Expected Result	User actions	Result
UTL-1	**Controller.login:** Demand login procedures	Return user existence checking result	---	Action successful
UTL-2	**Model.checkUser:** Query user table from the database to see if the user exists	Return user existence	---	Action successful
UTL-3	**Controller.requestList:** Request media list from model	Return the corresponding list mode object	---	Action successful
UTL-4	**Model.queryList:** Query media list from database	Read media list from database	---	Action successful

6.2.2. Integration tests

These integration testing cases will be implemented after passing all unit testing cases.

Test ID	Description	Expected Result	User actions	Result
ITL-1	User login	Login successfully or ask the new ones to sign up	User login before entering the main page to view media list	Action successful

6.3. Media demand

Usually a user will click a hyperlink to demand (playing) a specific media file, and the controller formats a request and pass it to the model. The model checks if the file exists or not, then gives the corresponding response to the user.

6.3.1. Unit test

These unit testing cases will be implemented during or after development.

Test ID	Description	Expected Result	User actions	Result
UTMD-1	**Controller.demandMediaInfo:** Controller demands media information object from the corresponding model	Media information object is returned	---	Action successful
UTMD-2	**Model.queryMediaInfo:** Model queries media information from the database	Media information is returned	---	Action successful
UTMD-3	**Controller.retrieveMediaData:** Controller starts fetching the media data and stream it to the view	Stream the media data on a view page	---	Action successful

6.3.2. Integration test

These integration testing cases will be implemented after passing all unit testing cases.

Test ID	Description	Expected Result	User actions	Result
ITMD-1	Stream existing media from an appropriate request	Media required is successfully displayed	User clicks a media link	Action successful

6.4. Upload operation

When a user clicks the "upload" button, the controller demands an upload UI from the view, as usual, the view returns an updated view to user directly. Of course the user can optionally cancel the operation, otherwise when the user selects a media file to upload, the controller send a file existence check request to the model, and if the media file selected doesn't exists, the controller will stream it onto the sever, meanwhile the controller will add the relevant information to the model and the database. Finally, the user will get an updated view indicating the success of the operation.

6.4.1. Unit test

These unit testing cases will be implemented during or after development.

Test ID	Description	Expected Result	User actions	Result
UTU-1	**Controller.checkFile:** Check file existence	File existence is returned	---	Action successful
UTU-2	**Model.queryMediaData:** Query media data from database	Query result is returned	---	Action successful
UTU-3	**Controller.sendMedia:** Stream media file into the file storage	Media file is successfully sent	---	Action successful
UTU-4	**Controller.addMediaInfo:** Pass media information object to model	Media information object is successfully past	---	Action successful
UTU-5	**Model.addMediaInfo:** Retrieve media information model and add the information to database	Media information is successfully added to database	---	Action successful

6.4.2. Integration test

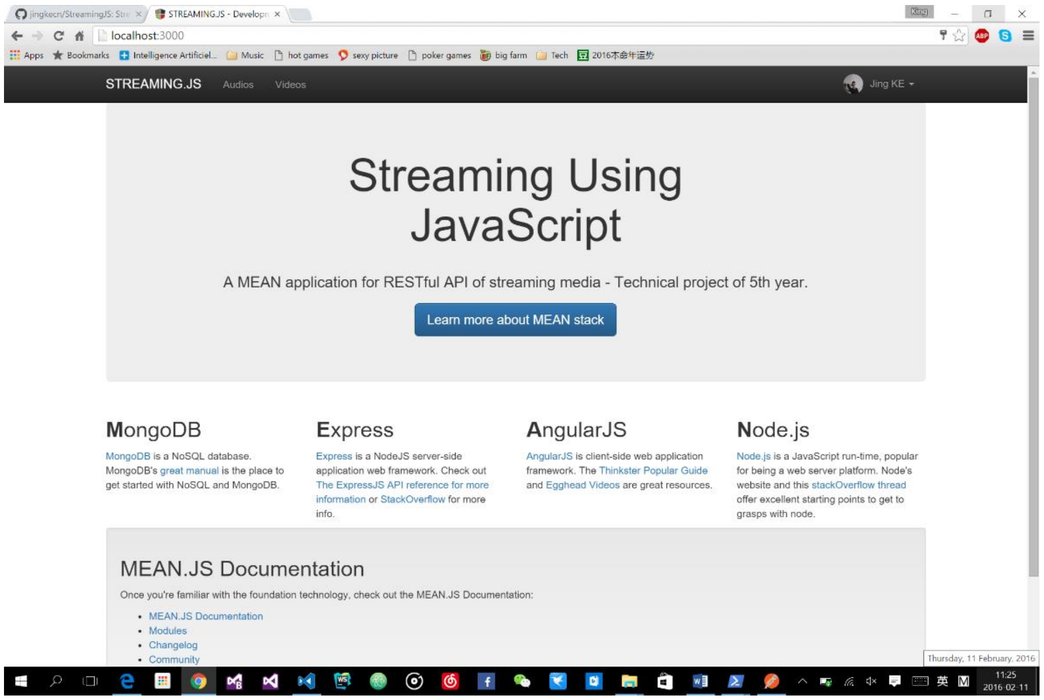
These integration testing cases will be implemented after passing all unit testing cases.

Test ID	Description	Expected Result	User actions	Result
ITU-1	Upload a media file from local storage to the server storage	Upload successfully done	User clicks an upload button and choose a local media file to upload	Action successful

7. Results Presentation

We will present respectively client-side user interface and server-side data operations of our application.

7.1. Client Side



Streaming Using JavaScript

A MEAN application for RESTful API of streaming media - Technical project of 5th year.

MongoDB

MongoDB is a NoSQL database. MongoDB's great manual is the place to get started with NoSQL and MongoDB.

Express

Express is a NodeJS server-side application web framework. Check out The ExpressJS API reference for more information or StackOverflow for more info.

AngularJS

AngularJS is client-side web application framework. The Thinkster Popular Guide and Egghead Videos are great resources.

Node.js

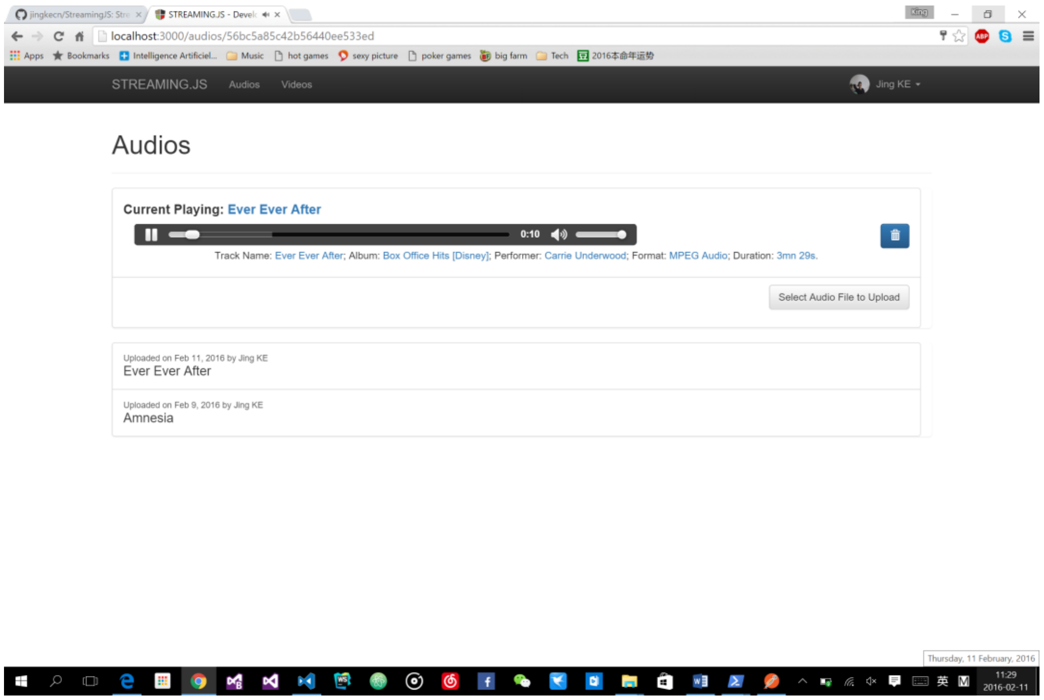
Node.js is a JavaScript run-time, popular for being a web server platform. Node's website and this stackOverflow thread offer excellent starting points to get to grasps with node.

MEAN.JS Documentation

Once you're familiar with the foundation technology, check out the MEAN.JS Documentation:

- MEAN.JS Documentation
- Modules
- Changelog
- Community

FIGURE 24 - HOME PAGE



Audios

Current Playing: Ever Ever After

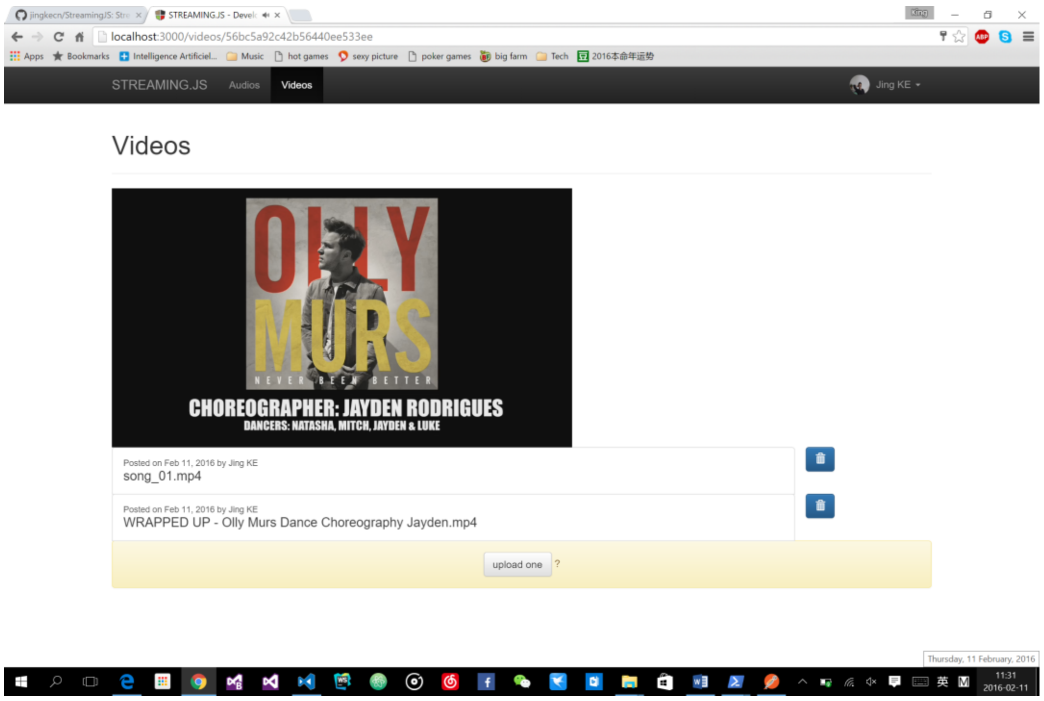
Track Name: Ever Ever After; Album: Box Office Hits [Disney]; Performer: Carrie Underwood; Format: MPEG Audio; Duration: 3mn 29s.

Select Audio File to Upload

Uploaded on Feb 11, 2016 by Jing KE
Ever Ever After

Uploaded on Feb 9, 2016 by Jing KE
Amnesia

FIGURE 25 - AUDIO PAGE


localhost:3000/videos/56bc5a92c42b56440ee533ee
STREAMING.JS
Audios
Videos
Jing KE
OLLY MURS
NEVER BEEN BETTER
CHOREOGRAPHER: JAYDEN RODRIGUES
DANCERS: NATASHA, MITCH, JAYDEN & LUKE

Videos

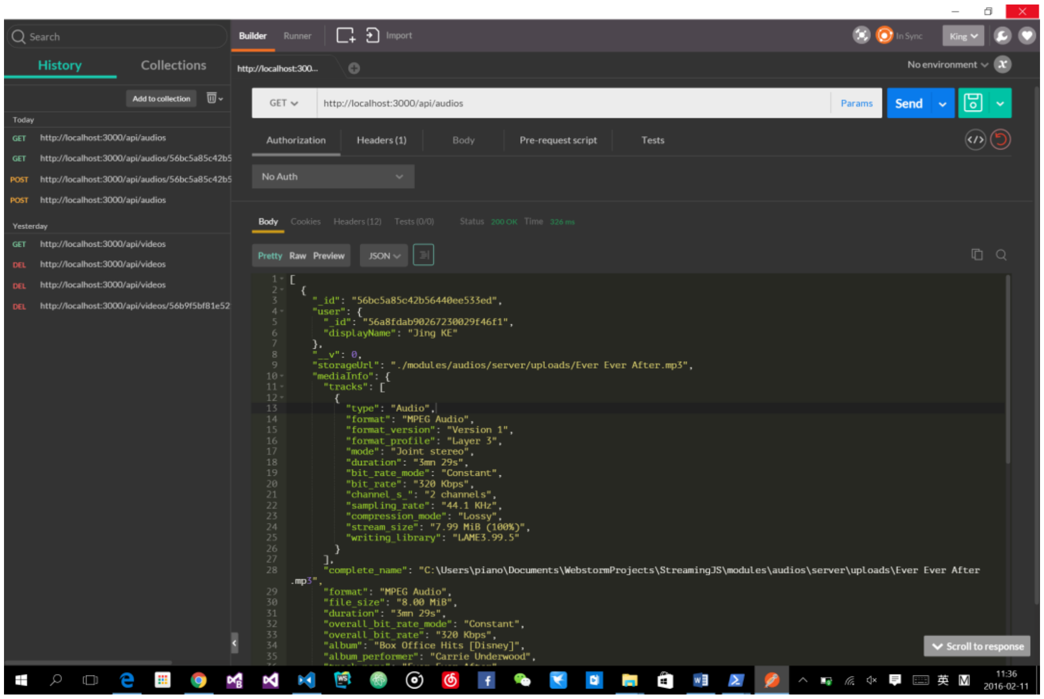
Posted on Feb 11, 2016 by Jing KE
song_01.mp4

Posted on Feb 11, 2016 by Jing KE
WRAPPED UP - Olly Murs Dance Choreography Jayden.mp4

upload one ?

FIGURE 26 - VIDEO PAGE

7.2. Server Side

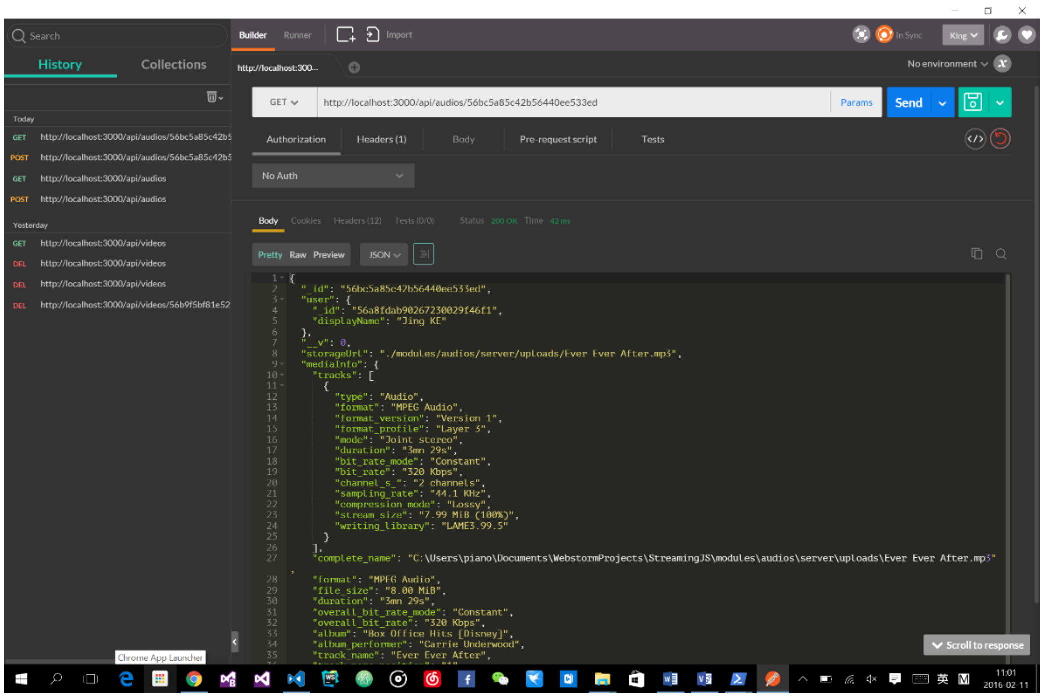


```
GET http://localhost:3000/api/audios
```

Status 200 OK Time 326 ms

```
[
  {
    "_id": "56bc5a85c42b56440ee533ed",
    "user": {
      "_id": "56a8fdab90267230029f46f1",
      "displayName": "Jing KE"
    },
    "__v": 0,
    "storageUrl": "./modules/audios/server/uploads/Ever Ever After.mp3",
    "mediaInfo": {
      "tracks": [
        {
          "type": "Audio",
          "format": "MPEG Audio",
          "format_version": "Version 1",
          "format_profile": "Layer 3",
          "mode": "Joint stereo",
          "duration": "3mn 29s",
          "bit_rate_mode": "Constant",
          "bit_rate": "320 Kbps",
          "channel_s_": "2 channels",
          "sampling_rate": "44.1 KHz",
          "compression_mode": "Lossy",
          "stream_size": "7.99 MiB (100%)",
          "writing_library": "LAME3.99.5"
        }
      ],
      "complete_name": "C:\Users\piano\Documents\WebstormProjects\StreamingJS\modules\audios\server\uploads\Ever Ever After
.mp3",
      "format": "MPEG Audio",
      "file_size": "8.00 MiB",
      "duration": "3mn 29s",
      "overall_bit_rate_mode": "Constant",
      "overall_bit_rate": "320 Kbps",
      "album": "Box Office Hits [Disney]",
      "album_performer": "Carrie Underwood",
```

Figure 27 – Get Media List

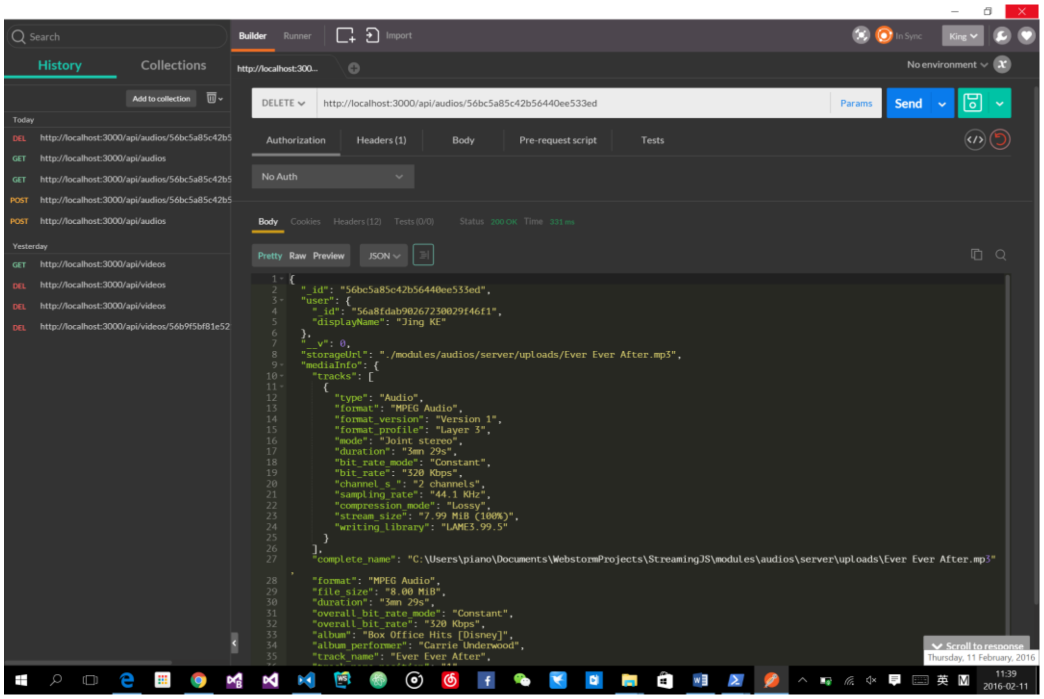


```
GET http://localhost:3000/api/audios/56bc5a85c42b56440ee533ed
```

Status 200 OK Time 42 ms

```
{
  "_id": "56bc5a85c42b56440ee533ed",
  "user": {
    "_id": "56a8fdab90267230029f46f1",
    "displayName": "Jing KE"
  },
  "__v": 0,
  "storageUrl": "./modules/audios/server/uploads/Ever Ever After.mp3",
  "mediaInfo": {
    "tracks": [
      {
        "type": "Audio",
        "format": "MPEG Audio",
        "format_version": "Version 1",
        "format_profile": "Layer 3",
        "mode": "Joint stereo",
        "duration": "3mn 29s",
        "bit_rate_mode": "Constant",
        "bit_rate": "320 Kbps",
        "channel_s_": "2 channels",
        "sampling_rate": "44.1 KHz",
        "compression_mode": "Lossy",
        "stream_size": "7.99 MiB (100%)",
        "writing_library": "LAME3.99.5"
      }
    ],
    "complete_name": "C:\Users\piano\Documents\WebstormProjects\StreamingJS\modules\audios\server\uploads\Ever Ever After.mp3"
,
    "format": "MPEG Audio",
    "file_size": "8.00 MiB",
    "duration": "3mn 29s",
    "overall_bit_rate_mode": "Constant",
    "overall_bit_rate": "320 Kbps",
    "album": "Box Office Hits [Disney]",
    "album_performer": "Carrie Underwood",
    "track_name": "Ever Ever After",
```

Figure 28 - Get Specific Media By Id



Today

```
DEL   http://localhost:3000/api/audios/56bc5a85c42b5
GET   http://localhost:3000/api/audios
GET   http://localhost:3000/api/audios/56bc5a85c42b5
POST  http://localhost:3000/api/audios/56bc5a85c42b5
POST  http://localhost:3000/api/audios
```

Yesterday

```
GET   http://localhost:3000/api/videos
DEL   http://localhost:3000/api/videos
DEL   http://localhost:3000/api/videos
DEL   http://localhost:3000/api/videos/56b9f5bf81e52
```

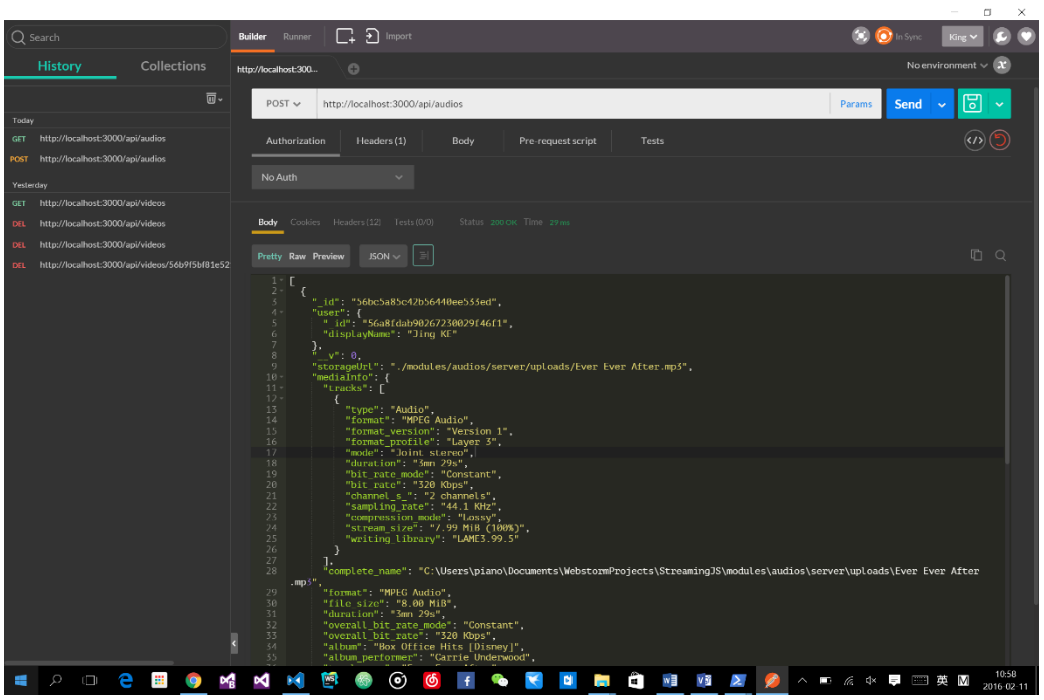
```
DELETE  http://localhost:3000/api/audios/56bc5a85c42b56440ee533ed
```

No Auth

Status 200 OK Time 331 ms

```
{
  "_id": "56bc5a85c42b56440ee533ed",
  "user": {
    "_id": "56a8fdab90267230029f46f1",
    "displayName": "Jing KE"
  },
  "__v": 0,
  "storageUrl": "./modules/audios/server/uploads/Ever Ever After.mp3",
  "mediaInfo": {
    "tracks": [
      {
        "type": "Audio",
        "format": "MPEG Audio",
        "format_version": "Version 1",
        "format_profile": "Layer 3",
        "mode": "Joint stereo",
        "duration": "3mn 29s",
        "bit_rate_mode": "Constant",
        "bit_rate": "320 Kbps",
        "channel_s_": "2 channels",
        "sampling_rate": "44.1 KHz",
        "compression_mode": "Lossy",
        "stream_size": "7.99 MiB (100%)",
        "writing_library": "LAME3.99.5"
      }
    ],
    "complete_name": "C:\Users\piano\Documents\WebstormProjects\StreamingJS\modules\audios\server\uploads\Ever Ever After.mp3"
,
    "format": "MPEG Audio",
    "file_size": "8.00 MiB",
    "duration": "3mn 29s",
    "overall_bit_rate_mode": "Constant",
    "overall_bit_rate": "320 Kbps",
    "album": "Box Office Hits [Disney]",
    "album_performer": "Carrie Underwood",
    "track_name": "Ever Ever After",
```

FIGURE 29 - DELETE A SPECIFIC MEDIA BY ID



Today

```
GET   http://localhost:3000/api/audios
POST  http://localhost:3000/api/audios
```

Yesterday

```
GET   http://localhost:3000/api/videos
DEL   http://localhost:3000/api/videos
DEL   http://localhost:3000/api/videos
DEL   http://localhost:3000/api/videos/56b9f5bf81e52
```

```
POST  http://localhost:3000/api/audios
```

No Auth

Status 200 OK Time 29 ms

```
[
  {
    "_id": "56bc5a85c42b56440ee533ed",
    "user": {
      "_id": "56a8fdab90267230029f46f1",
      "displayName": "Jing KE"
    },
    "__v": 0,
    "storageUrl": "./modules/audios/server/uploads/Ever Ever After.mp3",
    "mediaInfo": {
      "tracks": [
        {
          "type": "Audio",
          "format": "MPEG Audio",
          "format_version": "Version 1",
          "format_profile": "Layer 3",
          "mode": "Joint stereo",
          "duration": "3mn 29s",
          "bit_rate_mode": "Constant",
          "bit_rate": "320 Kbps",
          "channel_s_": "2 channels",
          "sampling_rate": "44.1 KHz",
          "compression_mode": "Lossy",
          "stream_size": "7.99 MiB (100%)",
          "writing_library": "LAME3.99.5"
        }
      ],
      "complete_name": "C:\Users\piano\Documents\WebstormProjects\StreamingJS\modules\audios\server\uploads\Ever Ever After
.mp3",
      "format": "MPEG Audio",
      "file_size": "8.00 MiB",
      "duration": "3mn 29s",
      "overall_bit_rate_mode": "Constant",
      "overall_bit_rate": "320 Kbps",
      "album": "Box Office Hits [Disney]",
      "album_performer": "Carrie Underwood",
```

FIGURE 30 - CREATE A MEDIA DATA RECORD

8. Conclusion

Dating back to our original objective, we can tell that we have basically accomplished our goal – streaming media using JavaScript. Under the guidance of our tutor, Mr. RICORDEL, we implemented our project step by step by going through the UML methodology.

But there is still something for us to improve. In spite of what we have done so far, it is necessary to transplant our client-side application to as many devices as possible. Thus, I propose that in the future we could use Cordova to encapsulate the client-side application to the mobile devices including Android, iOS and Windows.

9. References

- Doug Rosenberg & Matt Stephens – Use Case Driven Object Modeling with UML (Theory and Practice)
- Fernando Monteiro – Learning Single-page Web Application Development
- MongoDB documentation – https://docs.mongodb.org/manual/?_ga=1.214296785.798237404.1455181328
- Express documentation – http://expressjs.com/en/4x/api.html
- AngularJS documentation – https://docs.angularjs.org/api
- Node.js documentation – https://nodejs.org/en/docs/