Assignment 2
Design for Reuse

Node Upload-Api

Jonas Tornfors, jt222ii
Lucas Wik, lw222gz
Kristoffer Svensson, ks222rt

Linnéuniversity, Kalmar
Using Components and Apis, 1DV436

Outline of the design:

The goal of this assignment and this rapport is to understand how to use and develop an API and then develop an API to upload files. You should be able to upload files to different collections and share these collections with others using a special key and you should be able to download that file.

Assumptions:

- An artifact can be contained in many different collections
  - If an artifact is deleted, it is removed from all collections
  - If an artifact is updated, it is updated in all collections

- An artifact is contained at least in one collection

- A Collection can contain both artifacts and other collections

- Collections are not public. Rather, they can be accessed only by using a special “key”, which is automatically generated when the collection is created
  - Key must be unique.
  - Given a key, you can access all subordinates elements (both artifacts and collections) but not its superiors.

Operations:

- **On artifacts:**
  - Create (upload) an artifact
  - Read (download) an artifact
  - Update (replace) an artifact
  - Delete an artifact

- **On collections:**
  - Create a collection (generate the key identifying it)
  - Read the list of elements contained in the collection
  - Add an element (artifact or collection) to the collection
  - Remove an element (artifact or collection) from the collection
  - Delete a collection
    - In case a given collection contains other collections, the latter will be
    - deleted recursively. Make sure the above assumptions always hold.

Use-cases (XHR = XMLHttpRequest)

Collections

Create:

1. User specifies the collection name in the input field.
2. User press the submit button.
3. The client script sends a XHR to the server with the collection name.
4. The node server grabs the request.
5. Creates a collection.
6. Sends back the collection with an special key which is given when creating a collection.
7. The collection is sent back like a json object.

Delete:

1. The user chooses the collection to remove.
2. Grabs the special key and send it to the server with a XHR.
3. The node server receives the key.
4. The node server checks if it exists.
5. If it exists, removes the collection and its subcollection.
6. If it doesn't exists, throw an error.

Read:

1. The user's searches for an collection on the special key.
2. The client send the special key with a XHR to the server.
3. The node server checks if the collections exists.
4. If it exists, send back the collection as an JSON object to the client.
5. If it doesn't exists, send back an error.

Add subcollections:

1. The user specifies the new collections name in the input field.
2. Then specifies which collection is the “owner” of the subcollection with the speciel key.
3. Send these 2 parameters with a XHR to the server.
4. If it exists, create subcollection and add it on “owner” collection and send back subcollection.
5. If it doesn't exists, send back an error.

Artifacts

Create:

1. The user specifies which collection/collections to add the artifact.
2. The user choose which file to upload.
3. The client script sends an XHR to the server with the file and collection/collections.
4. The node server retrieves the request with the file and collection/collections.
5. The server saves the artifact in the specified folder on the server.
6. The server then adds the artifact to the specified collection/collections.

Remove:

1. The user takes the artifact id.
2. Sends it to the server with a XHR.
3. The node server retrieves the request.
4. Deletes the specified file from the saved folder.
5. Then removes it from all collections.

Read:

1. The user gets a collection with its artifacts.
2. The artifact is shown as a link on the client side.
3. The user clicks on the link to retrieve the artifact.
4. The link sends the file id to the server.
5. The server retrieves the file id and starts to download the file to the user.

Update:

1. The user gets grabs the specified artifact ID.
2. Sends that and the new file (with an upload) to the server with a XHR.
3. The server retrieves the file and ID.
4. Then the server rewrites the new file with the old file based on the ID.

So we are going to use node to store and handle the collections and artifacts. We’ll use javascript to handle the functions and store the collections as JSON object and the artifacts as the file they are. We'll send back JSON objects to the client side so based on the application and view you can do whatever you want with the object and just display specific things from that object.

Then on the client side we are just going to use a simple view of the functions on the page and one javascript to call the server with XMLHttpRequest.

Major design issues

First of all we had to make some research about APIs and how to create them since we had no idea about it. We had some clues how it's working since we've used both APIs and frameworks before, but we found some major issues we had to think about and resolve before we began to code.

1. **Learn more about APIs**
   - How to write the code as easy as possible to let the user have a friendly approach to our API.
   - How to store everything, locally, cloud or on a server.
2. **Choose which language we'll use**
   - We are used to C#, javascript and PHP so we want to use any of them.
   - Have to decide which is the best solution for this task and how to bind it together with the part with storing everything.
3. **How to upload files with that language**
   - We have some knowledge about how to upload and store files but not on every language.
   - Decide where we want to store files.
4. **How to store collections**
   - Don´t know which solutions is the best with collections. If its to create folders and put every file in the specific folder.
   - Or if we will use json objects.
5. **How to bind a collection with subcollections and artifacts**
   - How to bind all collections to each other and all artifacts. If we store them in the collections or we bind them with id´s like in a database.

Now when we are done with the code we can look back on the major issues we had from the beginning and reflect about the issues we had while coding the API. The issues mentioned above explains almost the issues we had while coding as well. You always get problems even if you think you know how to solve the problem. In this case we had 3 major issues while coding.

1. **Choose which language**
   - We decided to go for javascript and with node as the server since we are very familiar to javascript and node is based on javascript. Only thing we had to learn here was how to use express and all modules, how to write a get and a post function. When we got those two functions working and understanding how they worked we could go further on and keep work on the API.
2. **How to upload files**
   - This was the hardest part of them all. We sat down in 3-4 days figuring out how you could send files with XMLHttpRequests down to the server and keep all the data on the way. So we find the solution with using multer on the server, a node module for storing files and on the client side we used

FormData which stores files from an input field in a form into the formData object and then you send that object down to the server.

3. **How to store collections and bind everything together**
   - This part was kinda tricky. We started of with learning how to create folders and to store folders in folders with code, but we changed our minds 1 day later. So our solution is to store the collection as a JSON object. In this object you got a special key for that collection and references with special keys to subcollections and artifacts. So a collection has 2 arrays for storing its subcollection special keys and artifacts ID. We found this a lot easier to handle when using node and javascript, rather than using folders.

Design details

Software architecture

As mentioned above have we been using node as the server application and javascript to handle every call to the server. Node is using modules to make the performance better and if you need a specific function or feature that node contains you just install that part. Our API contains four scripts or components which are used by the user.

Server.js

Server.js is the script which is provided to the user. The user will call server.js with POST and GET requests in form of XMLHttpRequests. With a GET/POST and url the user will send the parameters needed for the server to respond and send back JSON object for the client to handle.
Server.js is providing these GET/POST functions to the user:

1. **get("/collections/:id");**
   This is the search call. The user will pass forward the ID for a collection with the xhr. This function will fetch the collection and control if its subcollection and artifacts exists before sending back the collection to the user. If the collection of the ID doesn´t exists it will send back an error.

2. **get("/download/:id/:name");**
   This is the get function to download a file. The user must provide this function both the ID and the name of the file within the xhr to retrieve the right file to download. If the function gets the right parameters it will download the file for the user.

3. **post("/collections/delete");**
   This is the post function to delete a collection or a subcollection. The user needs to send the special key for the specific collection to delete down to the server. If the collection exists it will be deleted and its sub collections and artifacts. If it doesn´t exists an error will be thrown.

4. **post("/collection/deleteartifact");**
   This post function will delete an artifact from where its saved and to all collections who owns it. The user passes the artifact id with a xhr down to the server. If it´s an artifact and it´s exists it will be removed otherwise an error will be thrown.

5. **post("/collection");**
   This post function will create a new collection. It will either create a new main collection if the parameters only contains a name of the collection. If the parameters contains both a name and a collection special key then the collection will be created as a subcollection in the specified collection. If the special key isn´t a existing collection an error will be thrown. These parameters will be passed down to the server with a xhr.

6. **post('/collection/addartifact');**
   This post function will get two parameters from the xhr that the user used. First parameter is the file itself. It will save the file to the artifact folder and then create a JS object. Second parameter is either one collection ID or several ID´s based on how many collections the user wants to store the artifact in. The function will then put the artifact object in every collection and save them.

7. **post('/collection/updateartifact');**
   This post function will update and replace an artifact. It will save the new artifact over the old one. After it’s saved it will get all collections who owns the old artifact and update them with the new artifact.

Api.js

This script contains all the “ugly” code, or the glue code if you might say so. Server.js just receives the parameters from the client, but it will then call api.js so this script can handle all the functions. These functions are hidden for the user and will only be called by the Server script. Here´s a small information about what they are doing. Some of the functions only exists because javascript is tricky to code so every function is synchronous.

1. **Api.ArtifactExists:**
   Check if an artifact exists in the artifacts folder. Returns true or false to the callback function.
2. **Api.CollectionExists:**
   Check if a collection exists in the collection folder. Return true or false to the callback function.
3. **Api.CheckCollections:**
   This functions check every collections if the subcollections exists on the server by calling Api.CollectionExists, if they don’t exists it will remove them from the main collection.

4. **Api.CheckArtifacts:**
   This function check every artifact in a collection if they exists by calling Api.ArtifactExists, if they don't exists it will remove them from the collection.
5. **Api.GetCollection:**
   This functions checks if a collection exists and if it exists it will read the file and send back a JSON object of that collection. If it doesn't exists it will throw an error.
6. **Api.SaveCollection:**
   This function save a collection to the collection folder if it is an instance of a collection.
7. **Api.DeleteCollection:**
   If the collection exists, then this function will remove it from the server.
8. **Api.DeleteArtifact:**
   If the artifact exists, then this function will remove it from the server.
9. **Api.DeleteAllSubCollections:**
   This function exists to make the other function as synchronous as possible. It only calls on Api.DeleteSubCollections.
10. **Api.DeleteSubCollections:**
    This function will check every subcollection in every collection, if the subcollection doesn't have any subcollections it will be removed and the owner of it. When every subcollection is removed it will remove the parent of them. To be able to check every subcollection this function will call Api.DeleteAllSubcollection to check that specific collection for more subcollections. If a collections is empty of subcollections or has been removed of all subcollections, then the Api.DeleteMain function will be called to remove the parent.
11. **Api.DeleteMain:**
    This function removes the collection of the ID given as a parameter.
12. **Api.UpdateArtifact:**
    This function will be called when it's time to update an artifact. It will check every collection if it contains the specific artifact. The artifact object contains an array of owners. If the function finds an owner of the artifact it will update the collection with the new artifact.
13. **Api.removeTmpFiles:**
    This function removes all of the files in the tmp folder. The tmp folder is the folder we temporary upload images to before we update an image, just to validate the data before the image is updates.

Collection.js

This script is the object of every collection. When the user creates a new collection, it will create a collection object and parse the object into a JSON object. The collection object has some functions on it which all are also hidden by the user and is only called by the glue code.

1. **Collection.Constructor:**
   When a collection is created the constructor is called and you´ll need to pass some parameters. It takes a name of the collection, an existing array of artifact or an empty array, an existing array of collections or an empty array, and a ID. If the collection is new it only needs a name, the collection and artifact array will be created as empty and the ID will be generated by GenerateUID. If the collection constructor is called by Collection.Build then it will have 4 parameters thats allready set.

2. **Collection.AddCollection:**
   This function adds a subcollection to the collection array.
3. **Collection.AddArtifact:**
   This function adds an artifact to the artifact array.

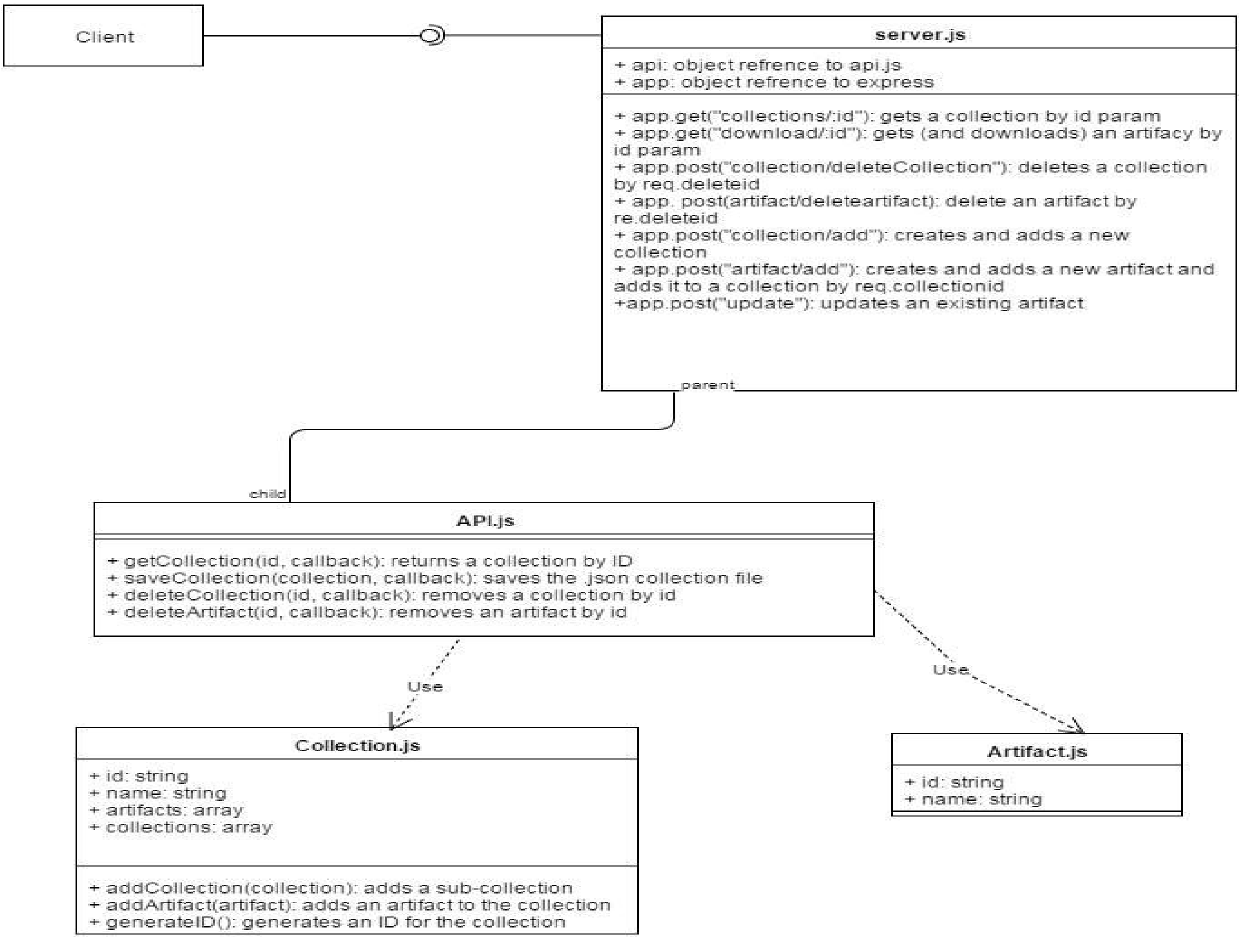
4. **Collection.Build:**
   This function will be called from Api.js when the user wants a collection. The function will be passed a collection and then create a new collection of it to make as synchronous as possible.
5. **GenerateUID:**
   This function will be called by the constructor if the collection object doesn´t have an ID. It will generate a six char long string which contains both numbers and chars.

Artifact.js

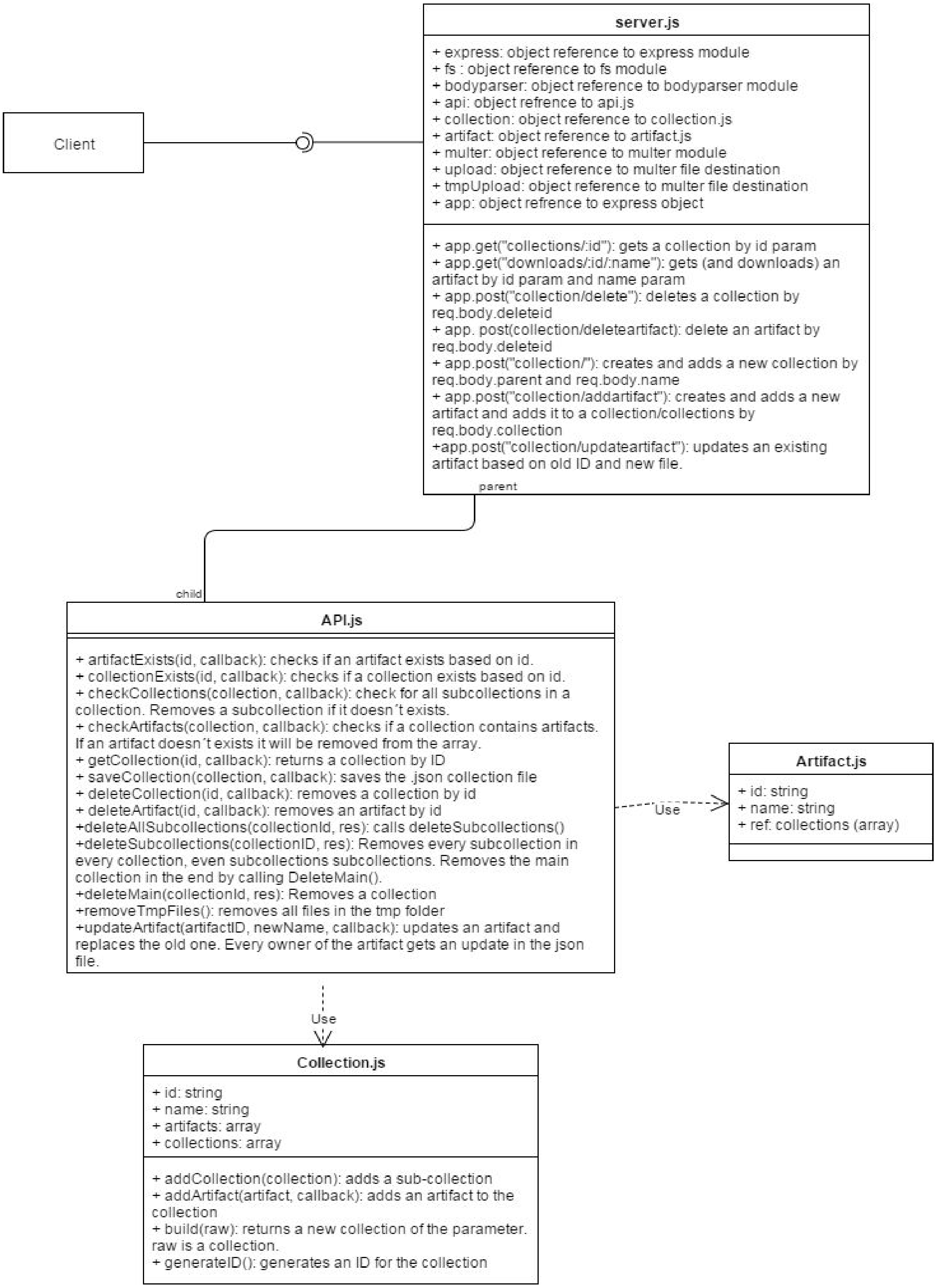
This script is only one function big. It´s the artifact constructor so we can make an object of the file which is uploaded. The artifact object contains a ID of the file which is generated by multer a module to node, a name of the file and an array of all the owners of the artifact, in this case different collections.

Class diagram

This is the class diagram we fought of before coding. It contains the classes we fought about and some functions we knew at that point we would be needed.


Client
server.js
+ api: object refrence to api.js
+ app: object refrence to express
+ app.get("collections/:id"): gets a collection by id param
+ app.get("download/:id"): gets (and downloads) an artifacy by id param
+ app.post("collection/deleteCollection"): deletes a collection by req.deleteid
+ app. post(artifact/deleteartifact): delete an artifact by re.deleteid
+ app.post("collection/add"): creates and adds a new collection
+ app.post("artifact/add"): creates and adds a new artifact and adds it to a collection by req.collectionid
+app.post("update"): updates an existing artifact
parent
child
API.js
+ getCollection(id, callback): returns a collection by ID
+ saveCollection(collection, callback): saves the .json collection file
+ deleteCollection(id, callback): removes a collection by id
+ deleteArtifact(id, callback): removes an artifact by id
Use
Use
Collection.js
+ id: string
+ name: string
+ artifacts: array
+ collections: array
+ addCollection(collection): adds a sub-collection
+ addArtifact(artifact): adds an artifact to the collection
+ generateID(): generates an ID for the collection
Artifact.js
+ id: string
+ name: string

The updated class diagram is a little bit bigger than the class diagram we did at the beginning. It contains every function and object reference.



server.js

+ express: object reference to express module
+ fs : object reference to fs module
+ bodyparser: object reference to bodyparser module
+ api: object refrence to api.js
+ collection: object reference to collection.js
+ artifact: object reference to artifact.js
+ multer: object reference to multer module
+ upload: object reference to multer file destination
+ tmpUpload: object reference to multer file destination
+ app: object refrence to express object

+ app.get("collections/:id"): gets a collection by id param
+ app.get("downloads/:id/:name"): gets (and downloads) an artifact by id param and name param
+ app.post("collection/delete"): deletes a collection by req.body.deleteid
+ app. post(collection/deleteartifact): delete an artifact by req.body.deleteid
+ app.post("collection/"): creates and adds a new collection by req.body.parent and req.body.name
+ app.post("collection/addartifact"): creates and adds a new artifact and adds it to a collection/collections by req.body.collection
+app.post("collection/updateartifact"): updates an existing artifact based on old ID and new file.

API.js

+ artifactExists(id, callback): checks if an artifact exists based on id.
+ collectionExists(id, callback): checks if a collection exists based on id.
+ checkCollections(collection, callback): check for all subcollections in a collection. Removes a subcollection if it doesn´t exists.
+ checkArtifacts(collection, callback): checks if a collection contains artifacts. If an artifact doesn´t exists it will be removed from the array.
+ getCollection(id, callback): returns a collection by ID
+ saveCollection(collection, callback): saves the .json collection file
+ deleteCollection(id, callback): removes a collection by id
+ deleteArtifact(id, callback): removes an artifact by id
+deleteAllSubcollections(collectionId, res): calls deleteSubcollections()
+deleteSubcollections(collectionID, res): Removes every subcollection in every collection, even subcollections subcollections. Removes the main collection in the end by calling DeleteMain().
+deleteMain(collectionId, res): Removes a collection
+removeTmpFiles(): removes all files in the tmp folder
+updateArtifact(artifactID, newName, callback): updates an artifact and replaces the old one. Every owner of the artifact gets an update in the json file.

Artifact.js

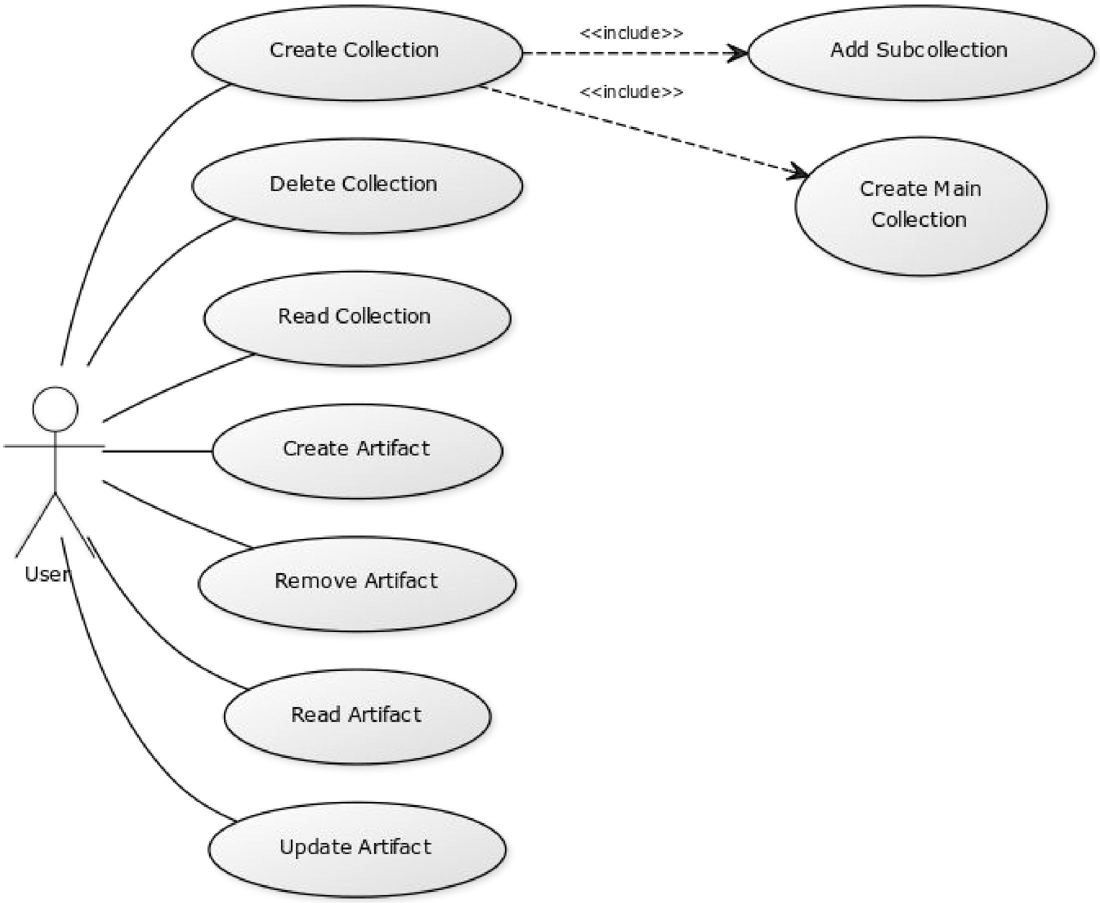
+ id: string
+ name: string
+ ref: collections (array)

Collection.js

+ id: string
+ name: string
+ artifacts: array
+ collections: array

+ addCollection(collection): adds a sub-collection
+ addArtifact(artifact, callback): adds an artifact to the collection
+ build(raw): returns a new collection of the parameter. raw is a collection.
+ generateID(): generates an ID for the collection

UML Use-case diagram



Implementation and API documentation

Installation

Download the API folder, and import the folder to your project. As the API is built you can work with your index and client scripts from the client folder in the API. When you run server.js on the server it will provide the website with every content in the client folder with express.static. The Api will work from your workspace folder which is the API folder. The Only work you need to do is to complement with an index page and a client script to access the server script.

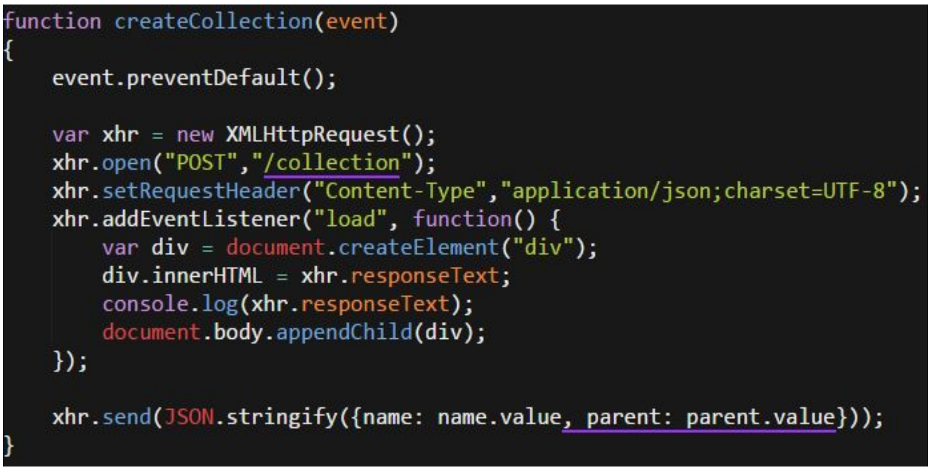
If you'll run the API locally you might need to install Node.js on your computer.
If you'll use a cloud service like C9 there will be no problem because C9 will provide you with everything you need, like node.js.

If you don't specify what port the api should run on it will default to port 3000.

<u>Usage:</u>

Create a collection:

- First you need to instantiate an object of the XMLHttpRequest. We named it XHR.
- Second you open the request with the first parameter as "POST" because we are posting down to the server. Second parameter is which URL or function you want to receive the post, in this case " /collection " will receive the post and create a collection.
- Then you set the RequestHeader.
- Before you send the request you want to make a function thats receive the response from the server. xhr.responsetext will carry the response from the server.
- Last thing to do is to send the object to the server. Very important to send the parameters as a JSON object.
  name = name of the collection.
  parent = parent collection ID if it´s a collection, empty if its a main collection.



```
function createCollection(event)
{
    event.preventDefault();

    var xhr = new XMLHttpRequest();
    xhr.open("POST","/collection");
    xhr.setRequestHeader("Content-Type","application/json;charset=UTF-8");
    xhr.addEventListener("load", function() {
        var div = document.createElement("div");
        div.innerHTML = xhr.responseText;
        console.log(xhr.responseText);
        document.body.appendChild(div);
    });

    xhr.send(JSON.stringify({name: name.value, parent: parent.value}));
}
```

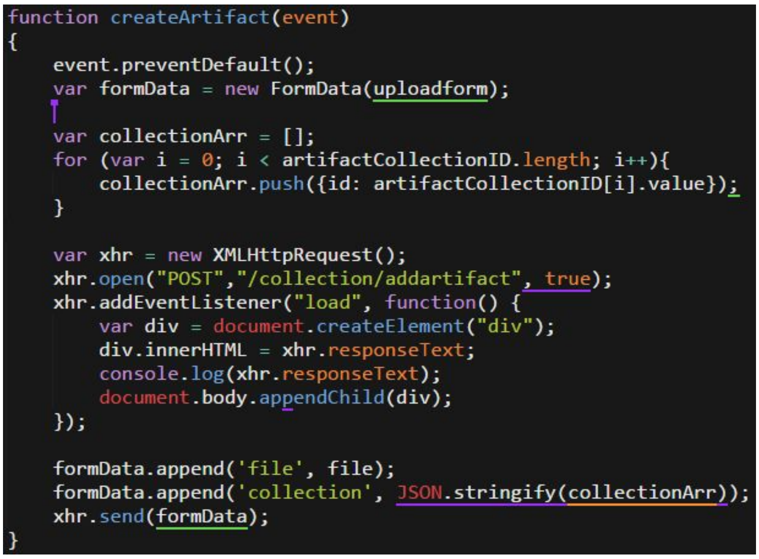
- As a response from the server you will get a json object that should look something like this:
**{
"name":"Collection1",
"artifacts":[{"name":"haaaaaaas.png","id":"65aeb10ecafe30907a102c5b92d65fdc",
"ref":[{"id":"6o9ulv"}]}],"id":"6o9ulv",
"collections":[{"name":"Collection1","id":"v6mbs3"},{"name":"Collection1","id":"u1507c"}]
}**

Create an artifact:

- Before open the XMLHttpRequest you need to handle the file and collection array.
- Start by Creating a formdata object and send in your form from the index site as a parameter, like new FormData(your form from the site).
- Then you handle the collection array from the input fields. Either the array contains one collection id och several ids.
- After these two steps you should instantiate a xhr object.
- Open the request with parameters: POST - /collection/addartifact, true. The last parameters says that it´s an asynchronous call, false will say it´s a synchronous and you should not use a synchronous call.
- Create the function to handle the response.
- Then you need to append the file to the formData object by writing: formData.append(“parameterName”, file) = file is the file you are uploading. formData.append(“parameterName”, collectionArray) = important that you JSON.stringify(collectionArray) first, otherwise the server cant handle it like an array with objects in it.
- Lastly you send the formdata down to the server with xhr.send().

The `FormData` interface provides a way to easily construct a set of key/value pairs representing form fields and their values, which can then be easily sent using the `XMLHttpRequest.send()` method. It uses the same format a form would use if the encoding type were set to `"multipart/form-data"`.

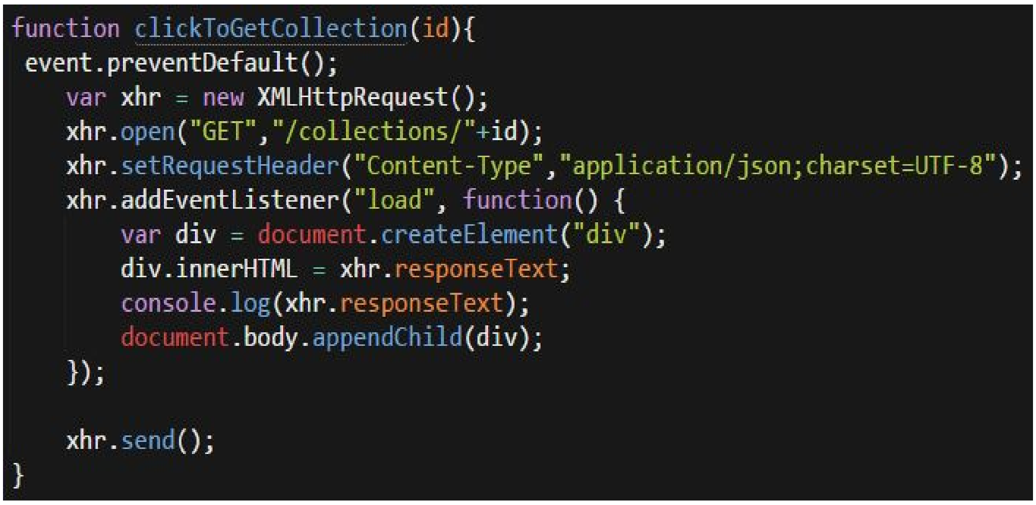
NOTE: It’s important that your form on your site is using, enctype="multipart/form-data".



```
function createArtifact(event)
{
    event.preventDefault();
    var formData = new FormData(uploadform);

    var collectionArr = [];
    for (var i = 0; i < artifactCollectionID.length; i++){
        collectionArr.push({id: artifactCollectionID[i].value});
    }

    var xhr = new XMLHttpRequest();
    xhr.open("POST","/collection/addartifact", true);
    xhr.addEventListener("load", function() {
        var div = document.createElement("div");
        div.innerHTML = xhr.responseText;
        console.log(xhr.responseText);
        document.body.appendChild(div);
    });

    formData.append('file', file);
    formData.append('collection', JSON.stringify(collectionArr));
    xhr.send(formData);
}
```

Read Collections:

- When you search for a collection you'll use this function.
- Instantiate a XHR object.
- Open the request with parameters: GET - /collections/ + id.
  ID is the parameter sent to the function, it´s the collection special Key.
- Set requestHeader.
- Write the function receiving the response and handle it as you want.
- Last thing is to send the request to the server.



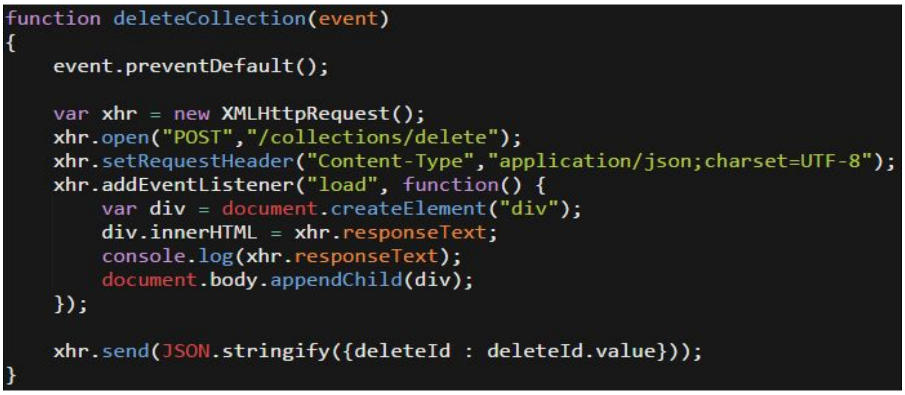
```
function clickToGetCollection(id){
 event.preventDefault();
    var xhr = new XMLHttpRequest();
    xhr.open("GET","/collections/"+id);
    xhr.setRequestHeader("Content-Type","application/json;charset=UTF-8");
    xhr.addEventListener("load", function() {
        var div = document.createElement("div");
        div.innerHTML = xhr.responseText;
        console.log(xhr.responseText);
        document.body.appendChild(div);
    });

    xhr.send();
}
```

- As a response from the server you will get a json object that should look something like this:

**{
"name":"Collection1",
"artifacts":[{"name":"haaaaaaas.png","id":"65aeb10ecafe30907a102c5b92d65fdc",
"ref":[{"id":"6o9ulv"}]}],"id":"6o9ulv",
"collections":[{"name":"Collection1","id":"v6mbs3"},{"name":"Collection1","id":"u1507c"}]
}**

Delete a collection(and it’s subcollections):

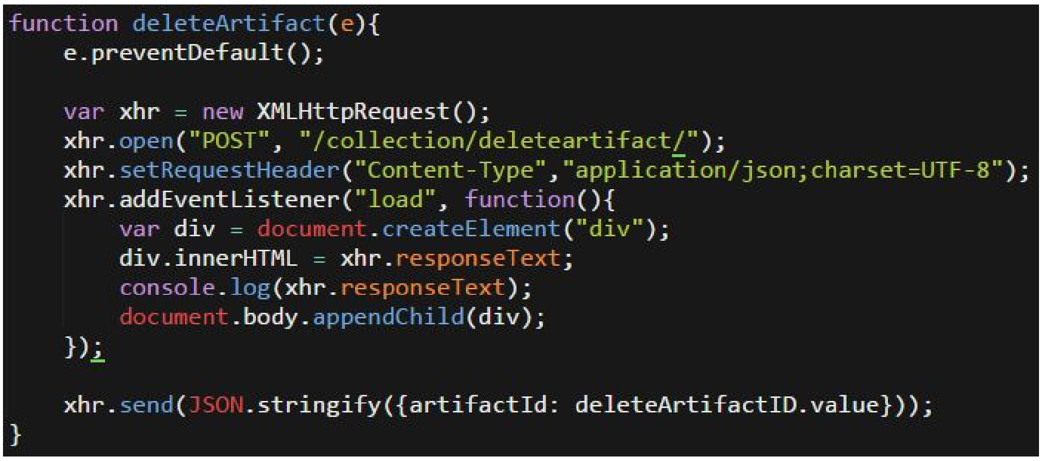
- Instantiate a XHR object.
- Open the request with parameters: POST - /collections/delete
- Set requestHeader
- Write the function receiving the response
- Send a JSON.stringify object with the specified special key id for the collection to the server with xhr.send().



```
function deleteCollection(event)
{
    event.preventDefault();

    var xhr = new XMLHttpRequest();
    xhr.open("POST","/collections/delete");
    xhr.setRequestHeader("Content-Type","application/json;charset=UTF-8");
    xhr.addEventListener("load", function() {
        var div = document.createElement("div");
        div.innerHTML = xhr.responseText;
        console.log(xhr.responseText);
        document.body.appendChild(div);
    });

    xhr.send(JSON.stringify({deleteId : deleteId.value}));
}
```

Delete an artifact:

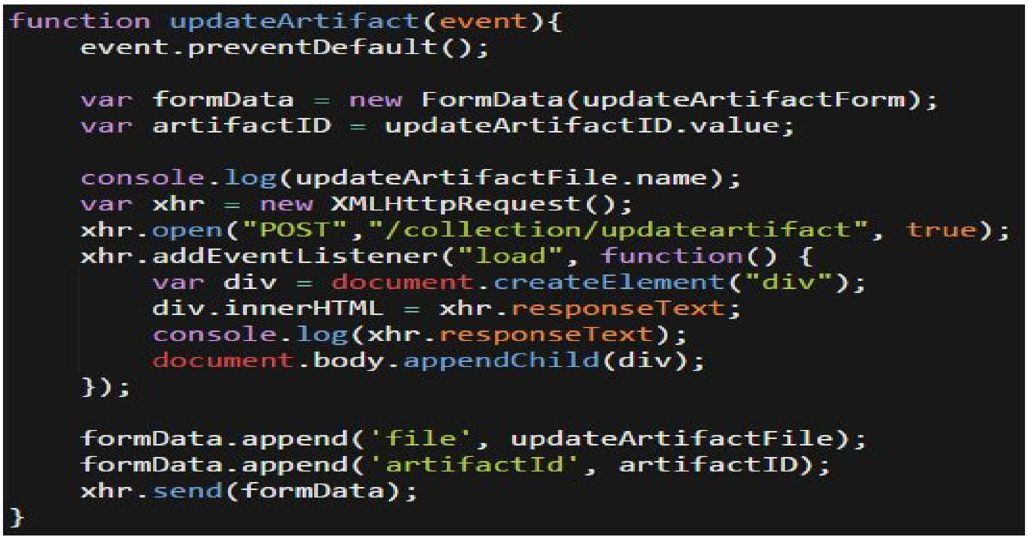
- Instantiate an xhr object.
- Open the request with parameters: POST - /collection/deleteartifact/
- Set requestHeader.
- Handle the response from the server.
- Use xhr.send() and send a json.stringify object with the artifact ID down to the server.



```
function deleteArtifact(e){
    e.preventDefault();

    var xhr = new XMLHttpRequest();
    xhr.open("POST", "/collection/deleteartifact/");
    xhr.setRequestHeader("Content-Type","application/json;charset=UTF-8");
    xhr.addEventListener("load", function(){
        var div = document.createElement("div");
        div.innerHTML = xhr.responseText;
        console.log(xhr.responseText);
        document.body.appendChild(div);
    });

    xhr.send(JSON.stringify({artifactId: deleteArtifactID.value}));
}
```

Update an artifact:

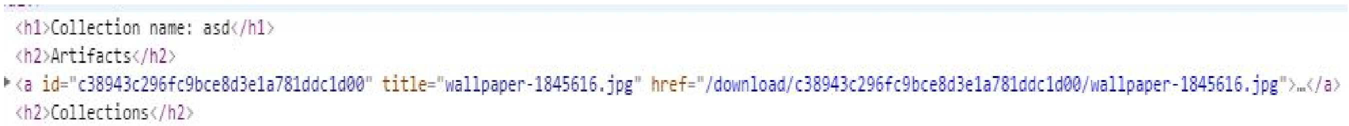
- Instantiate a formData object with the form from the index page.
- Instantiate a variable containing the artifactID from the input field.
- Instantiate a xhr object.
- Open the request with parameters: POST - /collection/updateartifact - true
- Write the response handle function
- formData.append(‘fileNameHere’, file), formData.append(‘oldArtifactIDHere, artifactID)
- xhr.send() down the formData object to the server.



```
function updateArtifact(event){
    event.preventDefault();

    var formData = new FormData(updateArtifactForm);
    var artifactID = updateArtifactID.value;

    console.log(updateArtifactFile.name);
    var xhr = new XMLHttpRequest();
    xhr.open("POST","/collection/updateartifact", true);
    xhr.addEventListener("load", function() {
        var div = document.createElement("div");
        div.innerHTML = xhr.responseText;
        console.log(xhr.responseText);
        document.body.appendChild(div);
    });

    formData.append('file', updateArtifactFile);
    formData.append('artifactId', artifactID);
    xhr.send(formData);
}
```

Download an artifact:

- The easiest way to download a file from the server is to let node to it for you.
- So when you are displaying all items from a collections, you can create an artifact as an a-tag or a link.
- Set the href source to /download/ + the artifactID / + the artifact filename.
- When the user clicks on the link it will call the download function on the server and download the file with the specified ID and rename it for the user with the filename parameter.



```
 <h1>Collection name: asd</h1>
 <h2>Artifacts</h2>
▸<a id="c38943c296fc9bce8d3e1a781ddc1d00" title="wallpaper-1845616.jpg" href="/download/c38943c296fc9bce8d3e1a781ddc1d00/wallpaper-1845616.jpg">…</a>
 <h2>Collections</h2>
```

Timelog and activities

Thursday 17/12, 12:00 - 14:00 - 3 hours

Researched about which language we should use. We went for Node.js and javascript. Started the documentation: outline of the design:

Wednesday 6/1, 14:00 - 17:30 - 3.5 hours

Documentation: Major design issues and class diagrams.

A lot of research about folder, file uploads and so on.

Managed to make a folder by the name from the input folder with mkdirp.

Thursday 7/1, 12:30 - 17:00 - 4.5 hours

Managed to download a file from a folder with the script and a post request.

Friday 8/1, 12:30 - 17:30 - 5 hours

PROGRESS!! We ditched the idea with folders and went for collections and JSON objects instead. Began to create collections and subcollections. Can delete them as well.

Monday 11/1, 12:30 - 17:30 - 5 hours

Began to code for the fileupload function. A lot of research for file uploads.
No solution at the moment.

Tuesday 12/1, 12:00 - 17:00 - 5 hours

Kept researched about fileuploads, no progress so far. Can now delete subcollections as well.

Wednesday 13/1, 12:00 - 17:30 - 5.5 hours

Fileupload done, PROGRESS!!! Finished the functions for deleting collections and subcollections.

Thursday 14/1, 12:00 - 17:30 - 5.5 hours

Made the functions for reading, uploading and deleting an artifact. Delete all subcollections should be bugfree now.

Friday 15/1, 12:00 - 18:00 - 6 hours

Began with the documentation today. A lot of writing and a lot of progress. Almost done with the update artifact function. Removed all bugs so SHOULD be bugfree. And added some validation so the server can´t crash.

Saturday 16/1, 14:00 - 19:00 - 5 hours

Began write API documentation and bug tested our application and found some bugs that had to be resolved but now everything should work.

Sunday 17/1, 14:00 - 16:30 - 2.5 hours

Found one bugg in the code so we solved it and now it should be able to run without crashing. We are done with the documentation and should be able to hand in the code and the rapport.

Date	Activity	Time
Thursday 17/12	Research / Documentation	3 hours
Wednesday 6/1	Research / Documentation / Code	3 hours, 30 minutes
Thursday 7/1	Code	4 hours, 30 minutes
Friday 8/1	Research / Code	5 hours
Monday 11/1	Research / Code	5 hours
Tuesday 12/1	Research / Code	5 hours
Wednesday 13/1	Code	5 hours, 30 minutes
Thursday 14/1	Code	5 hours, 30 minutes
Friday 15/1	Code / Documentation	6 hours
Saturday 16/1	Tests / Documentation	5 hours
Sunday 17/1	Tests / Documentation	2 hours, 30 minutes
		Total: 50 hours, 30 minutes