***Applied Research***

**Tim Verhees**

*20-12-2022   
V1.0*

# **Versions**

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| **02-09-2022** | 0.1 | Initial version |
| **04-11-2022** | 0.5 | Added Hypothesis |
| **20-12-2022** | 1.0 | Further infil, first finished draft |

# **Distribution list**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Name** | **Function** |
| **07-10-2022** | 0.2 | Tim Verhees | Project Handler |
|  |  | Maja | Technical Teacher |
|  |  | Jacco | Technical Teacher |
| **04-11-2022** | 0.5 | Tim Verhees | Project Handler |
|  |  | Maja | Technical Teacher |
|  |  | Jacco | Technical Teacher |
| **20-12-2022** | 1.0 | Tim Verhees | Project Handler |
|  |  | Maja | Technical Teacher |
|  |  | Jacco | Technical Teacher |

# **Table of contents**

[**Versions** 2](#_Toc115786129)

[**Distribution list** 2](#_Toc115786130)

[**Table of contents** 3](#_Toc115786131)

[**Introduction** 4](#_Toc115786132)

[**Method** 4](#_Toc115786133)

[**Results** 4](#_Toc115786134)

[Sub-question 1 4](#_Toc115786135)

[Sub-question 2 4](#_Toc115786136)

[Sub-question 3 4](#_Toc115786137)

[Main-question 4](#_Toc115786138)

[**Conclusion** 5](#_Toc115786139)

[**References** 5](#_Toc115786140)

# **Introduction**

With any projects, problems arise. Questions that need to be answered. The way these questions can be answered is through research. This project is, of course, no different. For this Yu-Gi-Oh ban list application an API will be used to pull some card data, including (but not limited to) the image of the card itself. This comes with some difficulties, seeing as the one who made the API doesn’t appreciate call after call for the image every time the application is used. This is why it will be necessary to save the image.

This however gives rise to some questions, including:

* Main question: How do I save a card image when making an API call, so it doesn’t need to in the future?
  + Sub question: How do you save images from URL’s using software?
  + Sub question: How do you implement a check whether an image already exists?
  + Sub question: How do I ensure that the external API is not overloaded with requests?

Answering these questions will greatly help in implementing the actual images of the cards in the application, which will improve the user experience.

# **Method**

There are many different ways of researching within the IT space. These range from using documentation to find out the best practices to field research by forms of trial and error. Since this problem is still quite alien to me as a developer, a **library** focussed research would probably be the best way of researching this issue.

Now within library focussed research there are several possible methods to apply but the one I chose is **Available product analysis**. This means that I will be looking at existing solutions to the sub-questions and using those to answer the main question at hand.

This means that I will work with documentation found on the internet (mainly **Google**) that help me with resolving the questions. A good example would be that if I were to find an existing application that saves images from URL’s, I could use that to find an answer. Any and all references will be documented in accordance to the APA guidelines.

# **Results**

In this part, I lay out the results of my research. It starts with my hypothesis on the solution before doing any actual research, followed by my findings and conclusions for the questions.

## Sub-question 1: How do you save images from URL’s using software?

Luckily, the API of my choice includes a direct link to the card’s image directly in its results. Meaning that the URL is provided to me with a simple API call. An example of this is for the card Pot of Duality. Using the API call “https://db.ygoprodeck.com/api/v7/cardinfo.php?name=Pot of duality” gives me some JSON data that includes “card\_images”. This is a list that contains “image\_url\_small” which holds a URL that goes directly to the image. So this is how I get the actual URL for the image, but now the question is, how do I use it?

Well after searching the internet I found one solution that works pretty well:

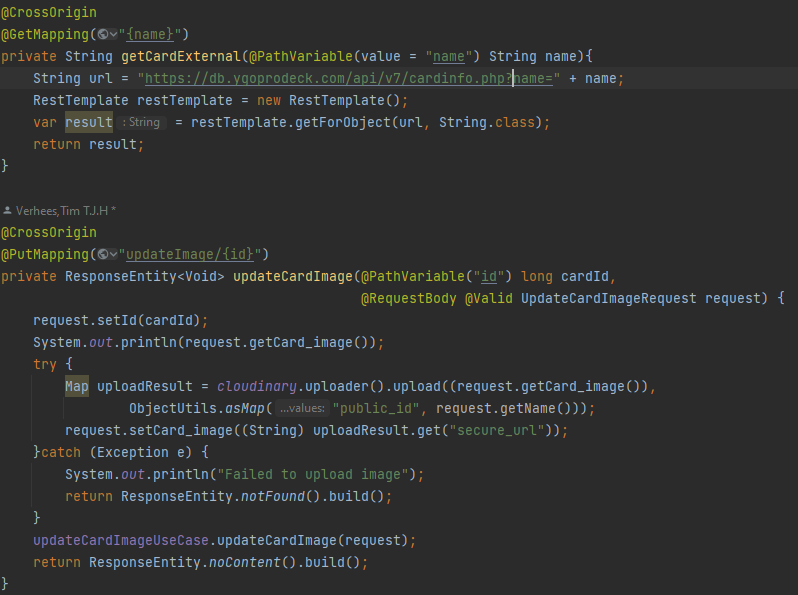
public static void main(String[] args) throws Exception {  
 String imageUrl = "https://images.ygoprodeck.com/images/cards/72302403.jpg";  
 String destinationFile = "C:\\Users\\timve\\Desktop\\Fontys\\Sem3\\Individuele opdracht\\git\\front-end-indiv-app\\src\\images\\Swords of Revealing Light.jpg";  
  
 *saveImage*(imageUrl, destinationFile);  
}  
  
public static void saveImage(String imageUrl, String destinationFile) throws IOException {  
 URL url = new URL(imageUrl);  
 InputStream is = url.openStream();  
 OutputStream os = new FileOutputStream(destinationFile);  
  
 byte[] b = new byte[2048];  
 int length;  
  
 while ((length = is.read(b)) != -1) {  
 os.write(b, 0, length);  
 }  
  
 is.close();  
 os.close();  
}

Let’s dissect this class. We first start by setting up the URL where we’re getting the image from, then we define the path we want the image to use, including the eventual name. Then we use a function we made. This function does several things, using Java’s own libraries to open an input and output stream to take the image from the given URL and save it on the chosen file path. This method will be used to save the card from a URL. Then I will save the file path into the database in order for the front end software to find it at a later time.  
  
Whilst this is a good way to store the files locally, the issue with this is that on the front-end side it fails to load local files dynamically. One way of doing this normally would be using the “require” method. However after many attempts I could not get it to work.

This lead me to a different solution, re hosting the images using an online image hosting service. After looking around for existing solutions I found a website called “Cloudinary”, which would allow me to host the images from the external database.  
  
Now looking over the documentation for Cloudinary I can see two options:

1. Doing it from the backend through a dependency.
2. Doing it from the frontend with an API provided by Cloudinary.

After discussing it with the technical teachers I decided to go with the backend approach and design my own API call for the front end to use. It looks a little something like this.



These API calls do two things; One is that the external card DB is checked in accordance to the card’s name for an existing card, the front end dissects the response and feeds it into the next call where the image from the link is uploaded onto cloudinary and the link returned from that is then saved into the database.

## Sub-question 2: How do you implement a check whether an image already exists?

Seeing as the data for the card’s images has to be stored in the database, my hypothesis on this would be first checking whether that part in the database is filled in for that specific card. If it is, the function for adding an image to the card is halted there. If it isn’t and it is NULL, the function continues and adds the card image to the database.

This one is actually quite simple. Seeing as I save the image the image link in the database with a fall-back for newly created cards. Meaning that the button that is made for requesting only displays if the image found in the database for the card is the fall-back image. This will act as a check on whether it already exists.

## Sub-question 3: How do I ensure that the external API is not overloaded with requests?

Seeing as the creators of the external API have specifically requested that API calls for images are kept at a minimum, this is something I have to consider when building my application. My hypothesis for achieving this would be the solution to the previous sub-question where, it first checks the local API for the card image data before making an API call to the external one. This should then, in theory, prevent multiple API calls on the same card. Disabling the button for a second or two after it is initially pressed should also help with reducing the spamming of said button before the function finishes.  
  
There are two ways I’ve found that can help remedy this issue. One is that once it has successfully added an image the page refreshes, at which point the button will no longer appear on the card details. And the second is that the functionality for requesting an image for a card is an admin only function. This will help counteract abuse of said button.

## Main-question: How do I save a card image when making an API call,

## so it doesn’t need to in the future?

Now onto the main question. It ends up being that when someone makes the API call, it checks the external database for the image URL. Then it uses that URL to save it to my Cloudinary library. This upload generates a response URL which I then save to the card in the database. Now on the front end side the button for requesting will not show up when looking at the cards details.

# **Conclusion**

There were a couple of solutions to this problem that could’ve worked, but the one I’ve found works out for me. Reuploading the image from the external database to my own cloud image server turned out to be it. Now with both Authentication and Authorisation, it will be very hard to cause my website to overload the external API

# **References**

<https://www.avajava.com/tutorials/lessons/how-do-i-save-an-image-from-a-url-to-a-file.html>

<https://console.cloudinary.com/documentation/java_integration>