APPELSCENE PORTFOLIO

Ghyferri Halfhide – ghyferri.halfhide@student.ju.se

Web Development Fundamentals - Jönköping University - 2023

Contents

1) Introduction 1
2) Method 2
2.1) Architecture 2
2.2) Database 3
2.3) Graphical User Interface 5
2.4) Web Application 6
2.4.1 Technology 6
2.4.2 Design Pattern 7
2.4.3 Security 8
2.4.4 Version Control 9
3) Results 10
4) Discussion 11
5) References 11

1) Introduction

Welcome to my report on my project for the subject Web Development Fundamentals. My project is called "Appelscene Portfolio." Now you must be wondering, "What is Appelscene?"

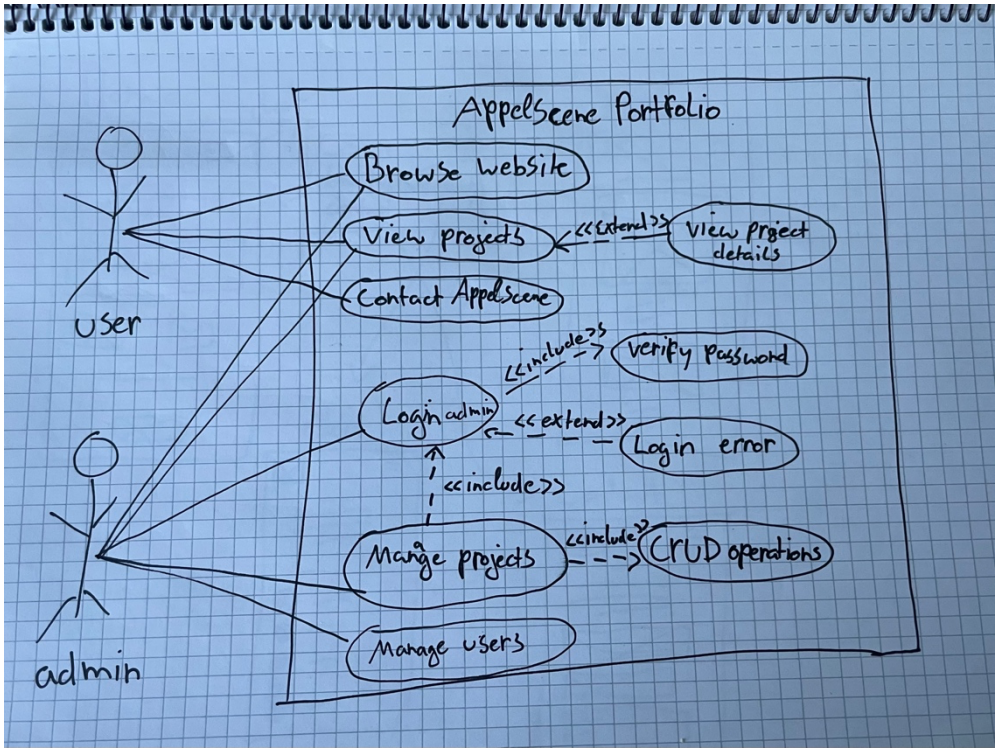
In short, Appelscene is a media company created by five friends. We make music videos, aftermovies, and produce music ourselves! We have been active in Belgium for over a year and are on our way to making our name known.

This project exists to exhibit the music videos we have realized over the past year. It should serve as a portfolio to show to new clients/artists. The aim is to make this as simple but also as unique as possible, so that you can grasp all the information at a glance.

The problem is that we don't really have 1 central place where you can find everything about us. For now, we have an Instagram, YouTube and Linktree account which can be quite confusing. The website should add value by appearing more professional. Users can easily see who we are and what we do. They can also view the different projects and see who we have all worked with already. In this way, we can easily increase and improve our online presence and reputation. Our strongest point is our creativity, and we want to show that in this way. Finally, there is also going to be a contact page so you can easily reach us for questions or possible collaborations.

This website can also function to send to different festivals to record an aftermovie. Of course, the whole team plays a role in the project, but not only ourselves. Also, all clients, artists, and music enthusiasts.

After working out this project, I expect the influx of customers to be even greater than it already was. This will make us more attractive to new artists and new clients who would like to work with us.

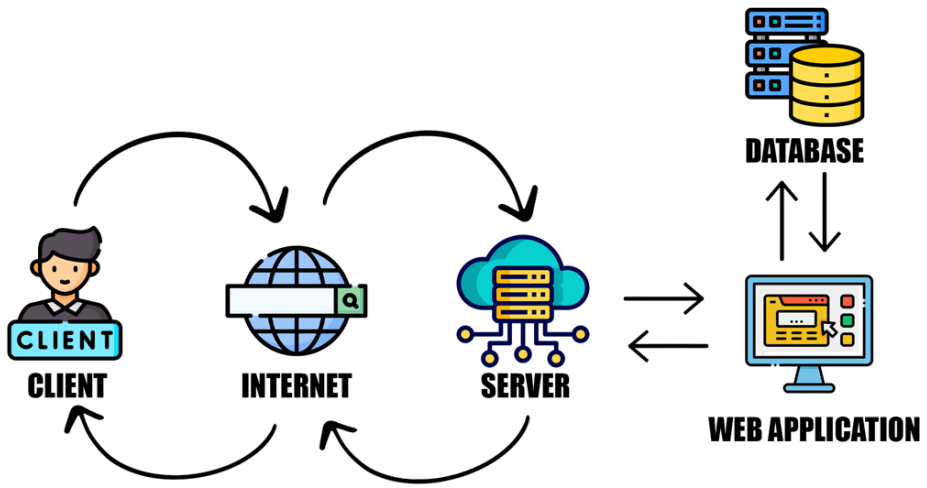


2) Method

2.1) Architecture

The overall structure of this project is not that complex. However, it is certainly interesting to know how it is put together to get a better knowledge of it. The architecture consists of several components that are all connected in some way where they can all work together. In this chapter, I will explain this in more detail.

Below there is an image outlined to make this visually clear:



The components:

- Client / End-user
  - These are the people who will use the application. They will visit the website via the internet and interact with the application. They can scroll through the website, retrieve all the required information, fill in and send a contact form, and so on.
- Web browser / Internet:
  - This is the channel that end users use to access the website. The most well-known web browsers are Chrome, Firefox, Safari and so on. These browsers send HTTP requests to the server, receive various content from it and render web pages that users can use to interact with.
- Server:
  - The server is responsible for handling and sending HTTP requests. They handle tasks like serving static files (HTML, CSS, images) and managing SSL encryption.
- Web application:
  - This is the actual application; it can be considered the beating heart of the website. It contains all the logic for handling various user requests, generating dynamic content and managing interactions with the database.
- Database / storage:
  - This is where all the data is stored. This can be different types such as a relational database, a NoSQL database or a very light embedded one. This ensures that everything is stored in a well-structured way. This allows end users to retrieve and modify data easily.

2.2) Database

The database is mainly about music videos because that is what we, Appelscene, focus on. The portfolio is also entirely about that, so the resources are all about that same subject.
I have 5 tables in total:

- videoclip: This table stores information about video clips, including their titles, release dates, durations, and genre IDs.
- artistVideoclip: This table establishes a many-to-many relationship between artists and video clips, allowing multiple artists to be associated with multiple video clips.
- artist: The "artist" table contains details about artists, including their names and countries.
- genre: This table defines various video clip genres and includes descriptions of those genres.
- users: This table defines the users for user management and their roles.

These tables display the various attributes and their meanings:

Videoclip table	
video_id (PK)	Unique identifier for each videoclip
title	The title of the videoclip
discription	The description of the video
release_date	The date when it got released
link	The youtube link to the videoclip
genre_id (FK)	References the genre of the video clip

Artistvideoclip Table	
artist_id (FK)	References the artist associated with a video clip
video_id (FK)	References the video clip associated with an artist

Artist Table	
artist_id (PK)	Unique identifier for each artist
name	The name of the artist
country	The country of origin
description	The description of the artist

Genre Table	
genre_id (PK)	Unique identifier for each genre
genre_name	The name of the genre (e.g., “Hip hop”)
description	The description of the genre

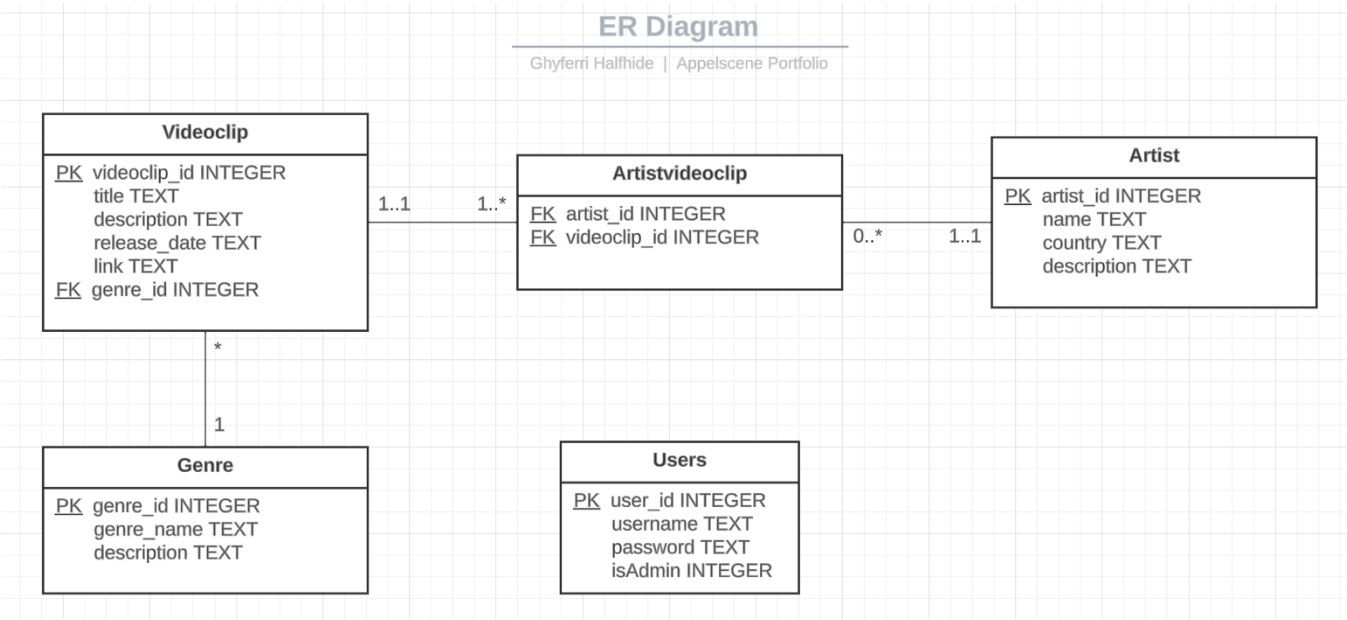
Users Table	
user_id (PK)	Unique identifier for each user
Username	The username of the account
Password	The password of the account
isAdmin	Says if they are an admin or not

The relationships between the different tables are not that difficult to explain. The genre table a one-to-many relationship with the video clips. Each genre can be associated with multiple video clips.
A video clip has a one-to-many relationship with artistvideo clip because each video clip can contain several artists. The same applies to the artist table. Each artist can be associated with several music videos. That is why we have the table artistvideo clip, which acts as a kind of bridge that ensures that there can be a many-to-many between the table's music video and artist.

This table contains, only 2 columns, the "artist_id" and the "video_id". It thus contains a reference to the primary keys of the 2 other tables. So, each row in this table contains a connection. For example, if "artist_id" is 1 and "video_id" is 2 in the "artistvideoclip" table, it means that "Artist 1" is associated with "Video Clip 2." This makes it possible that different artists, can contain different video clips. Also, this makes it possible 1 video clip can have multiple artists.

This data is stored in a relational database management system (RDBMS). This database I use is SQLite3. This system allows me to easily place, store and look up my data in it. The programme is very light and works very efficiently. It supports defining relationships between tables using keys. It stores the entire database in a single file on disk. This way the data easy to distribute and deploy the database. SQLite3 makes it very easy to use to deploy an application and because of this, we don't need complex data management either. No separate server process is needed. With SQL queries, we can easily access all the data and use CRUD (Create, Read, Update, Delete) on them.

Below you can see an overview of the entire ER diagram with the different tables, attributes, their data types, and it also shows how these entities are connected by relationships. This helps in understanding the entire data structure.



2.3) Graphical User Interface

For this project, I chose to go for a clean and modern style. The display should be very simple and self-explanatory so that users can easily find their way around. On the landing page, I also want a big image to grab attention right away and make users want to get to know more. I first started with some lo-fi sketches to already have a better idea. I have 4 pages: the homepage, videos page, about page and the contact page. *See Figure 1.*

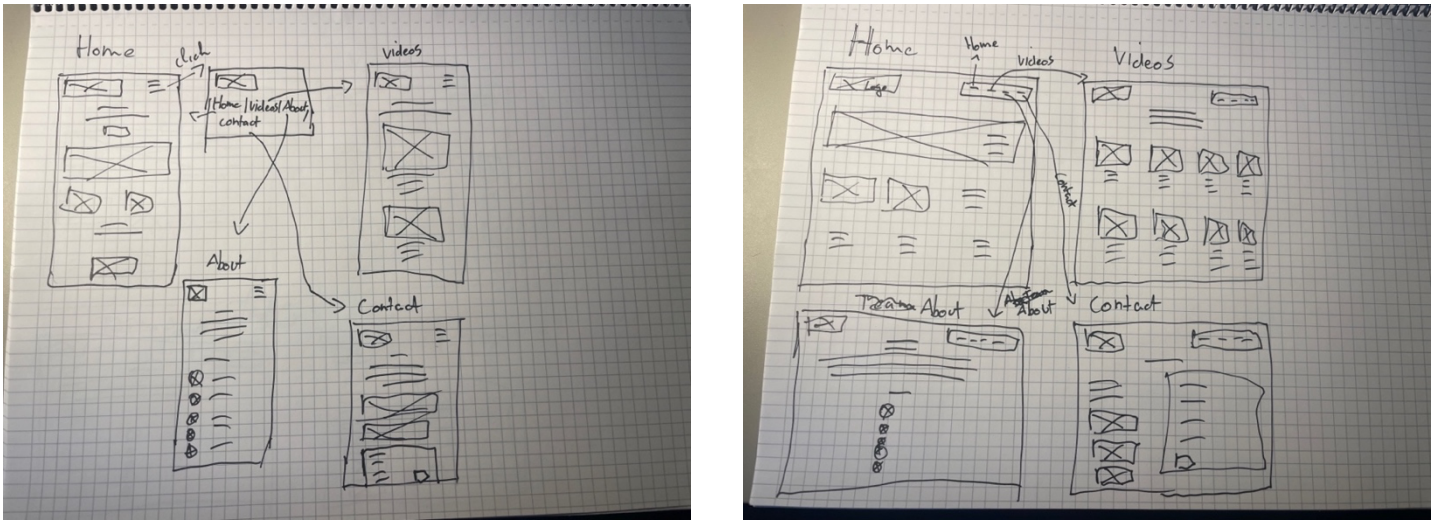


Figure 1, Low Fidelity // Wireframes and concept

I then headed towards mid-fi to see if I was happy with the concept. This had gone well, and I am definitely satisfied with it. I worked out the mid-fi in Figma to quickly have a good picture. See *Figure 2* to get a better picture. Providing me with a comprehensive evaluation of the user experience, the mid-fidelity stage bridged the gap between early sketches and high-fidelity designs. It granted me a clear understanding of interaction and functionality regarding the concept.

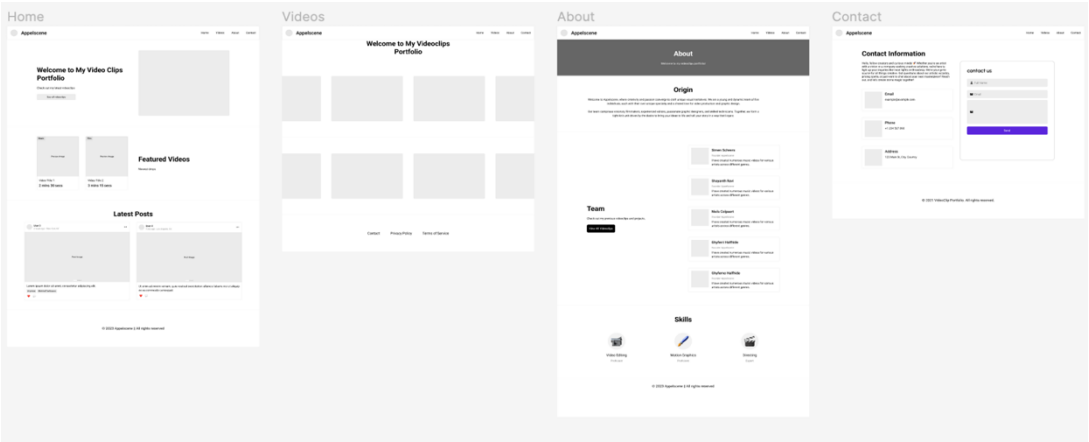


Figure 2, Mid Fidelity

2.4) Web Application

2.4.1 Technology

I developed the web application by using different technologies, languages, frameworks, and packages. The combination of these resulted in a well-functioning project with a good design. Work has been done with front-end, back-end and even a database management:

Languages/framework:

- HTML (HyperText Markup Language): This provides the foundation of the web application. This is used to structure different content and layout already a proper place.
- CSS (Cascading Style Sheets): This takes care of styling and making the different screens visually appealing. This is mainly where the font, colors, custom styles are created. This also takes care of making all my screens responsive.
- JS (Javascript -> server side): This language was used to display interaction and dynamic content behavior on the pages. With this I could easily do form validation or check if a user is authenticated or not.
- Handlebars: This is a language that works with a specific template. It allows you to work with reusable templates which allows HTML to be rendered dynamically on the server-side.
- Spectre.css: This is a CSS framework that I used to design my web application quickly but nicely. This also helps to make it responsive in a simple way.
- Node.js: This is JS on the server side, allows for server-side scripting.
- Express.js: A framework for Node.js that makes it easier to create server-side web applications.
- SQLIte3: A relational database management system for storing and extracting data.

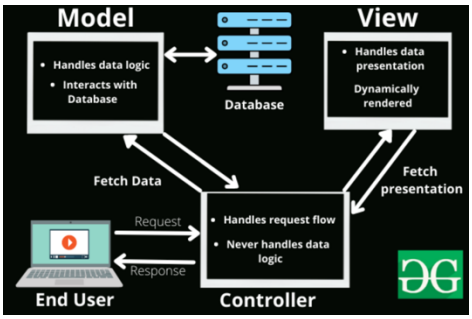
Libraries/packages:

- bcrypt: This is used to store passwords in a secure way. You can read more in the following chapters.
- Body parser: With this you can easily extract incoming HTTP requests and use it for user submissions.
- Connect-sqlite3: This makes the connection between session management and storing in SQLite. It is used to maintain session data.
- Cookie parser: This is used for handling HTTP cookies.
- Express-session: This is used for maintaining user sessions to server-side.

```
{
  "name": "portfolio-appelscene",
  "version": "1.0.0",
  "description": "Code for handlebars MVC (Model-View-Controller)",
  "main": "app.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "handlebars in Express."
  ],
  "author": "GH",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "connect-sqlite3": "^0.9.13",
    "cookie-parser": "^1.4.6",
    "express": "^4.18.2",
    "express-handlebars": "^7.1.2",
    "express-session": "^1.17.3",
    "sqlite3": "^5.1.6"
  }
}
```

2.4.2 Design Pattern

I have structured my web application following the MVC (Model-View-Controller) pattern. MVC is a design pattern used in many applications. It divides the application into 3 components:



MVC Framework Diagram obtained from GeeksforGeeks (https://www.geeksforgeeks.org/mvc-framework-introduction/)

- Model: This contains the data and all the business logic of the application. With this you can go and retrieve the data, edit it, and put it back into the database. This is where we usually interact with the database.

- View: This one is responsible for presenting the data to the user. It handles rendering the appropriate components and showing them to the user.

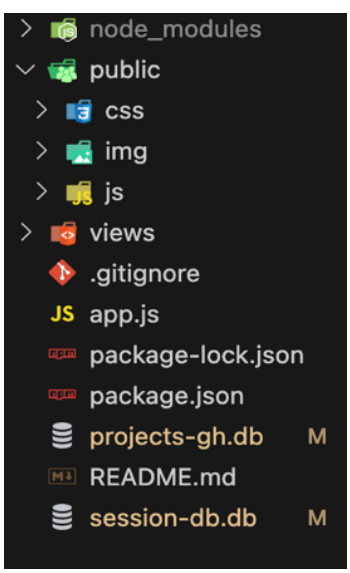
- Controller: This is saying the connection between the Model and View. It receives user input, handles it, and then goes on to interact with the Model to retrieve or edit data. It then goes on to render in the correct View. This is usually also where the different routes are defined.

The reason why I applied this is to make my code clearer and easier to maintain. In conjunction with the Handlebars, we can easily reuse code and templates to dynamically display different content to the end users. The Handlebars dynamically populate placeholders with the data they receive.

I did partition my folders and file. I did this so that I have easy structuring and organization in my code. This makes it easy to read and maintain the code.

I also used some middleware functions. These were mainly for requests and responses. For example, I have a middleware that logs every time I visit a URL. This was mainly the testing phase to see if everything arrives correctly.

I also have a middleware that parses the body request where. I also have middleware that will send a session to SQLite's storage. This way I don't have to do that every time.




```
// define session
app.use(
  session({
    store: new SQLiteStore({ db: "session-db.db" }),
    saveUninitialized: false,
    resave: false,
    secret: "this!@123I%%%%sASecret678Sentence@!!!#",
  })
);
// log user url
app.use((req, res, next) => {
  console.log(`Visited URL: ${req.url}`);
  next();
});
```

2.4.3 Security

Security is something this digital age must always think about. We must do our best to keep web applications secure to often work with sensitive data. This chapter is going to tell about all the things I have thought about and implemented in my current project. I have done my best to strengthen most of the major weaknesses.

2.4.3.1 HTTPS

HTTPS ensures that the data passed back and forth between the client and server is encrypted in a secure manner. This prevents people trying to steal the data from getting between the 2. It also keeps the data unchanged and prevents attackers from touching or editing it. SSL/TLS certificates are used to ensure that the server is considered secure. If you only use HTTP, then someone can get between the client and server to intercept sent text.

I myself have generated certificates that do work locally. These are not official, just for local testing development. My server now works only on https://localhost:8080. There was a possibility to get a real certificate for free, but since we work only on localhost, it is not really necessary.

2.4.3.2 Bcrypt

I have been using Bcrypt to hash my passwords. I created a user management system where the administrator can perform CRUD operations on the other users. Because I work with 2 roles, a normal user, and an administrator, I create these 2 first in me code. So, for this it is important that the passwords are hashed before they get into my database. This will prevent the hackers from decrypting the passwords. Also, the administrator has CRUD rights to manage the video clips. The rest, regular users/non-users, can only view the video clips. If you are not a user, you always have the option to register yourself on the registration page. You will not get administration rights though.

As I mentioned earlier, I used Bcrypt. Bcrypt is a library used on password hashing. With this I can prevent brute forces or rainbow table attack. This is an easy library to install and use. By using salting, a unique string is generated for each password that is impossible to unravel.

User data:

Role	**Username**	**Password**
Administrator	admin	admin123
Regular user (no crud rights)	user	user123

I've also prepared a videoclip that u can add with valid data:

Artist	Title	Release	Description	Link	Image	Genre
Yorokobi	Reconaissance	31/03/2023	Young French Boy	https://www.youtube.com/embed/oaD4KfVUfgg?si=ybpxaeHXD3gkcQ_1	/img/reconaissance_thumbnail.png	French

2.4.3.3 Sessions

I used sessions because we are getting information from the client to the server. The user logs himself into the login page and this information is sent to the server. We want to store this information because we want to keep track of user staying logged in. This session is also stored into the database. This makes that data less vulnerable because you can't steal it from the client-side cookies or URL.

The session is created when a user logs into the system. It is then also terminated when the user logs out of the system. To associate the user's session with their browser, a unique session ID is generated and stored in a cookie on the client's device.

2.4.3.4 Role based

Here I use authorization to check if the right user does have access to a particular page. so I always check if a use is logged in or not and in doing so I check if the user has admin rights or not. The users all have an attribute isAdmin. If this is 1 user has CRUD rights, if not you are redirected to another page.

I have also implemented error handling in my code. For example, I have provided a 403 page when a user does not have permissions. This is a standard HTTP response that the server understands. I have also provided a 404 page when the user tries to reach pages that do not exist.

```
if (!request.session.isLoggedIn || request.session.isAdmin == 0) {
```

2.4.4 Version Control

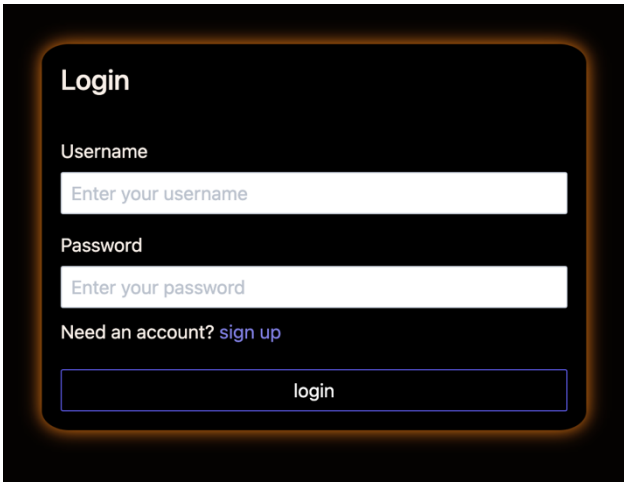
The version control that is used is Git. Git is very simple and easy to use. Here I can easily view the history of my project and collaborate with other people. I used GitHub as a web-based platform to maintain my Git repositories. Using this is also very simple.

Normally I work with branches per feature, but in this case, I chose not to do that and work on the main branch. This is because the complexity was not very high and because I don't have to collaborate with other people. This allowed me to easily go through my commits. This is also my preference for smaller projects and individual work. Then I also don't have to perform a merge request every time.

https://github.com/ghyferri/portfolio-appelscene

3) Results

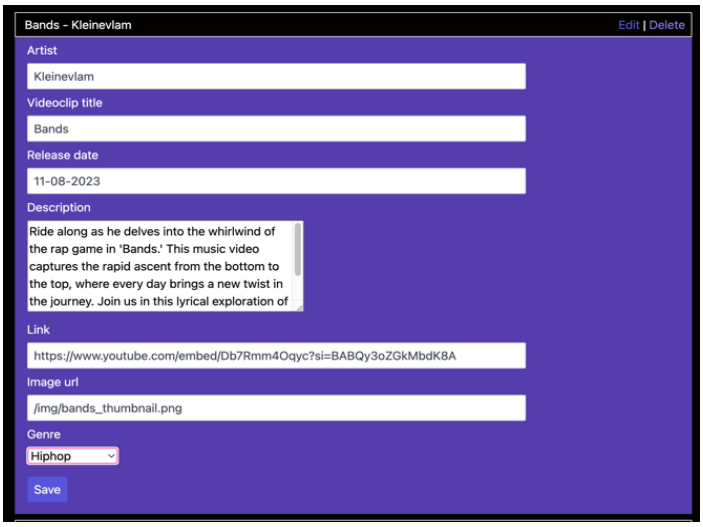
I developed a full stack web application. From frontend to backend to the database. The project was quite an adventure and it paid off with a great result.



In the beginning we have a login page where you can login as a user or as an administrator. There are 2 roles available, one has possibility of CRUD operations and the other does not. While brainstorming and designing I had forgotten about the login functionality, but it was very easy to design a screen for that.

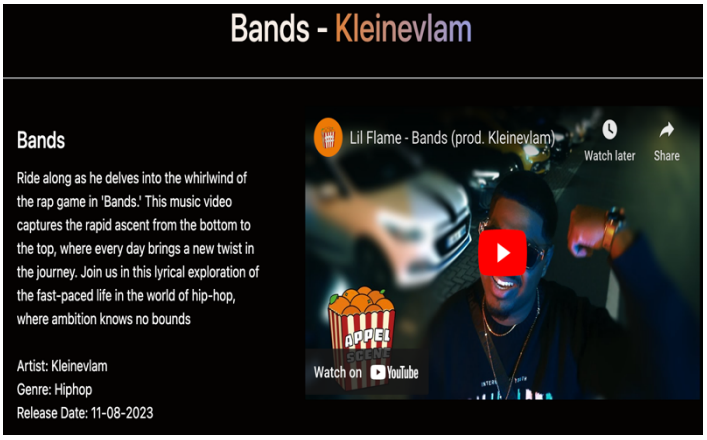
If you don't have an account, you can register yourself as a user. Only you don't get admin rights. When the administrator logs in, he has the ability to manage the other users. He can create new users, delete users, and modify existing users to give them admin rights. He can also change the names of the users.

Next, the administrator can maintain different projects. He can add, modify, and delete video clips. You can also change the artist's name. Because multiple tables are worked with, multiple innerjoins are also done to complete them. When you do this, everything is updated automatically giving you a good overview. Doing multiple updates on different tables in 1 code was a bit complex, but it all worked out in the end.



Next, all users/non-users can go to the videos tab and watch all video clips. These are displayed per 4, I made a pagination so that only 4 clips are displayed per page. This automatically calculates the number of max video clips and how many pages it should provide.

When you click on 1 video you get to see the detail page of 1 video clip. This displays the title, artists, description, release date, genre and a link which is converted to a YouTube video.



For the rest you have the about and contact pages. There are also pages provided for other routes such as 404 and 403. The whole web application is responsive which means that you can use it fully on your phone. This took some time and effort, but it was worth it.

4) Discussion

The project was a fun experience. It gave me more self-confidence because now I am sure that I can build a whole web application on my own. In such a short time we have accomplished quite a lot I personally think. I didn't find it difficult to choose a topic for this project. That makes it even more fun because I can work out something about something I am really passionate about.

The only really new thing I learned was handlebars. Handlebars in itself was a good tool to use. It was not difficult to learn, and everything was actually self-explanatory. The difficulty of the different software was at a good level. Also, it had been a long time since I had worked with SQL. I was eventually able to find my way around SQLite3.

Finally, I also worked with Spectre.css framework for the first time. This was not difficult to use. It has many of the same aspects as Bootstrap. Only I found that the documentation was not always complete and often you really have to figure out for yourself how something works.

Overall, I am definitely very proud of the product I delivered and can confidently say that I enjoyed this adventure!

5) References

Chua, E. H. C. (n.d.). *HTTP - Hypertext Transfer Protocol*. Nanyang Technological University. Retrieved October 8, 2023, from https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html

GeeksforGeeks. (n.d.). *MVC Framework Introduction*. Retrieved October 9, 2023, from https://www.geeksforgeeks.org/mvc-framework-introduction/

LaViska, C. (2023, August 4). *Hashing passwords with Node.js and bcrypt*. A Beautiful Site. Retrieved October 8, 2023, from https://www.abeautifulsite.net/posts/hashing-passwords-with-nodejs-and-bcrypt

GitHub. (n.d.). GitHub Docs. Retrieved October 9, 2023, from https://docs.github.com/en

Li, J. (2023, August 4). *Enable HTTPS for localhost during local development in Node.js*. Plain English. Retrieved October 9, 2023, from https://plainenglish.io/blog/enable-https-for-localhost-during-local-development-in-node-js