Containerization Assignment

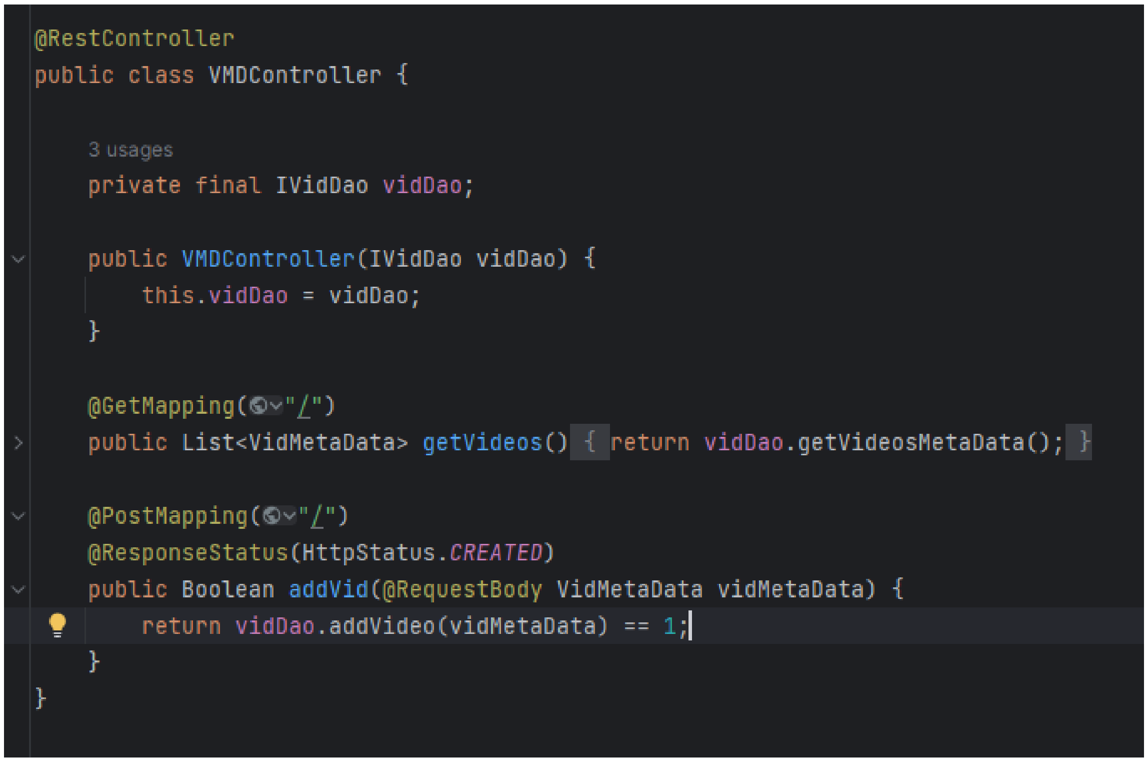
By: Ibrahem Jaradat.

Introduction:

In this assignment we were required to build a containerized Video Streaming System. Where this Web app is used to view videos. Users are allowed to view videos after validating their credentials through the Authentication Service. The list of videos and their paths are taken from MySql DB service, and the video itself can be read from the file storage through the File storage system service.

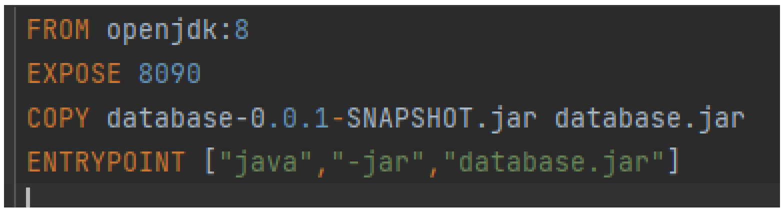
- Database Controller Service.

This service is to handle the requests going to the database, it connects to the MySQL database image where it stores the video's metadata, and for simplicity we have only two endpoints, one for getting all videos metadata, and one for adding new video.



```
@RestController
public class VMDController {

    3 usages
    private final IVidDao vidDao;

    public VMDController(IVidDao vidDao) {
        this.vidDao = vidDao;
    }

    @GetMapping("/")
    public List<VidMetaData> getVideos() { return vidDao.getVideosMetaData(); }

    @PostMapping("/")
    @ResponseStatus(HttpStatus.CREATED)
    public Boolean addVid(@RequestBody VidMetaData vidMetaData) {
        return vidDao.addVideo(vidMetaData) == 1;
    }
}
```

Database Controller Service Image: Here with simple and straightforward containerization; building a jar file with maven then creating this docker file.

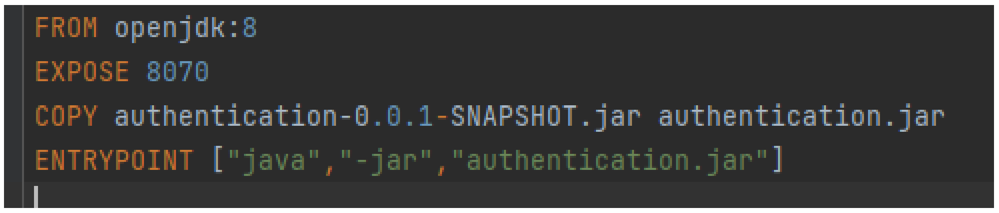


```
FROM openjdk:8
EXPOSE 8090
COPY database-0.0.1-SNAPSHOT.jar database.jar
ENTRYPOINT ["java","-jar","database.jar"]
```

- Authentication Service.

This service handles the validation of the user's credentials where each and every request in the application should be authenticated, so it's a simple service where other services send user credentials to validate.
For demonstration purposes, the username and password are "admin".

Authentication Service Image :

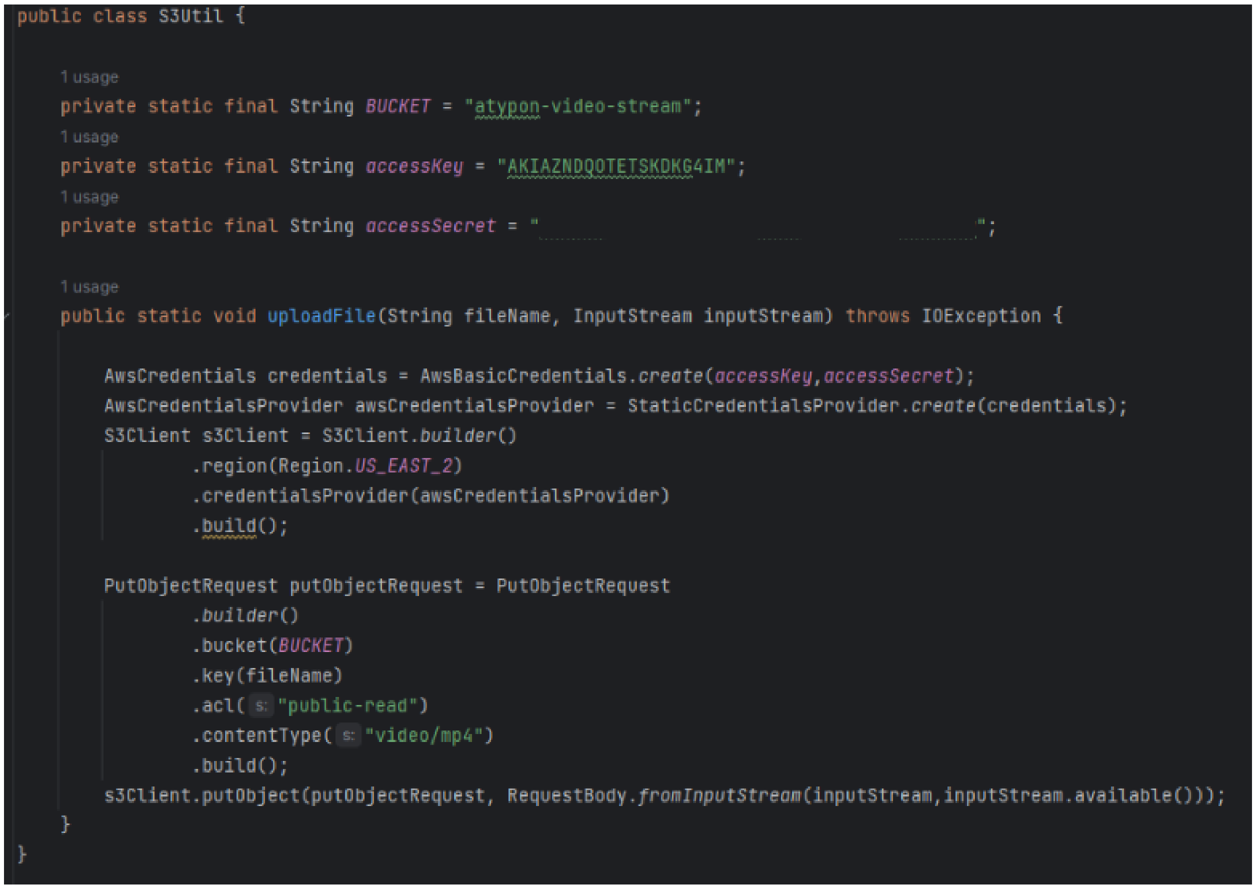


```
FROM openjdk:8
EXPOSE 8070
COPY authentication-0.0.1-SNAPSHOT.jar authentication.jar
ENTRYPOINT ["java","-jar","authentication.jar"]
```

-File System Service.

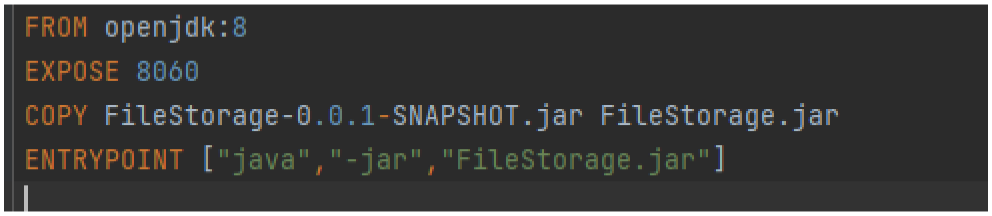
In my approach I decided to go with using (AWS S3) in my project, where this service has the responsibility of uploading the videos to the S3 bucket after the user uploads it through the upload service.

It manages the upload process to the bucket where it receives a multipart file with a name of the video and upload it using this utility.



```
public class S3Util {

    1 usage
    private static final String BUCKET = "atypon-video-stream";
    1 usage
    private static final String accessKey = "AKIAZNDQOTETSKDKG4IM";
    1 usage
    private static final String accessSecret = "[redacted]";

    1 usage
    public static void uploadFile(String fileName, InputStream inputStream) throws IOException {

        AwsCredentials credentials = AwsBasicCredentials.create(accessKey,accessSecret);
        AwsCredentialsProvider awsCredentialsProvider = StaticCredentialsProvider.create(credentials);
        S3Client s3Client = S3Client.builder()
                .region(Region.US_EAST_2)
                .credentialsProvider(awsCredentialsProvider)
                .build();

        PutObjectRequest putObjectRequest = PutObjectRequest
                .builder()
                .bucket(BUCKET)
                .key(fileName)
                .acl( s: "public-read")
                .contentType( s: "video/mp4")
                .build();
        s3Client.putObject(putObjectRequest, RequestBody.fromInputStream(inputStream,inputStream.available()));
    }
}
```

File System Service Image:

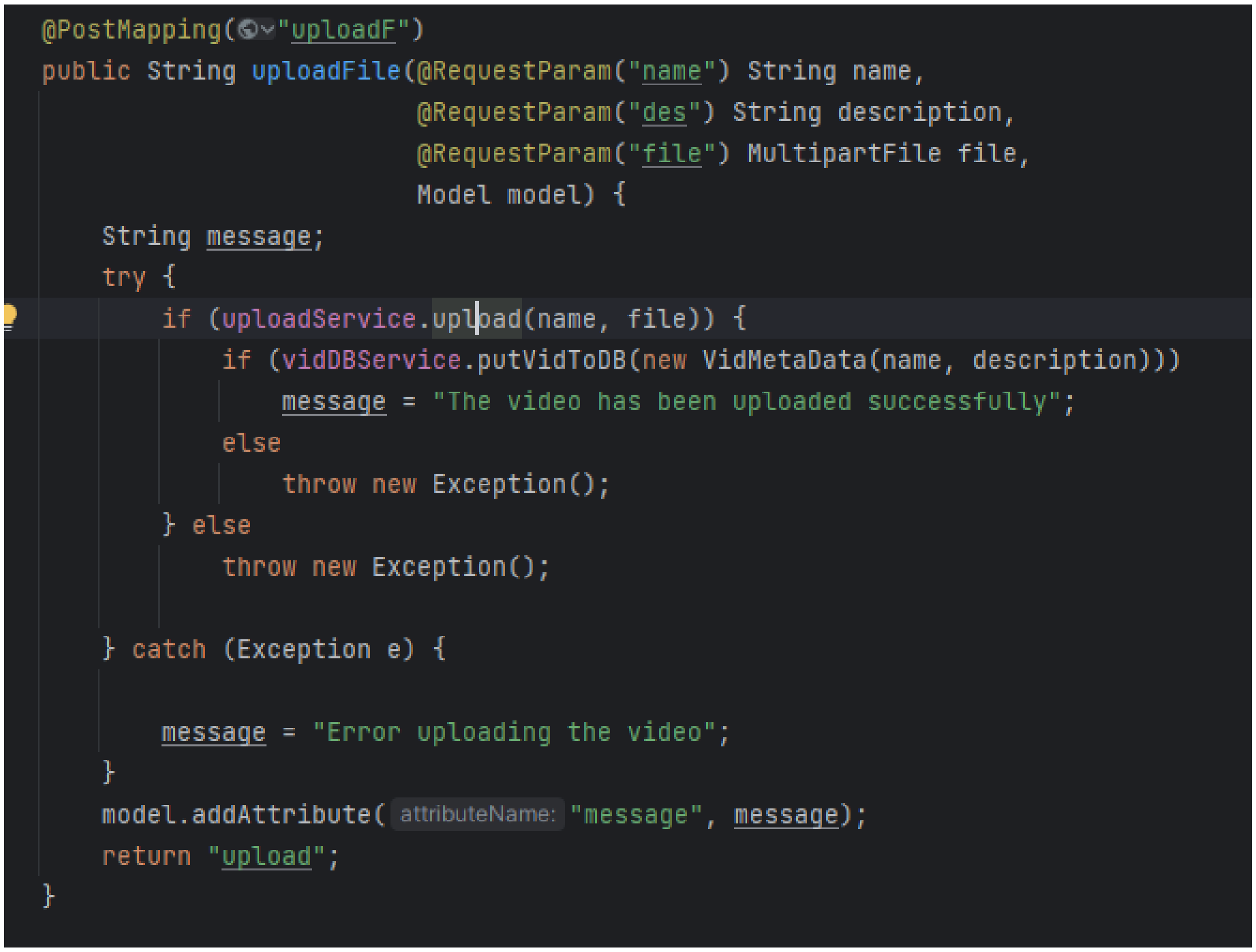


```
FROM openjdk:8
EXPOSE 8060
COPY FileStorage-0.0.1-SNAPSHOT.jar FileStorage.jar
ENTRYPOINT ["java","-jar","FileStorage.jar"]
```

- The Upload Service.

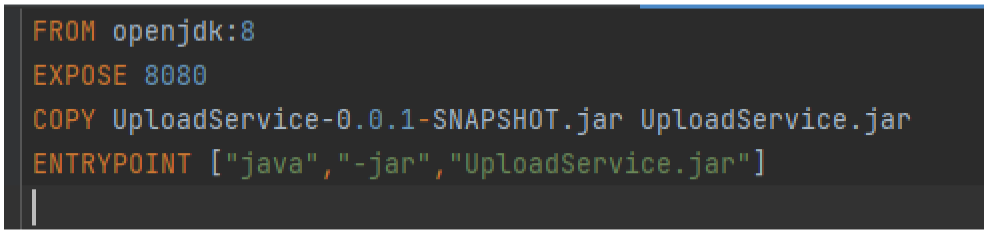
The Upload Service; The user can upload videos, all after being authenticated.
with a simple UI given to the user, it grants the ability to log in to the system and start uploading. Firstly the user have to be authenticated, then redirected to the upload page to upload a video with a name and description, the metadata{video name, video description, URL and the username will be sent to the

(Database Controller Service) to be saved into the database, and the video itself will be uploaded to the S3 bucket through the (File System Service).



```java
@PostMapping("uploadF")
public String uploadFile(@RequestParam("name") String name,
                         @RequestParam("des") String description,
                         @RequestParam("file") MultipartFile file,
                         Model model) {
    String message;
    try {
        if (uploadService.upload(name, file)) {
            if (vidDBService.putVidToDB(new VidMetaData(name, description)))
                message = "The video has been uploaded successfully";
            else
                throw new Exception();
        } else
            throw new Exception();

    } catch (Exception e) {

        message = "Error uploading the video";
    }
    model.addAttribute( attributeName: "message", message);
    return "upload";
}
```

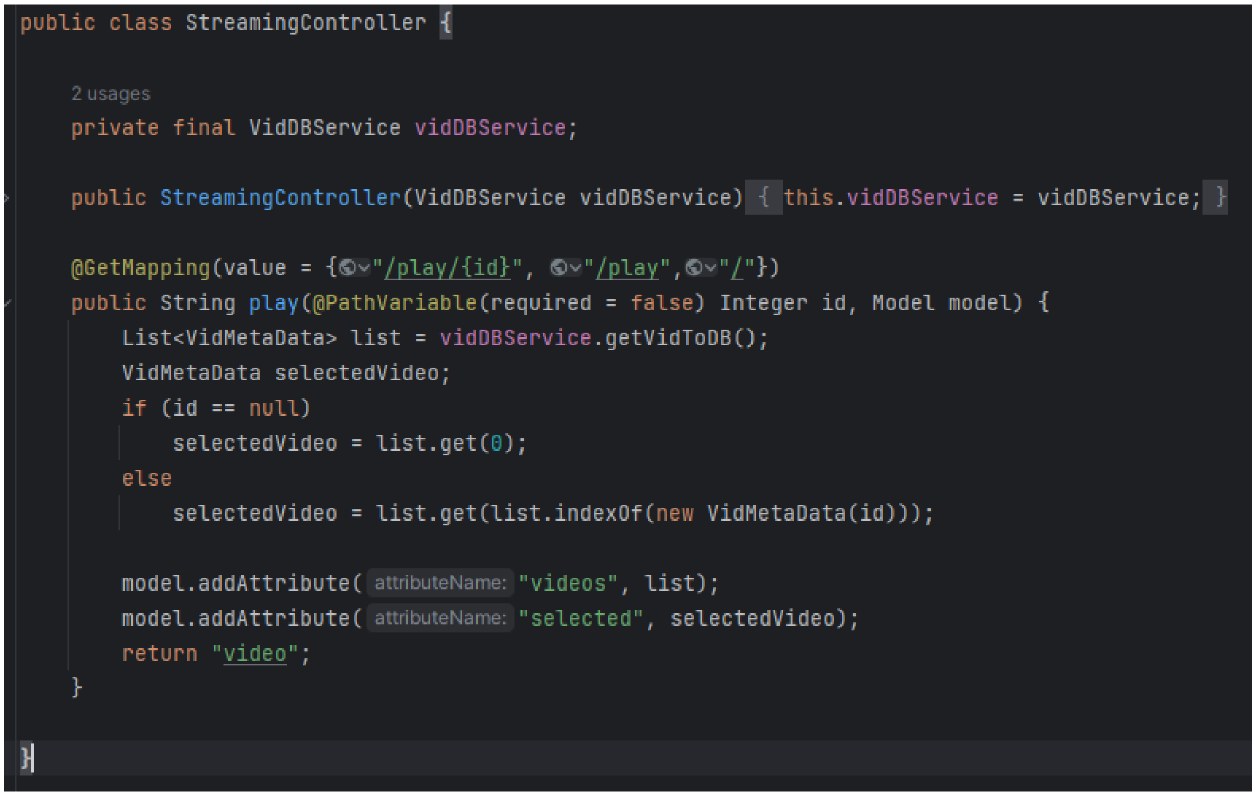
The Upload Service Image:



```dockerfile
FROM openjdk:8
EXPOSE 8080
COPY UploadService-0.0.1-SNAPSHOT.jar UploadService.jar
ENTRYPOINT ["java","-jar","UploadService.jar"]
```

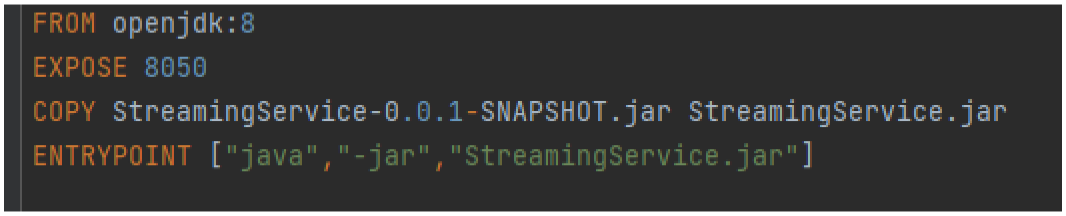
- The Streaming Service.

To get the video's data from the database and then stream the data to the user from S3.
So if user uploads some videos, this service gives him the ability to watch these videos straight from the S3 as a stream, after he's authenticated, the user has a list of videos to watch, where the videos data comes from the database through (Database Controller Service) and the video streams from the S3 after it's uploaded from the (File System Service).



```java
public class StreamingController {

    2 usages
    private final VidDBService vidDBService;

    public StreamingController(VidDBService vidDBService) { this.vidDBService = vidDBService; }

    @GetMapping(value = {"/play/{id}", "/play","/"})
    public String play(@PathVariable(required = false) Integer id, Model model) {
        List<VidMetaData> list = vidDBService.getVidToDB();
        VidMetaData selectedVideo;
        if (id == null)
            selectedVideo = list.get(0);
        else
            selectedVideo = list.get(list.indexOf(new VidMetaData(id)));

        model.addAttribute( attributeName: "videos", list);
        model.addAttribute( attributeName: "selected", selectedVideo);
        return "video";
    }

}
```

The Streaming Service Image:

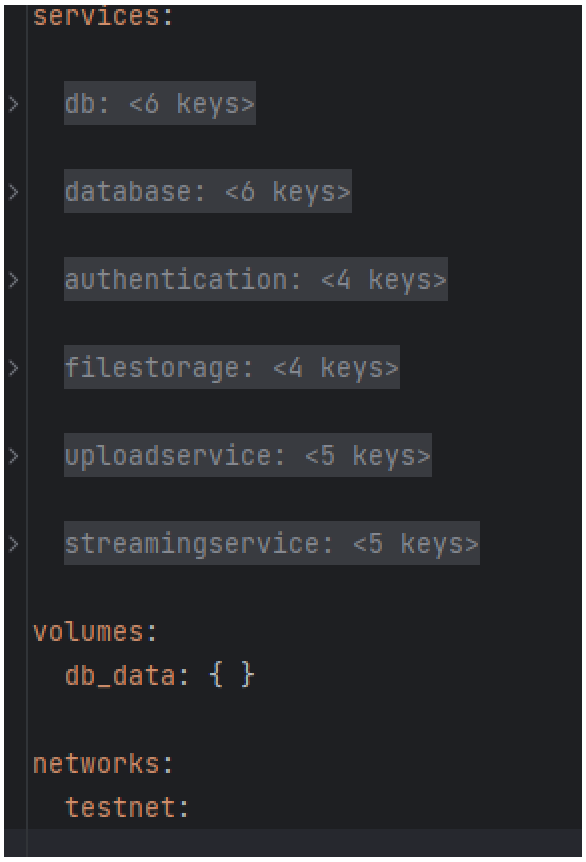


```dockerfile
FROM openjdk:8
EXPOSE 8050
COPY StreamingService-0.0.1-SNAPSHOT.jar StreamingService.jar
ENTRYPOINT ["java","-jar","StreamingService.jar"]
```

- Docker-Compose:

After implementing our services and creating docker files to create the images for the services, we are now ready to run our system and bring it to life using docker compose.

Docker Compose Content:

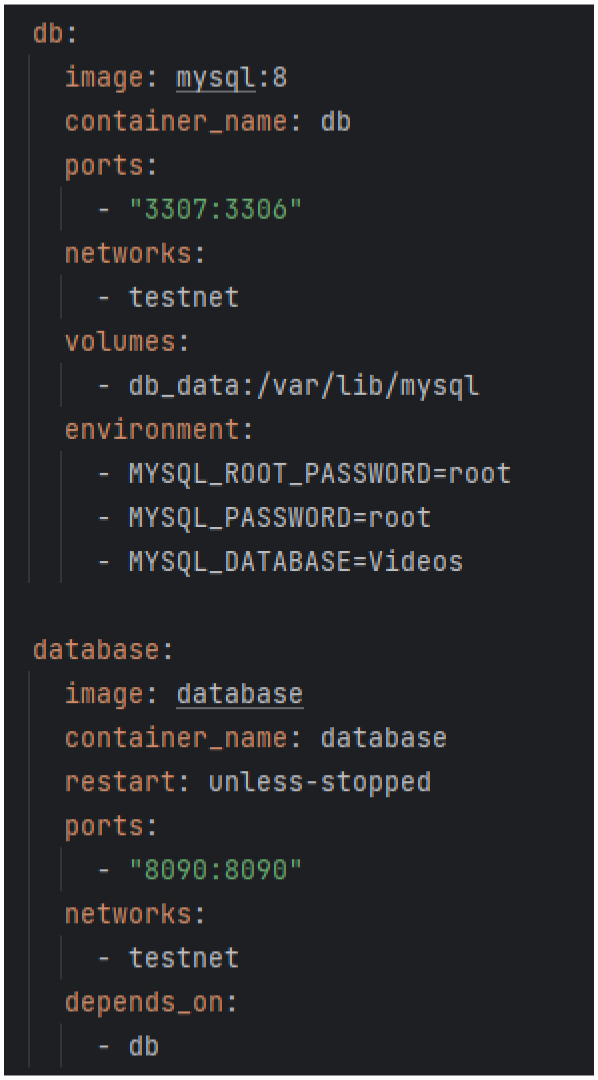


```
services:

  db: <6 keys>

  database: <6 keys>

  authentication: <4 keys>

  filestorage: <4 keys>

  uploadservice: <5 keys>

  streamingservice: <5 keys>

volumes:
  db_data: { }

networks:
  testnet:
```

So we need a network to share through our system, and for that I have created a simple network “testnet”, a volume to store the database data, so it does not erase when the containers are gone.

Then configured our five Services Images as the following:

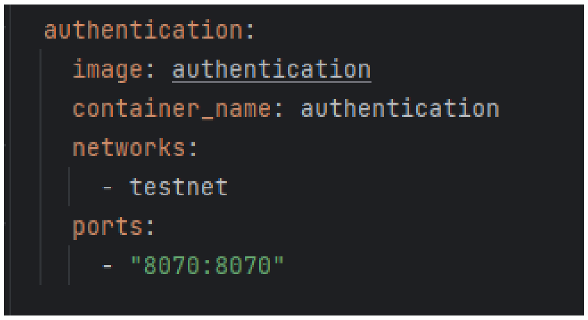
- Database Controller Service:



```yaml
db:
  image: mysql:8
  container_name: db
  ports:
    - "3307:3306"
  networks:
    - testnet
  volumes:
    - db_data:/var/lib/mysql
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_PASSWORD=root
    - MYSQL_DATABASE=Videos

database:
  image: database
  container_name: database
  restart: unless-stopped
  ports:
    - "8090:8090"
  networks:
    - testnet
  depends_on:
    - db
```

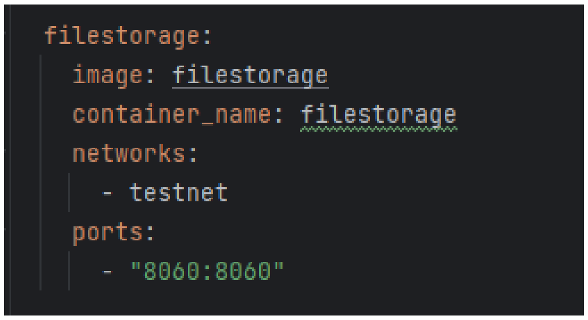
I'm using the standard MySQL image with the dedicated network and volume, with my Database Controller Service image to control it.

Authentication Service: Here connecting and exposing the (Authentication Service image) with its dedicated port.



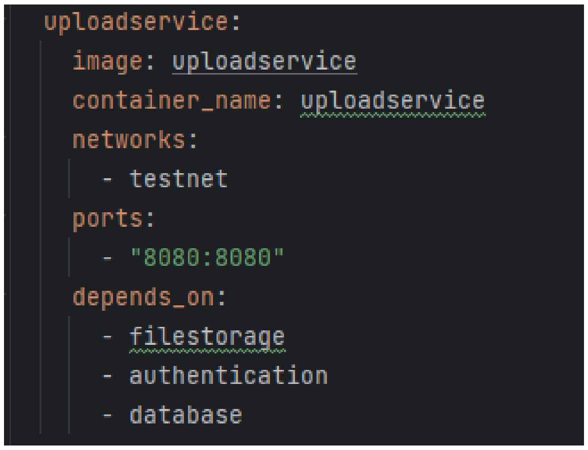
```
authentication:
  image: authentication
  container_name: authentication
  networks:
    - testnet
  ports:
    - "8070:8070"
```

File System Service: As the previous doing here in the (File System Service image).



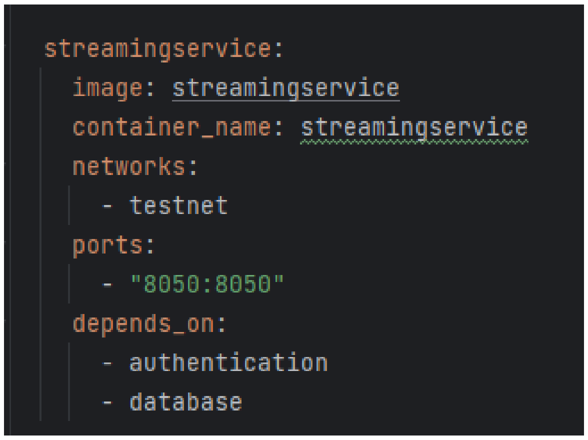
```
filestorage:
  image: filestorage
  container_name: filestorage
  networks:
    - testnet
  ports:
    - "8060:8060"
```

The Upload Service: Exposing and connecting the image, also mentioning the depends on images



```
uploadservice:
  image: uploadservice
  container_name: uploadservice
  networks:
    - testnet
  ports:
    - "8080:8080"
  depends_on:
    - filestorage
    - authentication
    - database
```

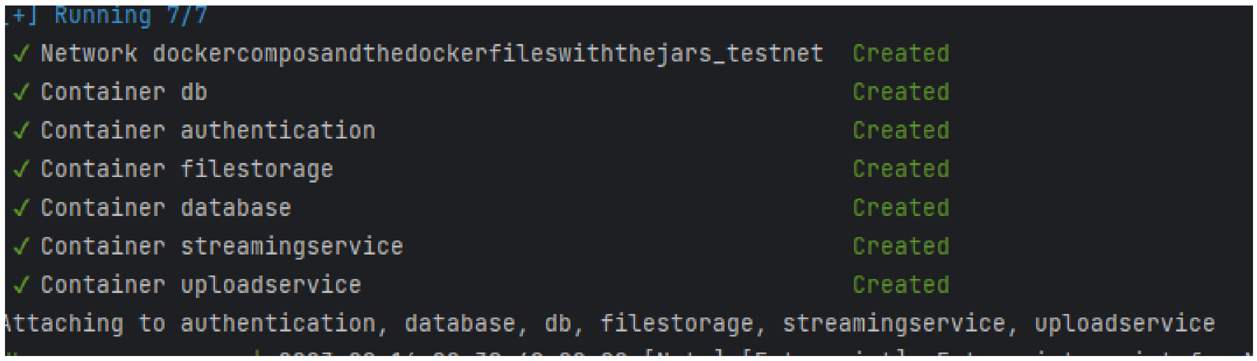
The Streaming Service: Exposing and connecting the image, also mentioning the depends on the authentication and the database controller.



```
streamingservice:
  image: streamingservice
  container_name: streamingservice
  networks:
    - testnet
  ports:
    - "8050:8050"
  depends_on:
    - authentication
    - database
```

Docker-Compose up:

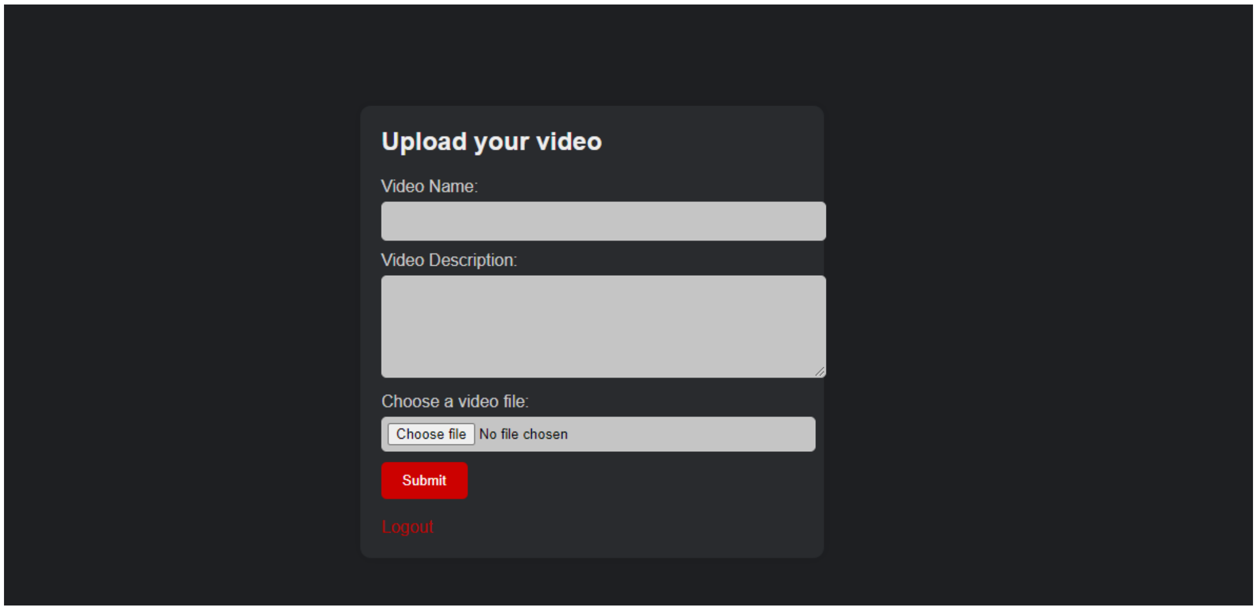
After creating the docker-compose yml file and having the images, Now we are ready to ship and run our project with - Docker-Compose up - .



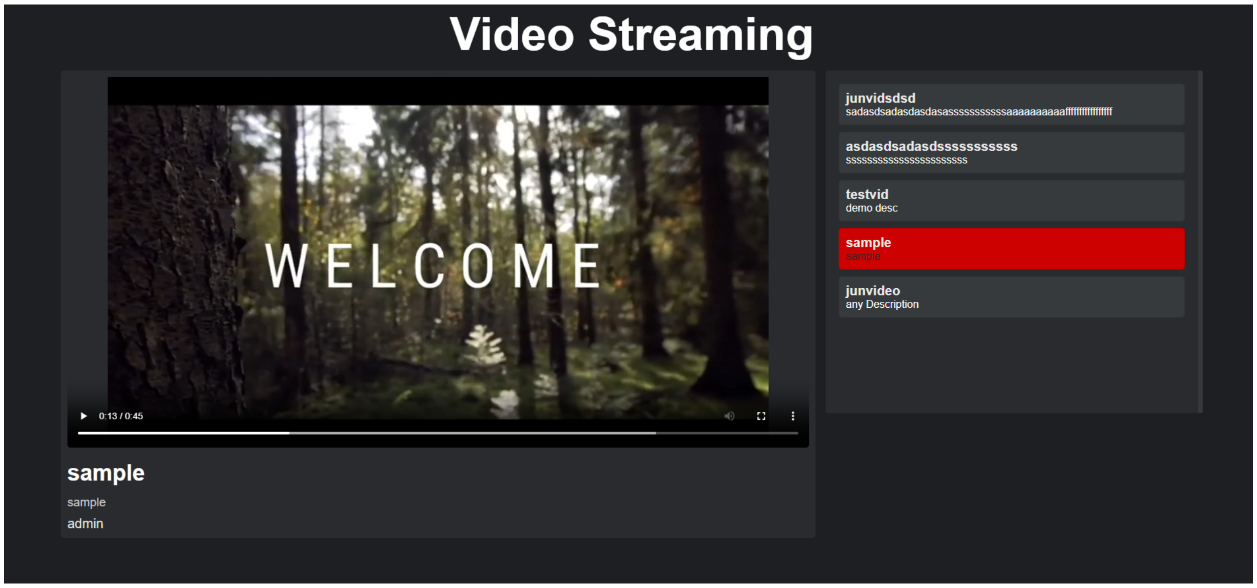
```
+] Running 7/7
✓ Network dockercomposandthedockerfileswiththejars_testnet  Created
✓ Container db                                              Created
✓ Container authentication                                  Created
✓ Container filestorage                                     Created
✓ Container database                                        Created
✓ Container streamingservice                                Created
✓ Container uploadservice                                   Created
Attaching to authentication, database, db, filestorage, streamingservice, uploadservice
```

As we see, we have six containers running together, each container handling a single responsibility.

Leaving Us to interact with :



A simple Upload Page from port 8080.



junvidsdsd
sadasdsadasdasdasasssssssssssssaaaaaaaaaaffffffffffffffff

asdasdsadasdsssssssssss
sssssssssssssssssssssss

testvid
demo desc

sample
sample

junvideo
any Description

sample
sample
admin

And a Streaming page with a video player and list of videos to choose from, at port 8050.