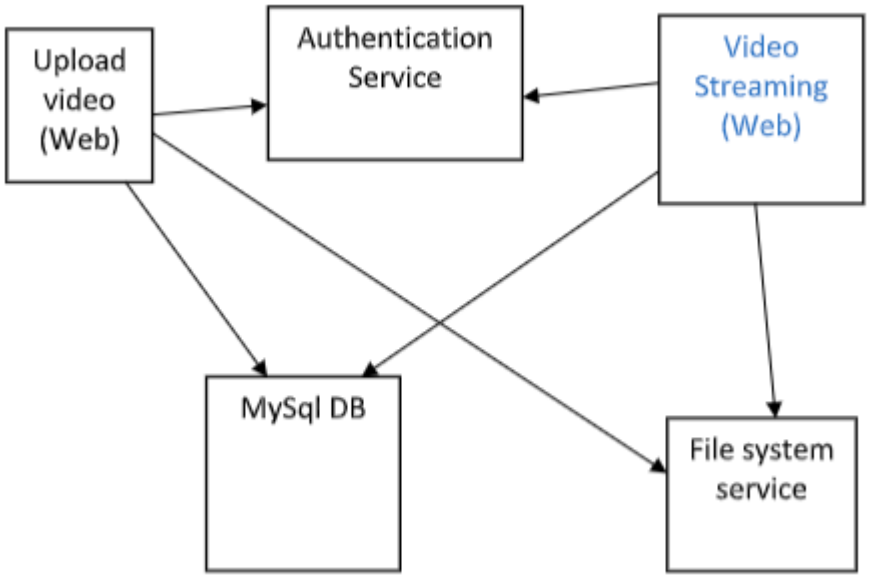
Introduction

System Design



We used the following services in our design:

- Authentication server: to authenticate the authorized users
- Database Controller service: to handle the requests to the database
- Upload Service: to handle uploading the videos
- Streaming Service: A service to get the data from the database and get videos from AWS S3
- File System service: I have gone with using 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.

Authentication Service

Every request of my application should be authenticated; 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 target/authentication.jar authentication.jar
ENTRYPOINT ["java","-jar","authentication.jar"]
```

Database Controller Service

In this service, it connects to the MySQL database image where we store our video's metadata, and given simpler and more strict access to the database, and for simplicity we have only two endpoints, one for getting all videos metadata, and one for adding new video.

```
@RestController
public class VideoMetaDataController {

    3 usages
    private final IVideoDao videoDao;

    public VideoMetaDataController(IVideoDao videoDao) { this.videoDao = videoDao; }

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

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

Database Controller Service Image

Going simple here with straight forward containerization; building a jar file with maven then creating this docker file

```
FROM openjdk:8
EXPOSE 8090
COPY  target/database_controller.jar database_controller.jar
ENTRYPOINT ["java","-jar","database_controller.jar"]
```

File System Service

As mentioned, I'm using S3 as my storage, This service Handling 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 AWSS3Utility {

    1 usage
    private static final String BUCKET = "atypon-video-islam";
    1 usage
    private static final String accessKey = "AKIASC2N7WFWDD5B4TOY";
    1 usage
    private static final String accessSecret = "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";

    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.EU_WEST_2)
                .credentialsProvider(awsCredentialsProvider)
                .build();

        PutObjectRequest putObjectRequest = PutObjectRequest
                .builder()
                .bucket(BUCKET)
                .key(fileName)
```

File System Service Image

```
FROM openjdk:8
EXPOSE 8060
COPY  target/storage.jar storage.jar
ENTRYPOINT ["java","-jar","storage.jar"]
```

The Upload Service

A service with a simple UI giving the user the ability to log in to the system and uploading a video, First the user have to be authenticated, after he got 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 the S3 through the **File System Service.**

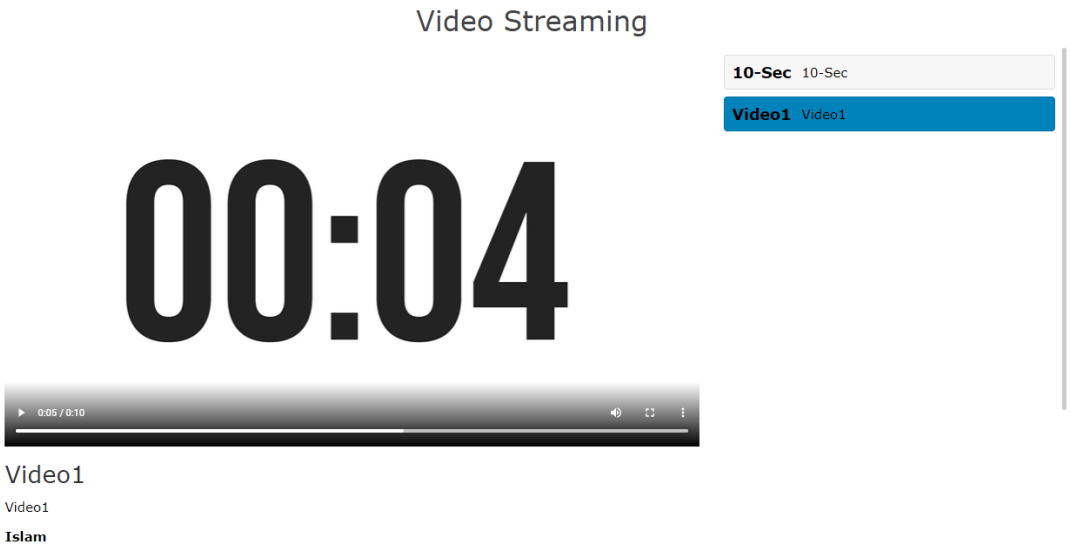
```
@PostMapping("uploadF")
public String uploadFile(@RequestParam("name") String name,
                         @RequestParam("des") String description,
                         @RequestParam("file") MultipartFile file,
                         Model model) {
    String message;
    try {
        if (uploadService.upload(name, file)) {
            if (vidDBService.putVidToDB(new VideoMetaData(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

```
FROM openjdk:8
EXPOSE 8080
COPY target/upload.jar upload.jar
ENTRYPOINT ["java","-jar","upload.jar"]
```

The Streaming Service

Let us say the user has uploaded some videos, this service giving him the ability to watch these videos straight from the S3 as a stream, after he has been authenticated he got a list of videos to watch, where the videos data came from the database through **Database Controller Service** and the video streamed from the S3 after it was uploaded from **File System Service.**



The Streaming Service Image

```
FROM openjdk:8
EXPOSE 8050
COPY target/stream.jar stream.jar
ENTRYPOINT ["java","-jar","stream.jar"]
```

Docker-Compose

Docker Compose is a tool that was developed to help define and share multi-container applications.

So… After implementing our services and creating a docker file to create the images for each service, we are now ready to run our system and bring it to life using docker compose.

Components:

```yaml
version: "3"
services:

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

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

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

```yaml
storage:
  image: storage
  container_name: storage
  networks:
    - testnet
  ports:
    - "8060:8060"

upload:
  image: upload
  container_name: upload
  networks:
    - testnet
  ports:
    - "8080:8080"
  depends_on:
    - storage
    - authentication
    - database
```

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

volumes:
  db_data: { }

networks:
  testnet:
```