

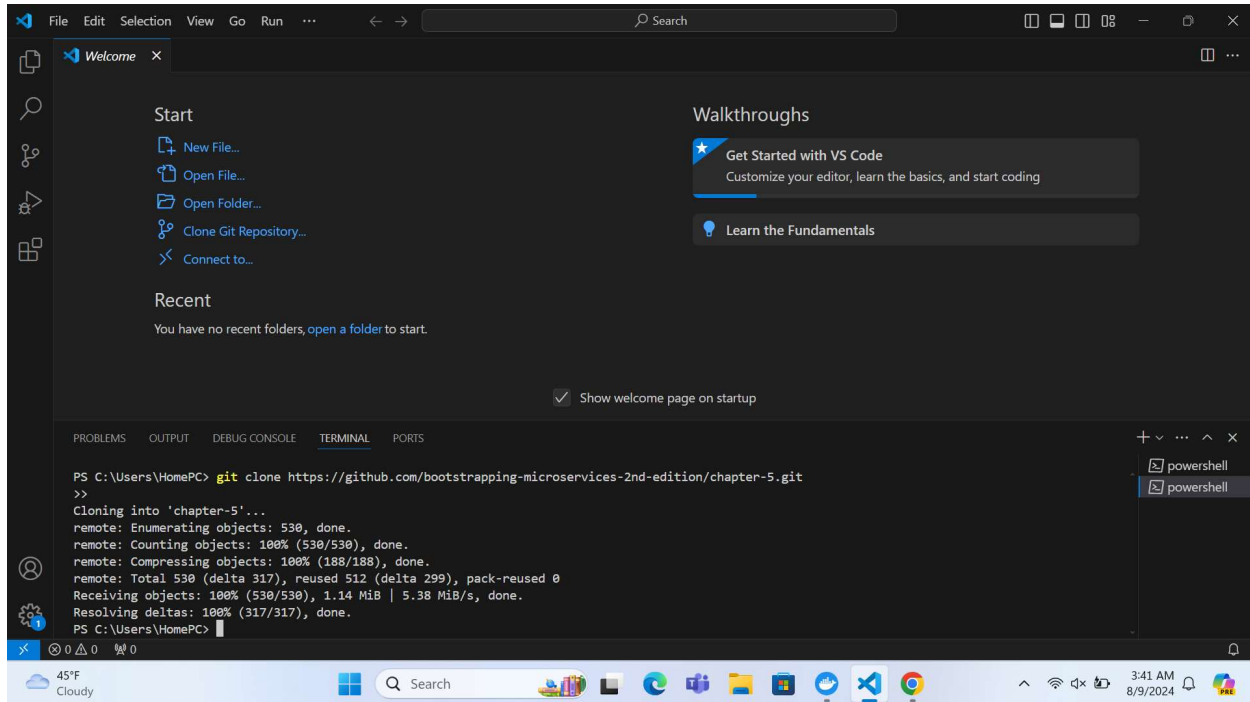
**Dennis Kimutai Kimaiyo**

**ID: S223224152**

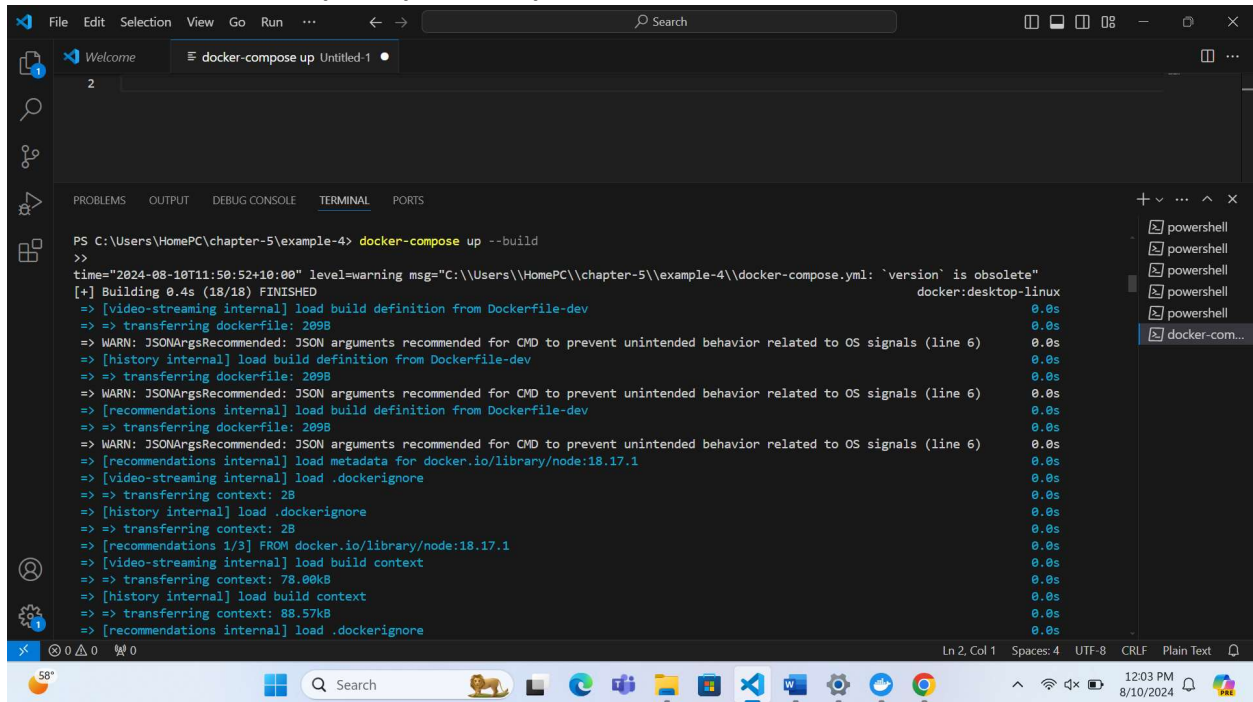
**Course: SIT722 Software Deployment and Operation**

**Unit: Task 4.1P**

**screenshot of the VS Code console showing the command and the successful cloning**

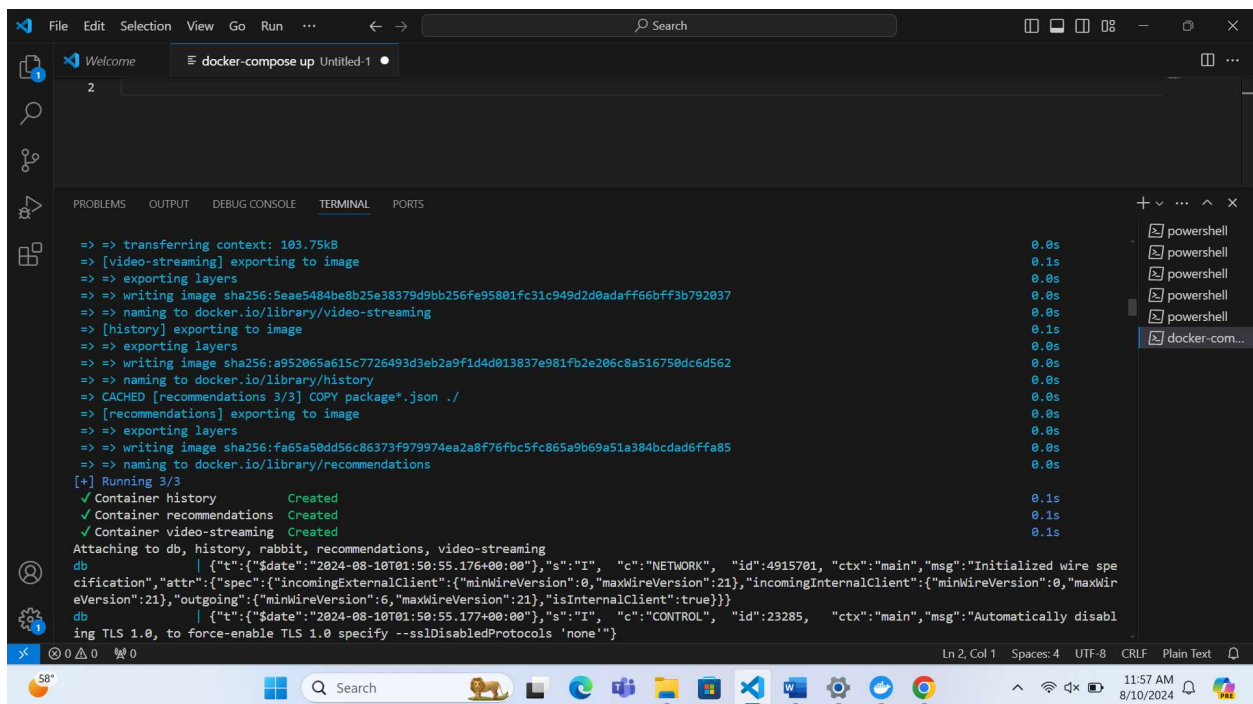


## Screenshot of docker-compose up for Example-4



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the output of the command `docker-compose up --build` executed in a PowerShell prompt at `C:\Users\HomePC\chapter-5\example-4>`. The output shows the build process for the `docker:desktop-linux` service, including loading build definitions, transferring Dockerfiles, and building images. The build process is completed successfully, and the terminal shows the following output:

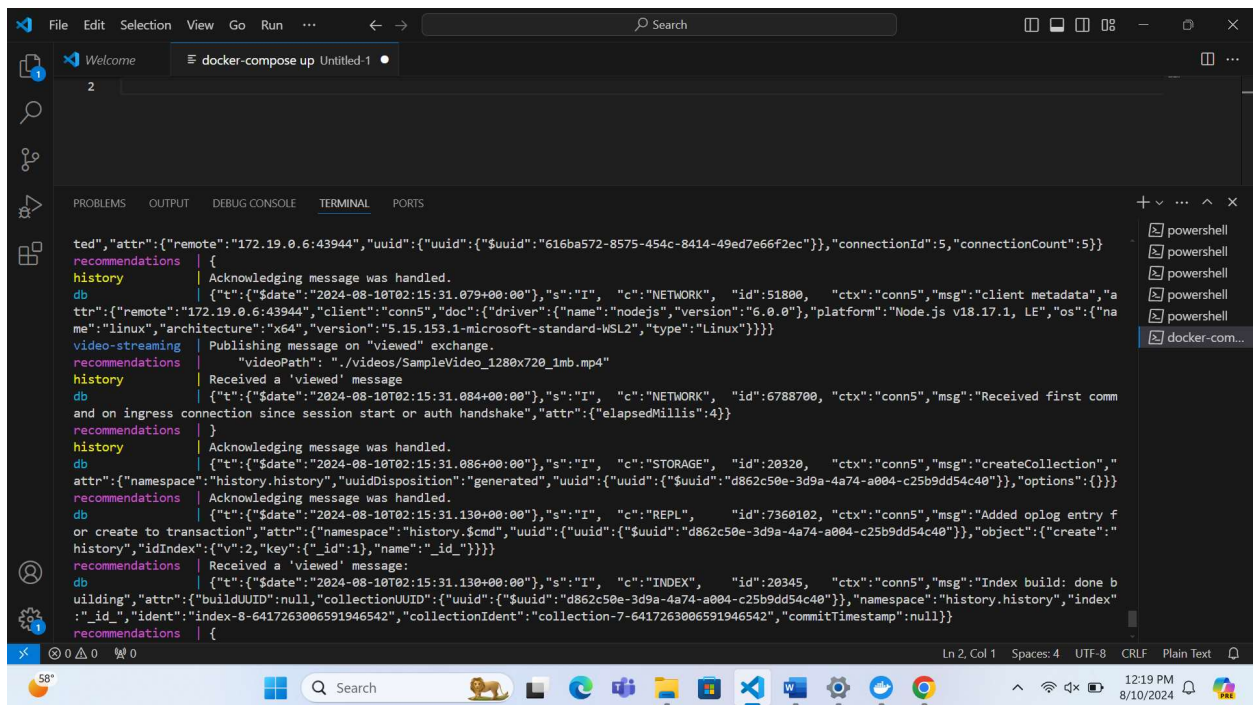
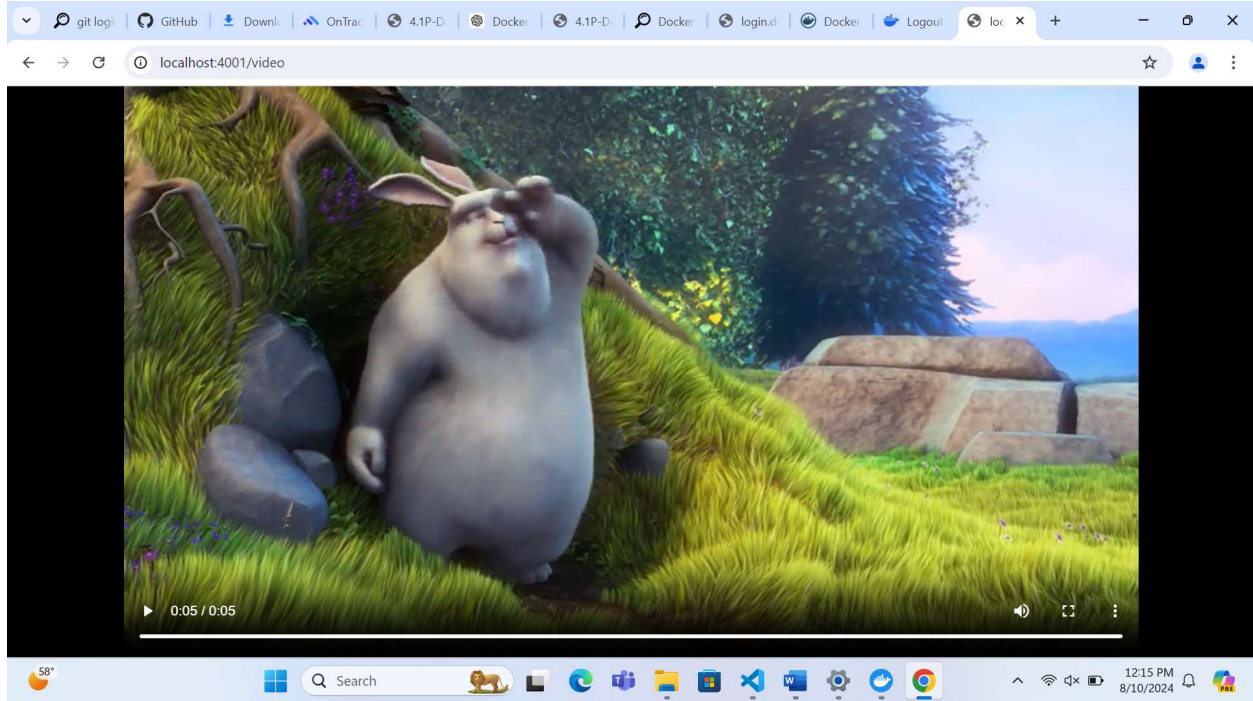
```
PS C:\Users\HomePC\chapter-5\example-4> docker-compose up --build
>>
time="2024-08-10T11:50:52+10:00" level=warning msg="C:\\Users\\HomePC\\chapter-5\\example-4\\docker-compose.yml: 'version' is obsolete"
[+] Building 0.4s (18/18) FINISHED
=> [video-streaming internal] load build definition from Dockerfile-dev
=> transferring dockerfile: 209B
=> WARN: JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 6)
=> [history internal] load build definition from Dockerfile-dev
=> transferring dockerfile: 209B
=> WARN: JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 6)
=> [recommendations internal] load build definition from Dockerfile-dev
=> transferring dockerfile: 209B
=> WARN: JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 6)
=> [recommendations internal] load metadata for docker.io/library/node:18.17.1
=> [video-streaming internal] load .dockerignore
=> transferring context: 2B
=> [history internal] load .dockerignore
=> transferring context: 2B
=> [recommendations 1/3] FROM docker.io/library/node:18.17.1
=> [video-streaming internal] load build context
=> transferring context: 78.00kB
=> [history internal] load build context
=> transferring context: 88.57kB
=> [recommendations internal] load .dockerignore
```



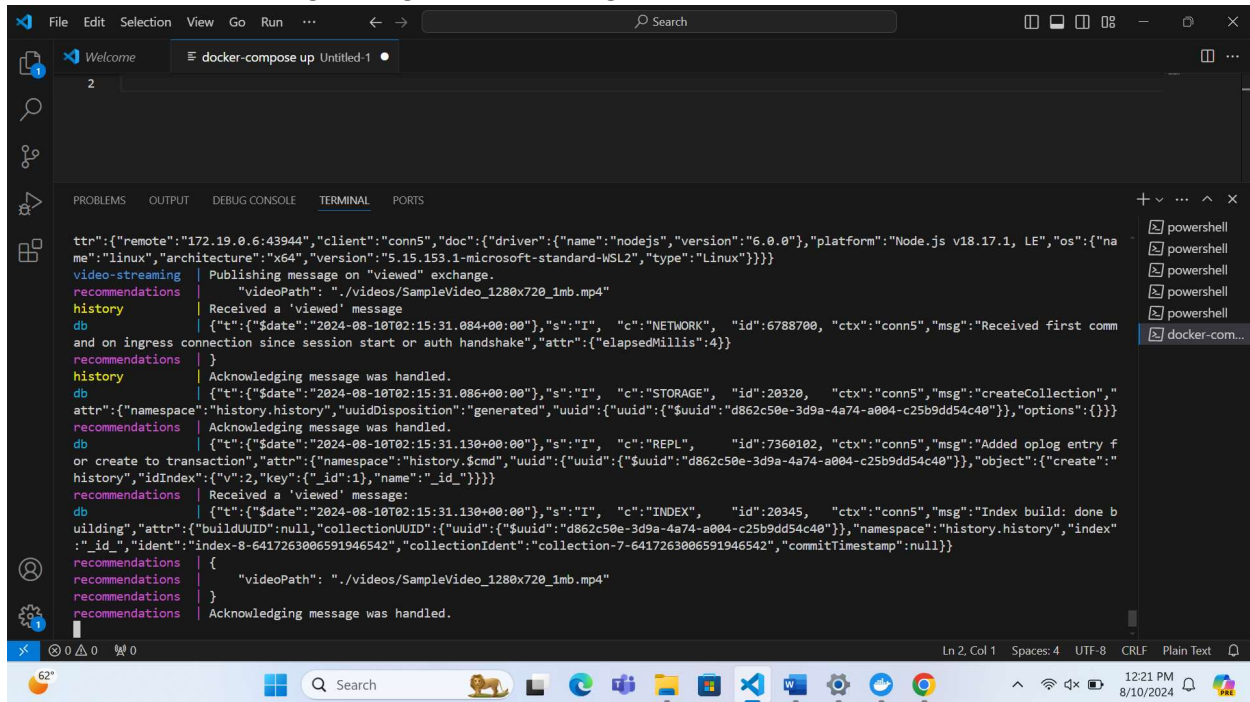
The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the output of the command `docker-compose up` executed in a PowerShell prompt at `C:\Users\HomePC\chapter-5\example-4>`. The output shows the containerization process, including exporting layers, writing images, and running containers. The containers `db`, `history`, `rabbit`, `recommendations`, and `video-streaming` are successfully created and attached to the network. The terminal shows the following output:

```
=> => transferring context: 103.75kB
=> [video-streaming] exporting to image
=> exporting layers
=> writing image sha256:5eae5484be8b25e38379d9bb256fe95801fc31c949d2d0adaff66bfb792037
=> naming to docker.io/library/video-streaming
=> [history] exporting to image
=> exporting layers
=> writing image sha256:a952065a615c7726493d3eb2a9f1d4d013837e981fb2e206c8a516750dc6d562
=> naming to docker.io/library/history
=> CACHED [recommendations 3/3] COPY package*.json ./
=> [recommendations] exporting to image
=> exporting layers
=> writing image sha256:fa65a50dd56c86373f97974ea2a8f76fbc85a9b69a51a384bcdad6ffa85
=> naming to docker.io/library/recommendations
[+] Running 3/3
✔ Container history Created
✔ Container recommendations Created
✔ Container video-streaming Created
Attaching to db, history, rabbit, recommendations, video-streaming
db | {"t":{"$date":"2024-08-10T01:50:55.176+00:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"main","msg":"Initialized wire spe
cification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":21},"incomingInternalClient":{"minWireVersion":0,"maxWir
eVersion":21},"outgoing":{"minWireVersion":6,"maxWireVersion":21},"isInternalClient":true}}}
db | {"t":{"$date":"2024-08-10T01:50:55.177+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main","msg":"Automatically disabl
ing TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
```

## A screenshot of the browser displaying the streamed video



## Screenshot of Console Logs during Video Streaming



```
ttr":{"remote":"172.19.0.6:43944","client":"conn5","doc":{"driver":{"name":"nodejs","version":"6.0.0"},"platform":"Node.js v18.17.1, LE","os":{"name":"linux","architecture":"x64","version":"5.15.153.1-microsoft-standard-WSL2","type":"Linux"}}}}
video-streaming | Publishing message on "viewed" exchange.
recommendations | "videoPath": "/videos/SampleVideo_1280x720_1mb.mp4"
history | Received a 'viewed' message
db | {"t":{"date":"2024-08-10T02:15:31.084+00:00"},"s":"I", "c":"NETWORK", "id":6788700, "ctx":"conn5","msg":"Received first comm
and on ingress connection since session start or auth handshake","attr":{"elapsedMillis":4}}
recommendations | }
history | Acknowledging message was handled.
db | {"t":{"date":"2024-08-10T02:15:31.086+00:00"},"s":"I", "c":"STORAGE", "id":20320, "ctx":"conn5","msg":"createCollection","
attr":{"namespace":"history.history","uuidDisposition":"generated","uuid":{"uuid":{"$uuid":"d862c50e-3d9a-4a74-a004-c25b9dd54c40"},"options":{}}}
recommendations | Acknowledging message was handled.
db | {"t":{"date":"2024-08-10T02:15:31.130+00:00"},"s":"I", "c":"REPL", "id":7360102, "ctx":"conn5","msg":"Added oplog entry f
or create to transaction","attr":{"namespace":"history.$cmd","uuid":{"uuid":{"$uuid":"d862c50e-3d9a-4a74-a004-c25b9dd54c40"},"object":{"create":"
history","idIndex":{"v":2,"key":{"_id":1},"name":"_id_1"}}}}
recommendations | Received a 'viewed' message:
db | {"t":{"date":"2024-08-10T02:15:31.130+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"conn5","msg":"Index build: done b
uilding","attr":{"buildUUID":null,"collectionUUID":{"uuid":{"$uuid":"d862c50e-3d9a-4a74-a004-c25b9dd54c40"},"namespace":"history.history","index"
:"_id_1","ident":"index-8-6417263006591946542","collectionIdent":"collection-7-6417263006591946542","commitTimestamp":null}}
recommendations | {
recommendations | "videoPath": "/videos/SampleVideo_1280x720_1mb.mp4"
recommendations | }
recommendations | Acknowledging message was handled.
```

## 1. Describe What Each Example Application Demonstrates

### Example 1: Direct Communication

Description: This example focuses on how two Docker containers can directly exchange data over their network connection. It shows how services can communicate with each other using basic network operations without needing an intermediary.

### Example 2: HTTP Communication

Description: In this example, Docker containers use HTTP protocols to communicate. It highlights how web services can make HTTP requests and responses to interact with each other, simulating a real-world web application environment where services talk to each other via HTTP.

### Example 3: Message Queues

Description: This example introduces a message queuing system where Docker containers send and receive messages asynchronously through a message queue. This setup demonstrates how services can operate independently and handle tasks at different times, improving scalability and decoupling of services.

### Example 4: Complex Communication

Description: This example integrates various services with a message queue to show a more complex communication scenario. It demonstrates how multiple services (video-streaming, history,

recommendations, db) interact through RabbitMQ to handle a video request, process data, and provide recommendations efficiently. It illustrates the use of a message queue in a multi-service architecture.

## **2. Describe the Sequence of Events in Example-4**

Client Request:

A client initiates a request to stream a video by navigating to <http://localhost:4001/video>.

Service Involved: video-streaming

Publish Message to RabbitMQ: The video-streaming service sends a message to RabbitMQ, indicating that a video has been requested and should be processed.

Services Involved: video-streaming and rabbit

Message Handling by History Service: The history service receives the 'viewed' message from RabbitMQ. It logs the view event and acknowledges the message.

Services Involved: history and rabbit

Request Recommendations:

After processing the view event, the recommendations service queries the db service to get video recommendations based on the requested video.

Services Involved: recommendations, db, and rabbit

Database Query:

The db service responds to the recommendations service with the list of recommended videos.

Services Involved: db and recommendations

Send Recommendations to Video-Streaming:

The recommendations service sends the recommendations back to the video-streaming service.

Services Involved: recommendations and video-streaming

Stream Video:

The video-streaming service streams the requested video to the client's browser.

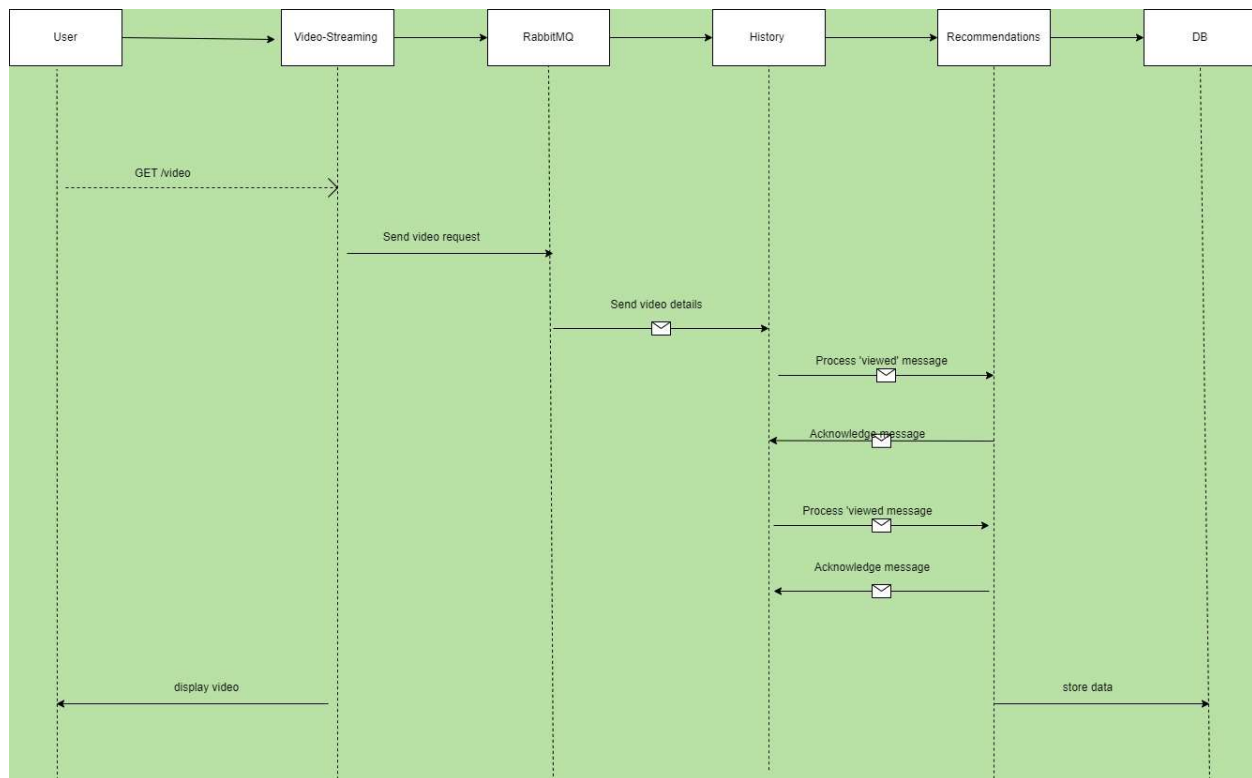
Services Involved: video-streaming

Video Plays in Browser:

The video is displayed and played in the client's browser window.

Service Involved: client





### 3. How Is the video-streaming Container Made to Wait for rabbit?

**Docker Configuration:** The docker-compose.yml file contains a depends\_on directive for the video-streaming service, which specifies that it depends on the rabbit service. This configuration ensures that the video-streaming container does not start until the rabbit container is fully initialized and running. This prevents potential issues where the video-streaming service might try to connect to RabbitMQ before it's ready.

### 4. How Are Docker-dev Dockerfiles Used in Example-4?

- **Purpose of Dockerfiles:** Dockerfiles are scripts used to build Docker images. In Example-4, each Dockerfile outlines the steps needed to prepare the environment for each service, such as installing necessary software, setting environment variables, and copying code into the container.
- **Impact on Execution:** The Dockerfiles ensure that each microservice is built with the correct configuration and dependencies, allowing each service to function as intended. They play a crucial role in setting up the services so they can communicate effectively and handle requests as part of the application's architecture