Media-Management-Platform

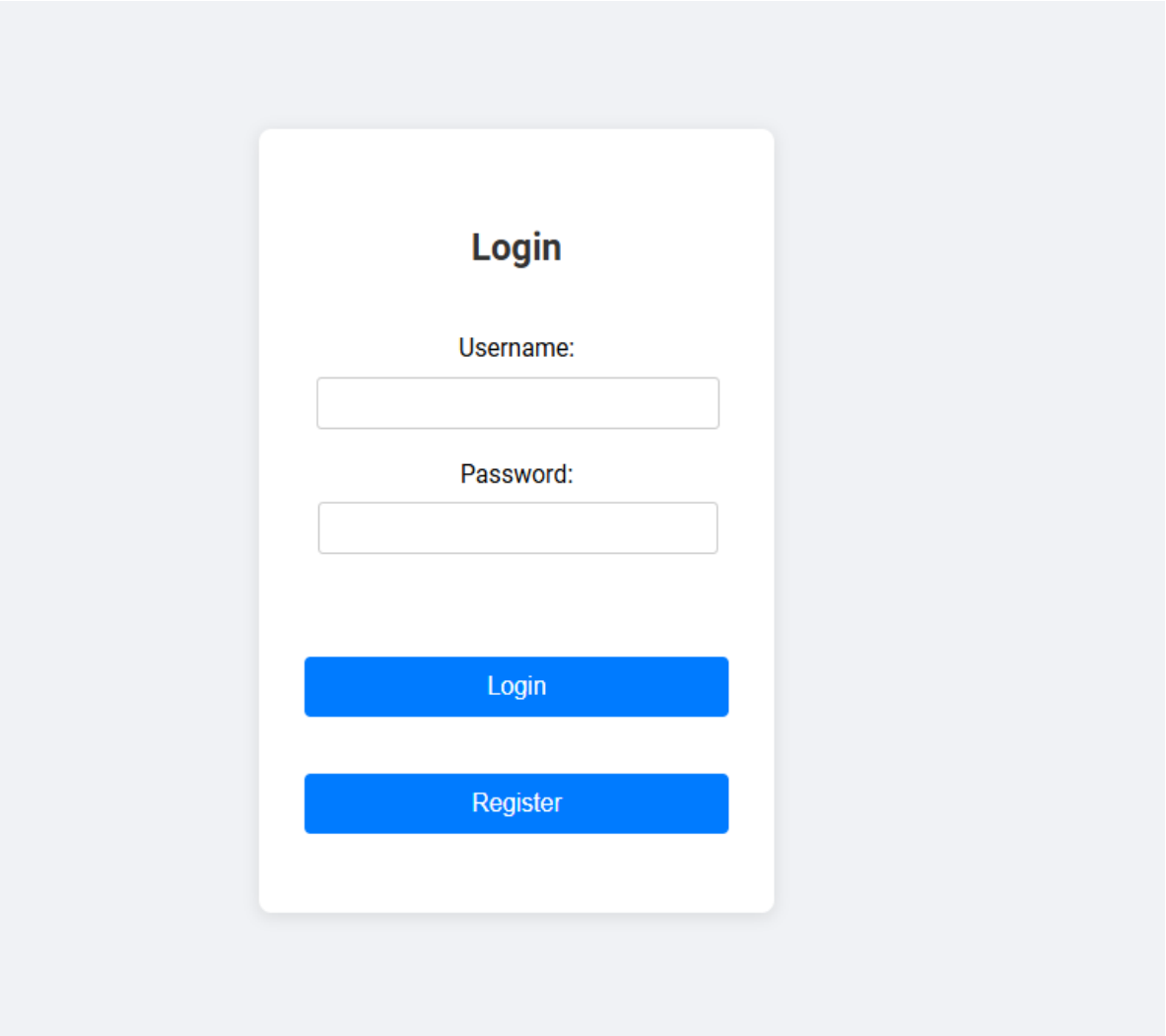
WT WS24/25

Authentication

Goals

- Require user to authenticate

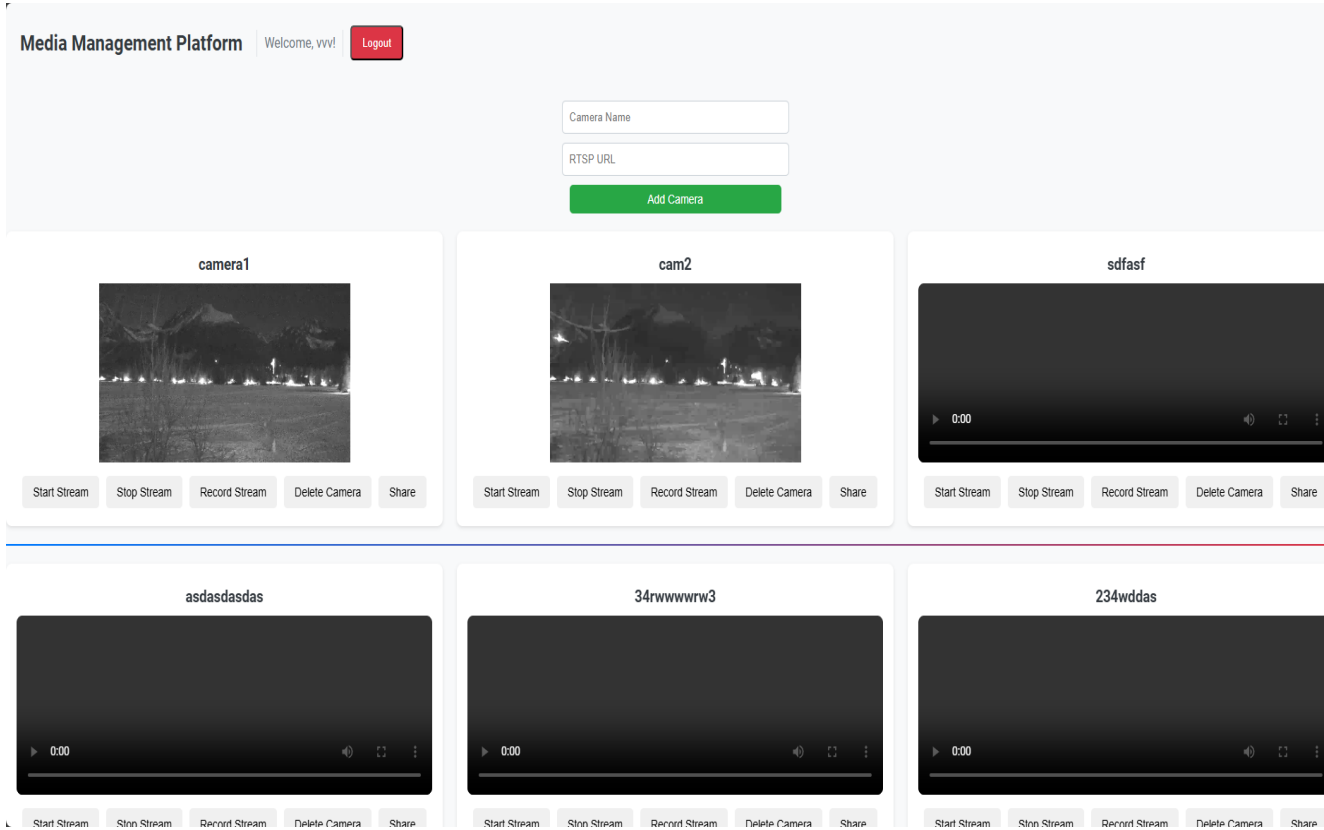
- Show different streams based on user



Media Processing

Goals

- Stream media over http
- Record the stream
- Store camera details
- User-friendly UI for end-user

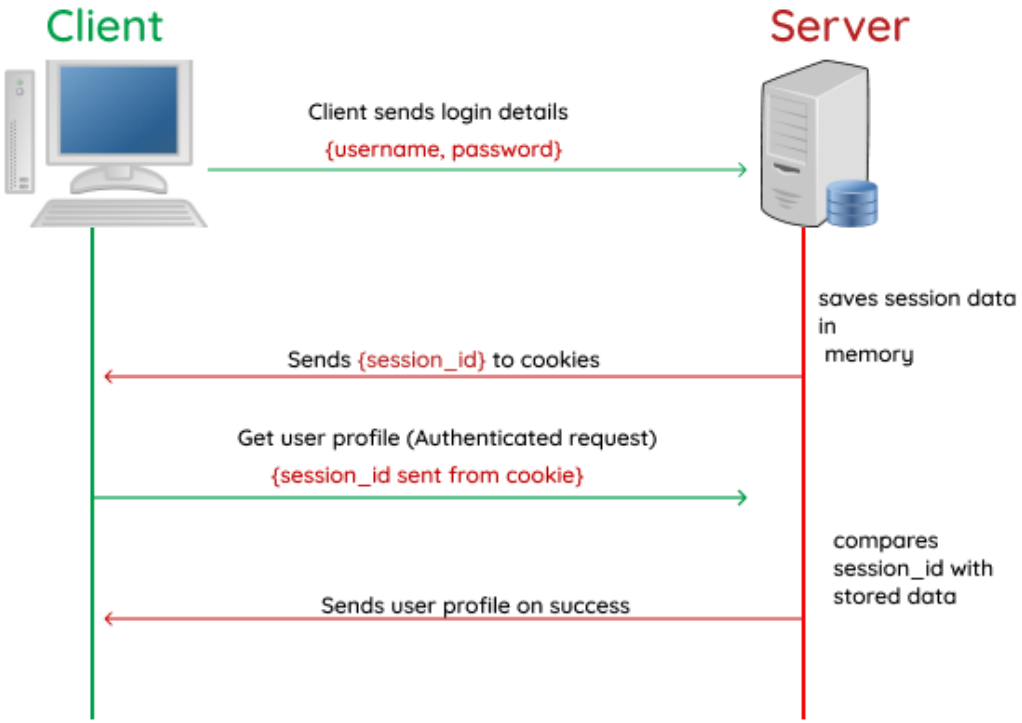


Backend

Authentication

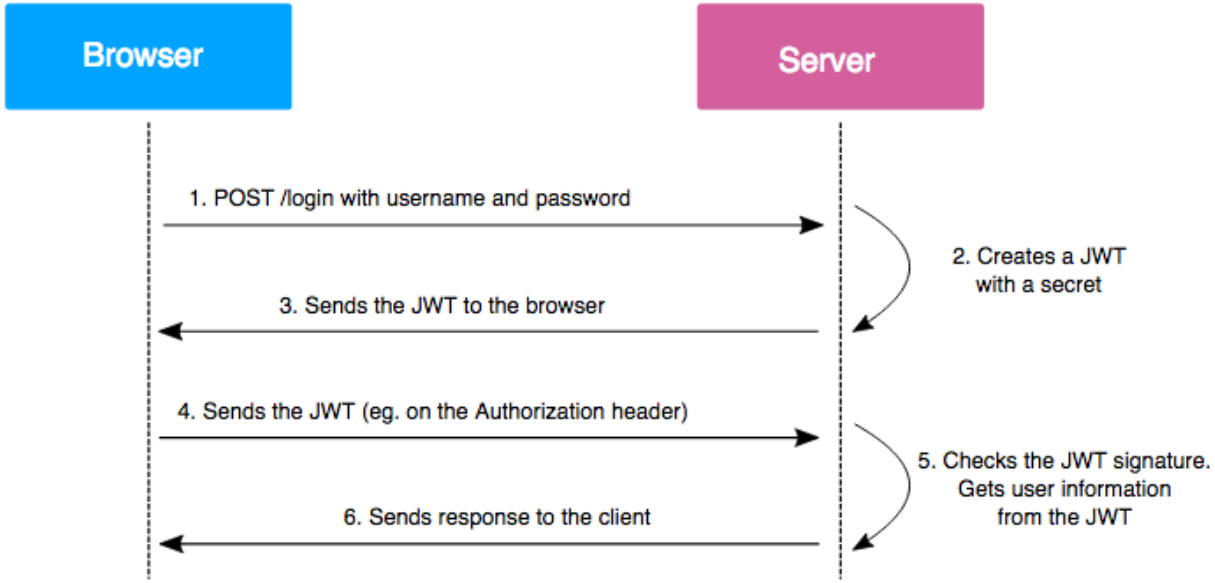
- At first: Session based

- Too complicated with frontend
  ➔ JWT



Session based authentication

JsonWebToken



Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV
CJ9.eyJkYXRhIjp7InVzZXJuYW1lIjoib
XlVc2VybmFtZSIsInVzZXJJRCI6IjU1MT
A4YmRhLTEyZjItNGJhYS1iNDhiLTdhMmM
2OGMwZTM4MSJ9LCJpYXQiOjE3MzcxODYx
NDAsImV4cCI6MTczNzI3MjU0MH0._JXqb
zg6QJ7wtWG6XnTEg7J5CPkY_2_W0LXPSW
REVx8
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```


PAYLOAD: DATA


```
{
  "data": {
    "username": "myUsername",
    "userID": "55108bda-12f2-4baa-b48b-
7a2c68c0e381"
  },
  "iat": 1737186140,
  "exp": 1737272540
}
```


VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

API

POST http://localhost:3000/register

```
1 {
2     "username": "myUsername",
3     "password": "password123"
4 }
```

200 OK

```
1 {
2     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
          eyJkYXRhIjp7InVzZXJuYW1lIjoibXlVc2VybmFtZSIsI
          2OGMwZTM4MSJ9LCJpYXQiOjE3MzcxODM5MDcsImV4cCI6
          XpE4Af6ZoxFSUM3dxeLqm2S-T36MkiUhXVmNlPe9vYQ"
3 }
```

POST http://localhost:3000/login

```
1 {
2     "username": "myUsername",
3     "password": "password123"
4 }
```

200 OK

```
1 {
2     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
          eyJkYXRhIjp7InVzZXJuYW1lIjoibXlVc2VybmFtZSIsI
          2OGMwZTM4MSJ9LCJpYXQiOjE3MzcxODM5MDcsImV4cCI6
          XpE4Af6ZoxFSUM3dxeLqm2S-T36MkiUhXVmNlPe9vYQ"
3 }
```

GET http://localhost:3000/self

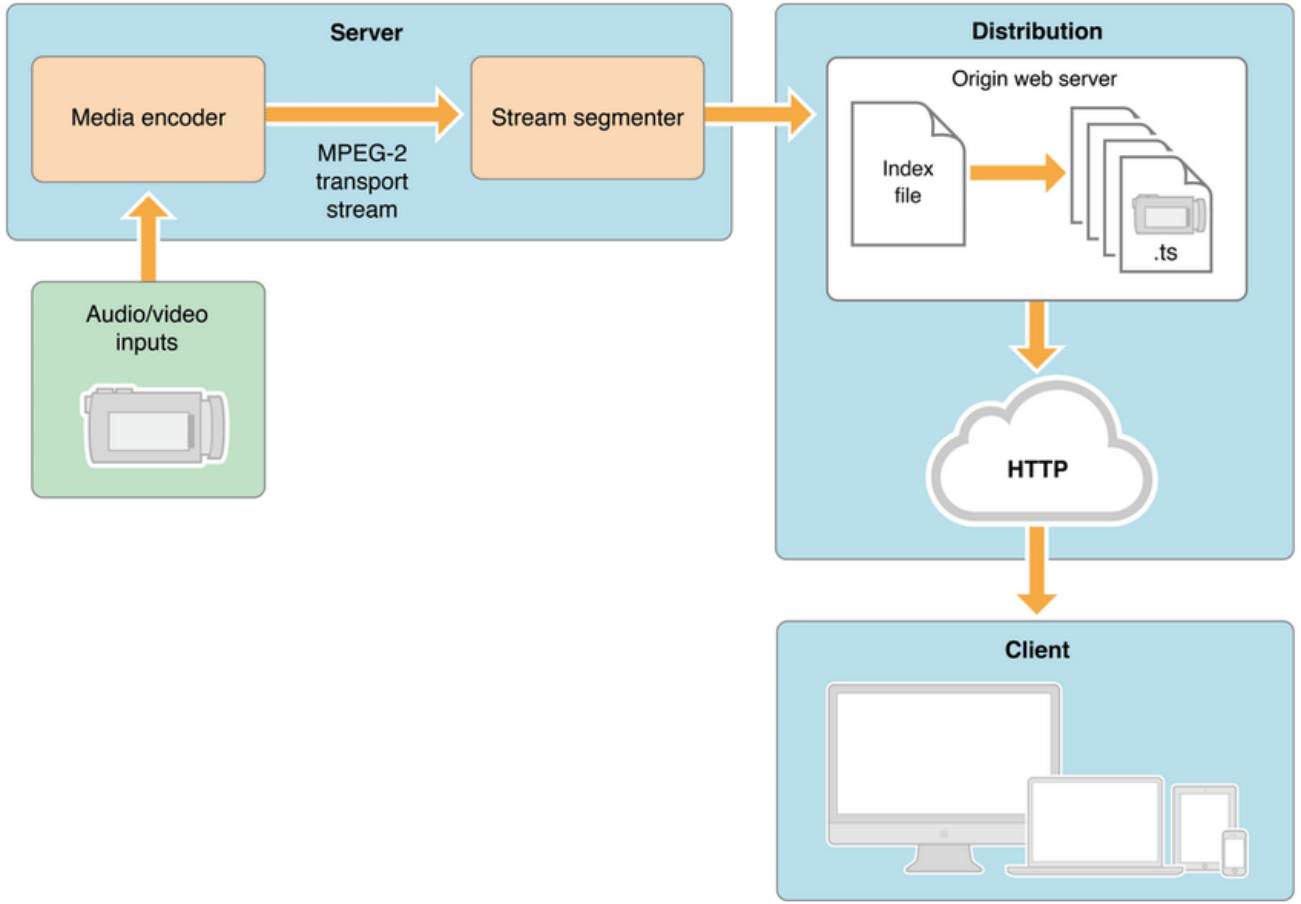
200 OK

401 Unauthorized

Access streams from different cameras

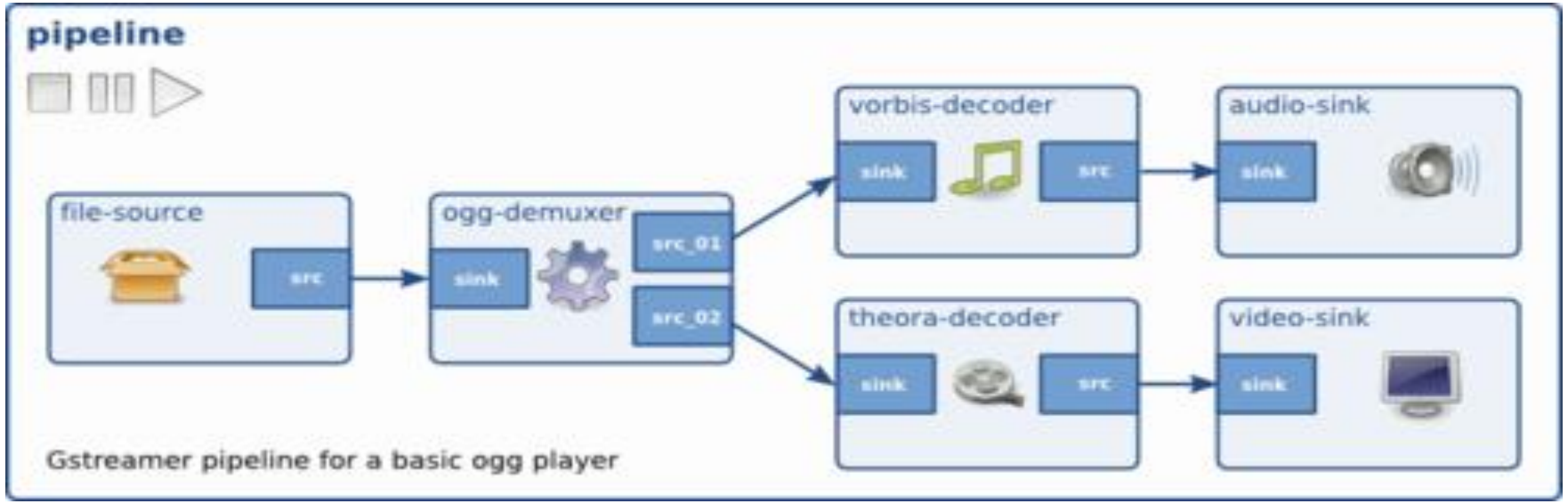
- RTSP – Widely used protocol by manufacturers
- Gstreamer – A robust media processing framework
- HLS – A apple made media format for live streaming on html
    - Have playlists
    - Have segments
    - shareable stream bucket
    - prebuilt buffering
    - supported by almost all .js based video players

HLS

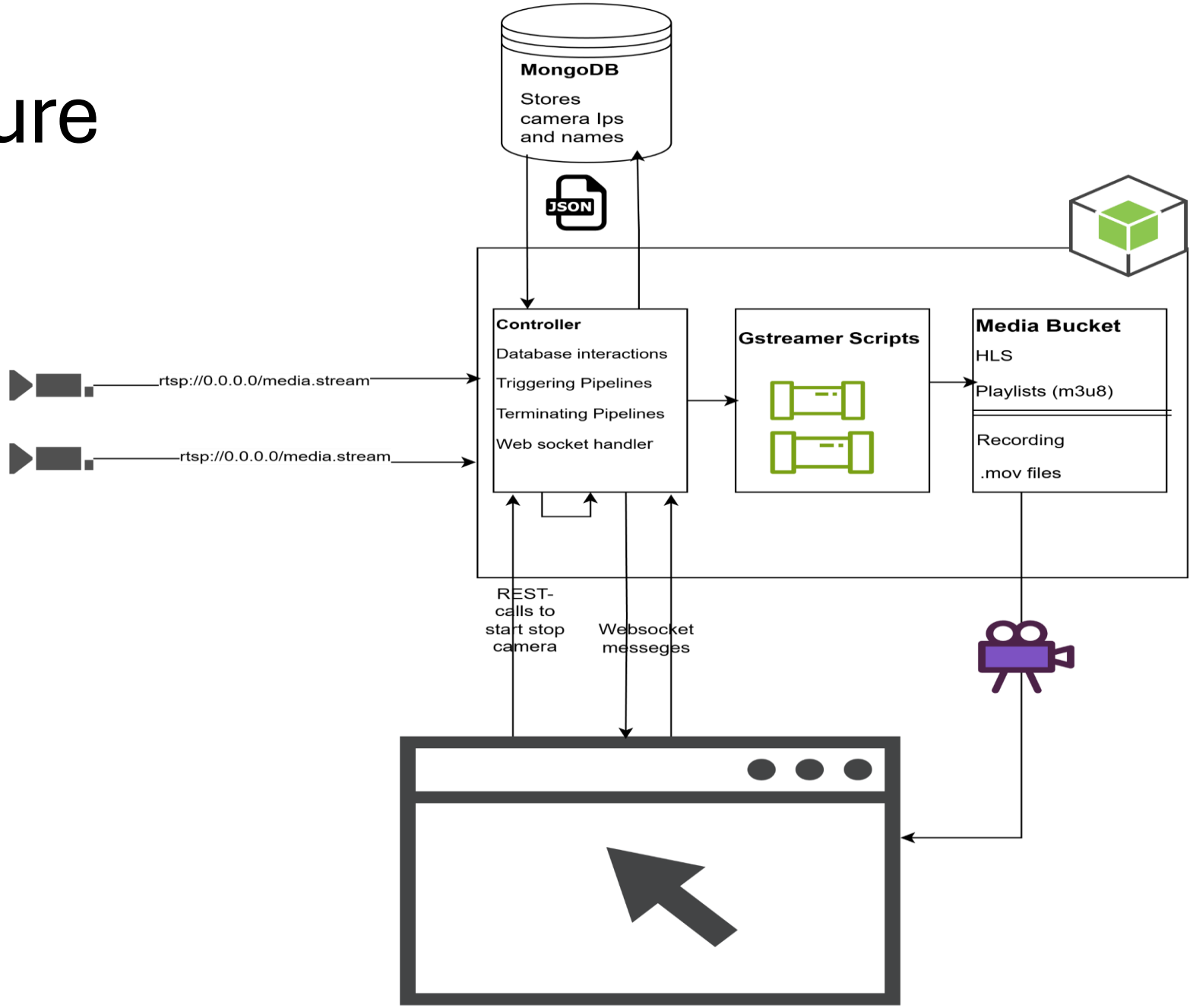

Server
Media encoder
MPEG-2
transport
stream
Stream segmenter
Audio/video
inputs
Distribution
Origin web server
Index
file
.ts
HTTP
Client

Gstreamer

- Pipelines
- Media Elements
- Caps
- Sink-point



Full Architechture



Frontend

Index.js

- AuthProvider: Attempts to authenticate user
- Router: Displays different elements depending on Route

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <AuthProvider>
      <Router>
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route path="/" element={<App />} />
          <Route path="/register" element={<Register />} />
        </Routes>
      </Router>
    </AuthProvider>
  </React.StrictMode>
);
```

AuthContext.js

- Verifies if token in storage is valid
- Sets authenticated state

```
const [isAuthenticated, setIsAuthenticated] = useState(false);
const [isLoading, setIsLoading] = useState(true);

useEffect(() => {
    const token = localStorage.getItem('token');
    console.log("AuthCon: Token detected");

    // Verify if Token in storage is valid
    fetch('http://localhost:3000/self', {
        headers: { authorization: `bearer ${token}` }
    })
        .then((res) => {
            if (res.status === 200) {
                setIsAuthenticated(true);
            } else {
                setIsAuthenticated(false);
            }
        })
        .catch(() => {
            setIsAuthenticated(false);
            console.log("Catch triggered");
        })
        .finally(() => {
            setIsLoading(false);
```

Login.js

- Attempts to log user in
- Display error message if unsuccessful

```
const handleLogin = async (e) => {
    e.preventDefault();
    setError('');
    const username = e.target.username.value;
    const password = e.target.password.value;
    // Send a POST request to the server with the login credentials
    const res = await fetch('http://localhost:3000/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
    });

    if (res.status === 200) {
        // If login is successful, store the token in local storage,
        const { token } = await res.json();
        localStorage.setItem('token', token);
        setIsAuthenticated(true);
        navigate('/');
    } else {
        // If login is unsuccessful, display an error message
        const data = await res.json();
        setError(data.message || 'Invalid credentials.');
```

App.js

```
useEffect(() => {
    // Redirect to /login if user is not authenticated after loading
    if (!isLoading && !isAuthenticated) {
        console.log('User is not authenticated, redirecting to /login.');
        navigate('/login');
    }
}, [isAuthenticated, isLoading, navigate]);
```

- Wait for AuthContext
- Redirect if not authenticated

```
if (isLoading)
    return (
        <div>Loading...</div>
    );
```

App.js socket

- Use Token in header
- Use sockets normally

```
socket = io(process.env.REACT_APP_API_URL || 'http://localhost:3000', {
    extraHeaders: {
        Authorization: `Bearer ${token}`
    }
});
```

```
socket.emit('shareCamera', { cameraId: cameraId.toString(), username }, (response) => {
    if (response.success) {
        console.log(`Camera ${cameraId} shared with ${username}`);
        setShareCameraId(null);
        setShareUsername('');
        alert(`Camera shared with ${username} successfully!`);
    } else {
```

Server.js socket

- Authenticate using passport
- Use token payload

```
io.engine.use((req, res, next) => {
  const isHandshake = req._query.sid === undefined;
  if (isHandshake) {
    passport.authenticate("jwt", { session: false })(req, res, next);
  } else {
    next();
  }

socket.on('shareCamera', async ({ cameraId, username }, callback) => {
  const userID = getUserIDFromSocket(socket);
  const camera = await Camera.findOne({ _id: cameraId, userID });
  if (!camera) return callback({ error: 'Camera not found or not authorized' });

  const userToShareWith = await findUser(username);
  if (!userToShareWith) return callback({ error: 'User to share with not found' });
    const newCamera = new Camera({ name: decodedToken.data.username+'sCamera', rtspUrl

    await newCamera.save();
  callback({ success: true });
});

function getUserIDFromSocket(socket) {
  const token = socket.handshake.headers.authorization.split(' ')[1];
  const decodedToken = jwt.decode(token);
  return decodedToken.data.userID;
}
```

App.css

- Designed a **responsive grid layout** for displaying camera consoles
- Unified the styling of **Start, Stop, Record, Delete, and Share buttons**
- a **loading spinner** for better user feedback when streams or dashboards are being initialized

App.js HLS player

- Open source video player with simple setup

```
import Hls from 'hls.js';
1 usages  studbaneso1322
const setupHlsPlayer = (cameraId, streamUrl) => {
    const videoElement = videoRefs.current[cameraId];

    // Ensure video element is available
    if (!videoElement) {
        console.error(`Video element for camera ${cameraId} is not available.`);
        return;
    }

    if (Hls.isSupported()) {
        const hls = new Hls();
        hls.loadSource(streamUrl);
        hls.attachMedia(videoElement);
        hls.on(Hls.Events.MANIFEST_PARSED, () => {
            if (videoElement) {
                videoElement.play().catch((err) => {
                    console.error('Error while trying to play the video:', err);
                });
            }
        });

        hls.on(Hls.Events.ERROR, (_, data) => {
            console.error('HLS error:', data);
        });
    } else if (videoElement.canPlayType( type: 'application/vnd.apple.mpegurl')) {
        videoElement.src = streamUrl;
        videoElement.addEventListener('loadedmetadata', () => {
            videoElement
                .play()
                .catch((err) => console.error('Error while trying to play the video:', err));
        });
    }
```

Deployment

Docker

- Docker images for easy-deployment ☺

```yaml
mongo:
  image: mongo:4.2
  ports:
    - "27017:27017"
  volumes:
    - mongo-data:/data/db
  networks:
    - app-network

mysql:
  image: mysql:5.7
  ports:
    - "3306:3306"
  environment:
    MYSQL_ROOT_PASSWORD: yourpassword
    MYSQL_DATABASE: user_management
  volumes:
    - mysql-data:/var/lib/mysql
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "-pyourpassword"]
    interval: 10s
    timeout: 5s
    retries: 5
  networks:
    - app-network
```

```yaml
frontend:
  build:
    context: ./frontend
  container_name: react_frontend
  ports:
    - "80:80"
  environment:
    - REACT_APP_API_URL=http://localhost:3000
  networks:
    - app-network
app:
  build:
    context: ./backend
  ports:
    - "3000:3000"
  environment:
    - MONGO_URI=mongodb://mongo:27017/mediasoup
    - MYSQL_HOST=mysql
    - MYSQL_PORT=3306
    - MYSQL_USER=root
    - MYSQL_PASSWORD=yourpassword
    - MYSQL_DATABASE=user_management
    - STREAM_BASE_URL=http://localhost:3000
    - STREAM_VOLUME_PATH=/video-storage
    - JWT_SECRET=s3cUr3!t0k3n#s3cr3t@1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
  volumes:
    - video-storage:/video-storage
  depends_on:
    mysql:
      condition: service_healthy
  networks:
    - app-network
```

Demo

Image sources:

- http live streaming frame work diagram - Search Images
- Introduction to network streaming using GStreamer - RidgeRun Developer Wiki
- Apple HTTP Live Streaming (Apple HLS)
- GStreamer C++ - Stream Webcam over TCP Tutorial - DEV Community