# Web Surveillance System

Aarón Riveiro Vilar
Filip Prodovic

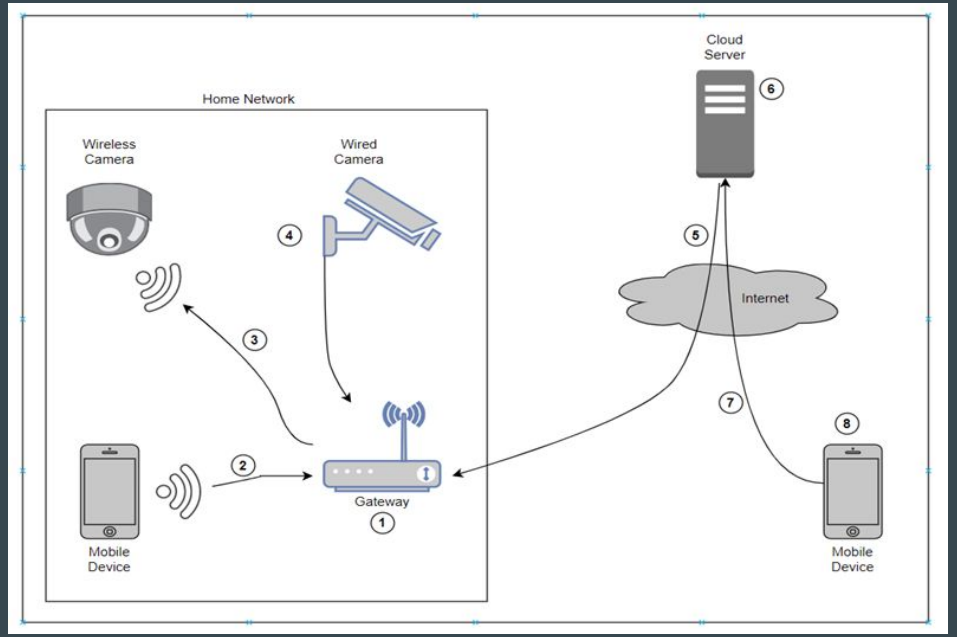
# Index

# Introduction

- **Objective:** develop a real-time web-based video surveillance system with an intuitive user interface

- **Key features:**
  - User-centric design
  - Flexible stream handling
  - Web-based
  - Multi-view support

- **Ideal use case:** an IoT environment (home, office, factory...) with multiple IP cameras connected to the local network



# Current Features

- **User management**
  - Secure login and registration system with password hashing
  - Session persistence per browser window
- **Stream management**
  - Ability to add new streams dynamically
  - Streams are associated with an user and stored in the database
  - Automatic fetching and display of registered streams upon login
- **Database integration**
  - SQLite database to manage users and their associated streams
  - Passwords are not stored in plain text, but their hashed
- **Video playback**
  - Multi-stream support with a dynamic grid layout
  - HLS integration for video reproduction
- **Frontend functionality**
  - Clean, responsive and intuitive user interface
  - Multiple screens for registering, login and stream visualization
- **Backend processing**
  - Integration of FFmpeg to handle video streams
  - Automatic start of FFmpeg process for new streams, or upon login

# Technologies - Backend

- **Node.js**
  - A powerful JavaScript runtime
  - Non-blocking I/O for high efficiency, enabling smooth handling of video streams
  - Large community and ecosystem with modules that simplify complex tasks
  - Ideal for building scalable and real-time applications

- **Express.js**
  - Minimalist web framework for Node.js
  - Lightweight and flexible for building RESTful APIs
  - Handles routing and middleware integration effortlessly
  - Simplifies backend logic for authentication, stream management, and data handling

# Technologies - Backend

- **SQLite**
  - A lightweight, file-based database management system, integrated in a Node.js module (better-sqlite)
  - Perfect for small to medium-sized applications
  - Does not require a separate server, making deployment straightforward
  - Ensures efficient storage and retrieval of user data and stream associations

- **Crypto**
  - A Node.js built-in library for cryptographic functions
  - Secure password hashing with scrypt ensures robust protection
  - Implements industry-standard encryption techniques without external dependencies
  - Ensures security for sensitive operations like user authentication

# Technologies - Frontend

- **HTML**
  - Provides the basic structure of the web application
  - Used for creating forms and embedding video elements

- **CSS**
  - Enables a responsive and visually appealing design
  - Used to customize grids, forms, buttons...
  - Allows to have a dynamic grid


- **JavaScript**
  - Core scripting language, used for multiple purposes
  - Handles events such as form submissions and button clicks
  - Manages video streams and HLS integration

- **HLS.js**
  - JavaScript library that implements an HTTP Live Streaming client
  - HLS is an HTTP-based adaptive bitrate streaming communication protocol develop by Apple
  - Ensures compatibility with most browsers, even those without native HLS support
  - Handles stream playback seamlessly

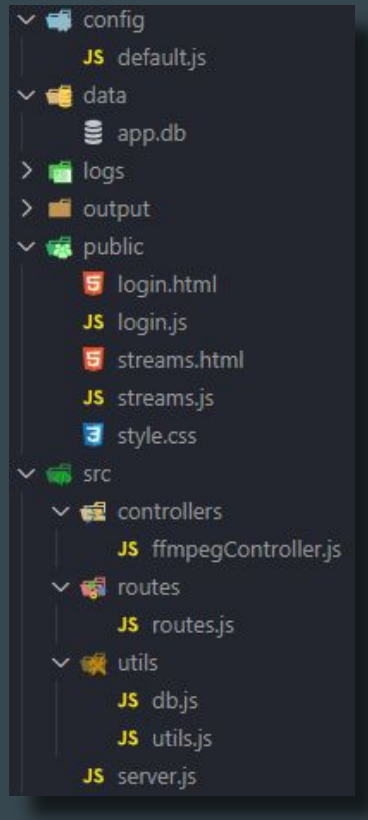
# Technologies - Other

- **FFmpeg**
  - Open-source software to handle multimedia data
  - Used for processing, transcoding, and streaming video
  - Enables live streaming and conversion of video formats
  - Highly efficient and widely used for multimedia application
  - Offers versatile options to modify streams


- **HTTP**
  - Core protocol for communication between the client and the server

# Project Structure

- **config:** contains configuration files, with parameters such as files routes, FFmpeg options, server port...
- **data:** contains the SQLite database
- **logs:** stores logs generated by each FFmpeg process
- **output:** contains FFmpeg output files, such as TS packets
- **public:** contains static frontend files
  - **login.html:** HTML for the login/register screen
  - **streams.html:** HTML for the streaming visualization screen
  - **login.js:** scripts to handle the login screen
  - **streams.js:** scripts to handle the visualization screen
- **src:** contains all backend files
  - **ffmpegController:** script to manage FFmpeg video processing
  - **routes.js:** script that defines API endpoints and backend routing
  - **utils.js:** contains multiple useful functions
  - **db.js:** script to handle database and tables creation
  - **server.js:** the entry point of the backend application

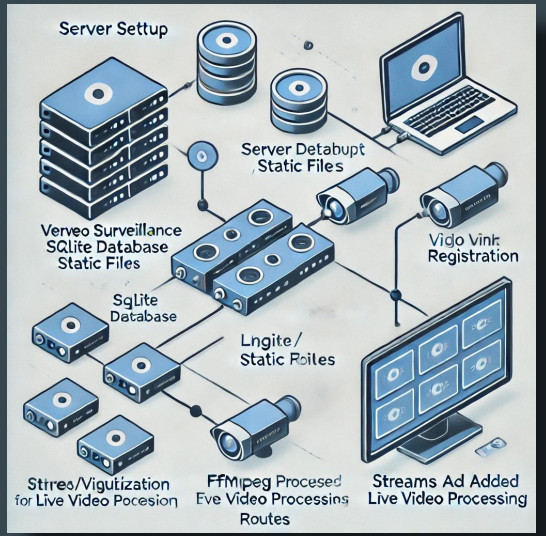


# Program Functionality

- **Server setup**
  - The server is started, initializing all necessary components (routes, configurations...)
  - The SQLite database is created or connected, ensuring tables for users and streams exist
  - Static files (HTML, CSS, JS) are served for the client-side interface

- **Login/registration**
  - After navigating to the server's URL, the login/registration page is displayed
  - New users can register by providing a username and password
  - Existing users can log in to access their saves streams
  - Successful authentication redirects users to the main video visualization page

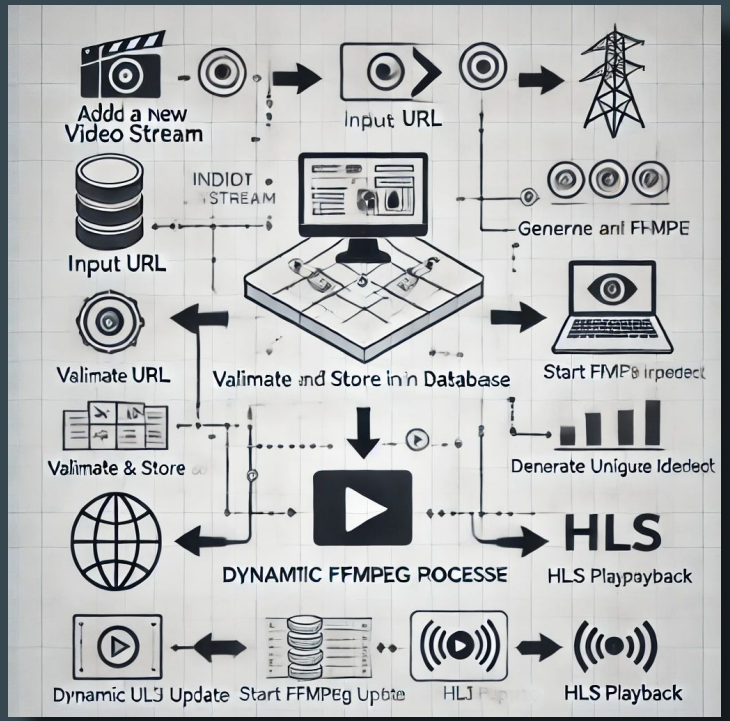
- **Video visualization**
  - The system checks the database for streams associated with the logged-in user
  - If streams are found, video players are dynamically added to the page, and FFmpeg processes are started for each stream
  - A text input is displayed for the user to add a new video source



# Program Functionality

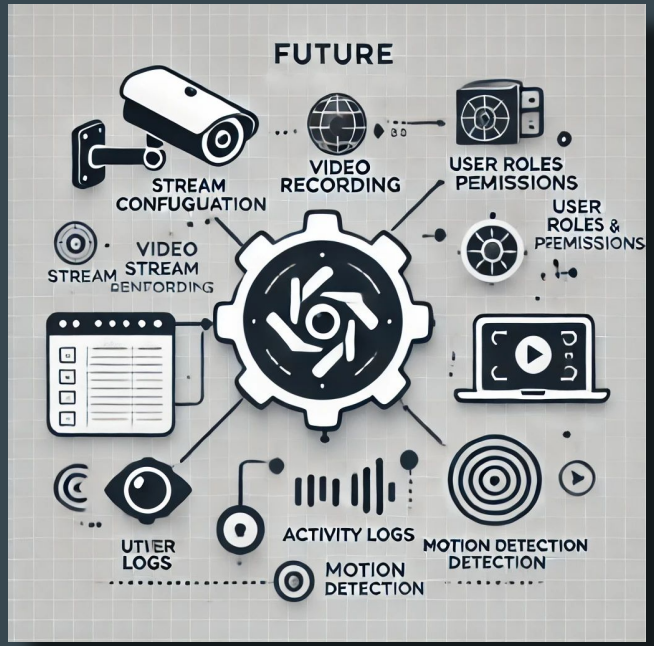
- **Adding a new stream**
  - After the user provides a video source URL, it gets sent to the server
  - The URL is validated and associated with the user in the database
  - A unique identifier is generated for the stream
  - A FFmpeg process is started to handle video transcoding and streaming
  - The page is dynamically updated to include a new video player for the added stream

- **Real-time playback**
  - The video player uses the library HLS.js to fetch the video chunks processed by FFmpeg
  - As more streams are added, the layout adjusts automatically for optimal viewing
  - Only streams requested by the user are processed and displayed, ensuring scalability



# Future Features

- **Stream configuration**
  - Customizable streams, giving users the ability to configure each stream individually, modifying many FFmpeg parameters
- **Video recording**
  - Ability to record the last X seconds or minutes of any stream
  - Option to save or download those recorded segments
- **User roles and permissions**
  - **User:** standard role with access to their own streams
  - **Admin:** elevated privileges with the ability to view and manage all users' streams
  - **Guest:** role with temporary access to specific streams
- **Activity logs**
  - Maintain detailed logs of user activity, including who accessed each stream and when
- **Motion detection**
  - Implement motion detection to alert users when activity is detected in a stream
  - Save detected motion events for later review or analysis



# Questions?