

Architecture Design, TI2806

Blazin and the Goons

Authors:

| | |
|-------------------|---------|
| Shane Koppers, | 4359062 |
| Floris List, | 4380258 |
| Hidde Lycklama, | 4397614 |
| Joris Oudejans, | 4393694 |
| Jordy Weijgertse, | 4247124 |

Contents

[1 Introduction](#)

[2 Software architecture views](#)

[2.1 Subsystem decomposition \(sub-systems and dependencies between them\)](#)

[2.2 Hardware/software mapping \(mapping of sub-systems to processes and computers, communication between computers\).](#)

[2.3 Persistent data management \(file/ database, database design\)](#)

[2.4 Concurrency \(processes, shared resources, communication between processes, deadlocks prevention\)](#)

[3 Glossary](#)

1 Introduction

This document describes the technical design of the application made by “Blazin and the Goons” for PolyCast to manage live musical video recordings. It provides functionality for operators to manage scripts and be in sync during the recording.

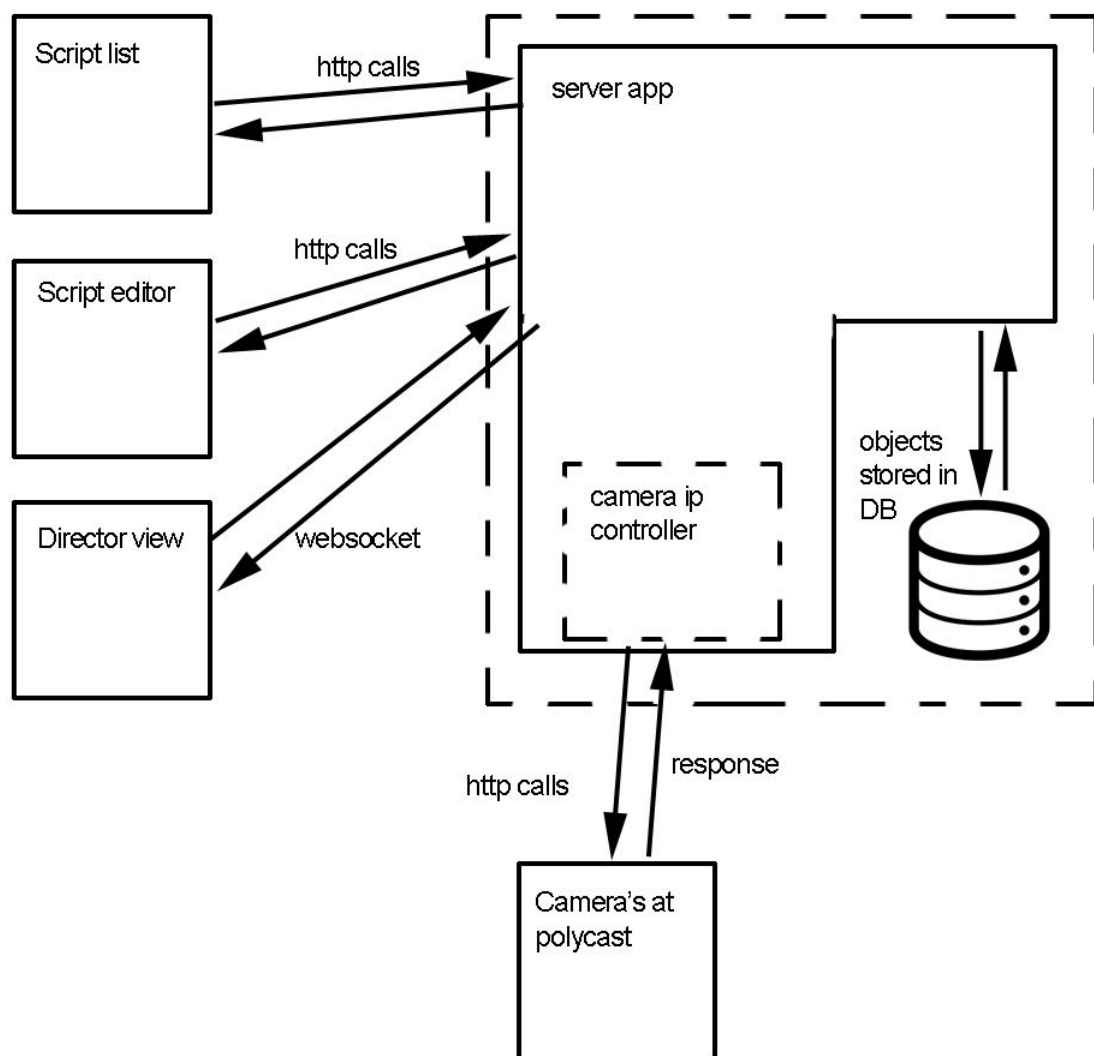
1.1 Design goals

- **Easy of use**
The system has to be fast and easy to use for all operators, because time is limited during a recording. Operators should not have to spent time using the system that was intended to make their live easier.
- **Performance**
Since we need to improve the workflow of the camera crew performance is of utmost importance. The system needs to work fast with as little delay as possible. Too much delay can lead to problems when cameras are not in position when they need to be. The system must therefore be realtime, there is no room for noticeable latency.
- **Reliability**
The system is being designed to lift pressure and workload off of the camera crew. If the cameras are not in the right preset at the right time or some other malfunction happens, the crew can't trust the system, especially not in a live environment. Especially in a live environment where multiple concurrent users are operating the system, reliability is even more important.
- **Security**
If the server is not going to be local it should be unable for outsiders to go in the system and control the cameras or change the presets for obvious reasons.

2 Software architecture views

This chapter will explain the architecture of the system. The first paragraph will show the system as a set of subsystems and how they interact with each other. The second paragraph will explain the relation between the hardware and the software of the system. The third paragraph shows how the system manages its data. Lastly, the fourth paragraph explains how the system stays concurrent and prevents unwanted interactions.

2.1 Subsystem decomposition



The system can be divided in different sections.

- **Client**
 - **AngularJS 2**

The client-side framework that supports responsive updating of the DOM after a data change. All views are based upon this framework.
 - **HTML Views**

The HTML partials for the different interface components.
- **Server**
 - **Play Framework**

The Java framework that provides the HTTP server, back-end logic and serves basic views. The structure of the system is following the MVC model. Controllers serve data from the models in views.
 - **Views**

There are three main views as of now. The script list view shows the available scripts to run. The script editor is the view responsible for creating and editing scripts before actually running them. The main view is the director view, which contains all aspects that are important during the recording of an event. This view is divided in a couple of subviews, namely the running script, the available presets, the livestream with scheduled next camera's and a dock that contains action buttons and a script timeline.
 - **Database**

The database is used to store the scripts, cameras, actions and presets. Currently it consists of tables to represent the following objects

 - Scripts
 - Actions - actions in a script
 - Presets - presets for cameras
 - ActiveScript - the current active script together with its position and running time.

2.2 Hardware/software mapping

The server

2.3 Persistent data management

Data of scripts, presets, actions and states is stored in a relational database system. Currently the Postgres DBMS is used, but it is very easy to switch systems as the data storage is implemented using an abstract DBMS driver together with the Ebean Object Relational Mapper. This way POJO (Plain Old Java Objects) can be easily stored into the DBMS.

For local development, an in-memory database is used. This provides good flexibility and makes sure no bugs occur that arise from the (lack of the) state in the database, because every new run the database is refreshed.

There is an online version of the system which interacts with a Postgres database. This instance can be found at <http://blazinandthegoons.herokuapp.com>

2.4 Concurrency

The application is designed to work with multiple clients operating concurrently.

Multiple clients can interact with the server in a very short time which creates an opportunity for race conditions. Elaborate tests have been developed to simulate the interaction of clients with the server. Included with this are tests that verify the right behaviour when multiple users are interacting together.

3 Glossary

Client - A remote browser that runs the app and is connected to the server.

Heroku - Service to deploy application online.

Postgres - Database system.

Server - Mainly the brain of the structure, the server synchronizes the views on every client.

View - A way the program represents itself through a user interface/web page.

VPN - Short for Virtual Private Network which can be used for secure communication between parties.