Architecture and Design

For the Trainboard firmware

Based on and adapted from arc42 template https://arc42.org/overview

Introduction and Goals

This document describes the software running on the ESP32 SoC, which is placed on the PCB representing the train lines from the SBB using RGB LEDs. The SoC drives the LEDs and retrieves live information about the trains from a web server.
The goal of the software is to display the live position of the trains in a user friendly and visually appealing fashion.

Requirements

Id	Description
R1	Retrieve LED status from a web server once a minute: - On/Off - Colour
R2	Adapt the LED intensity to the actual light conditions.
R3	Allow the user to connect the board to a Wi-Fi access point.
R4	Play the last 45 received frames continuously upon button press. Switch back to normal mode when the button is pressed again.
R5	Show preloaded traffic when connection to Wi-Fi or server is lost.

Quality Goals

Id	Description
Q1	Ship a robust and tested software
Q2	Ensure minimum cyber security standards are met and user privacy is respected.
Q3	Define interfaces between software layers for portability (different hardwares) and flexibility (different software stacks or frameworks, e.g. Arduino, Espressif SDK, etc.).

Stakeholders

Role/Name	Expectations
Customer	Easy to setup board, which works reliably. No threat to the home network.
SBB (on hold)	State-of-the-art IoT product compliant with best practices regarding cyber security and their branding.
Lenny	Well designed and innovative product that gives an amazing user experience
Ueli	<Todo>
Emile	State-of-the-art software, easy to maintain and reuse for other boards.

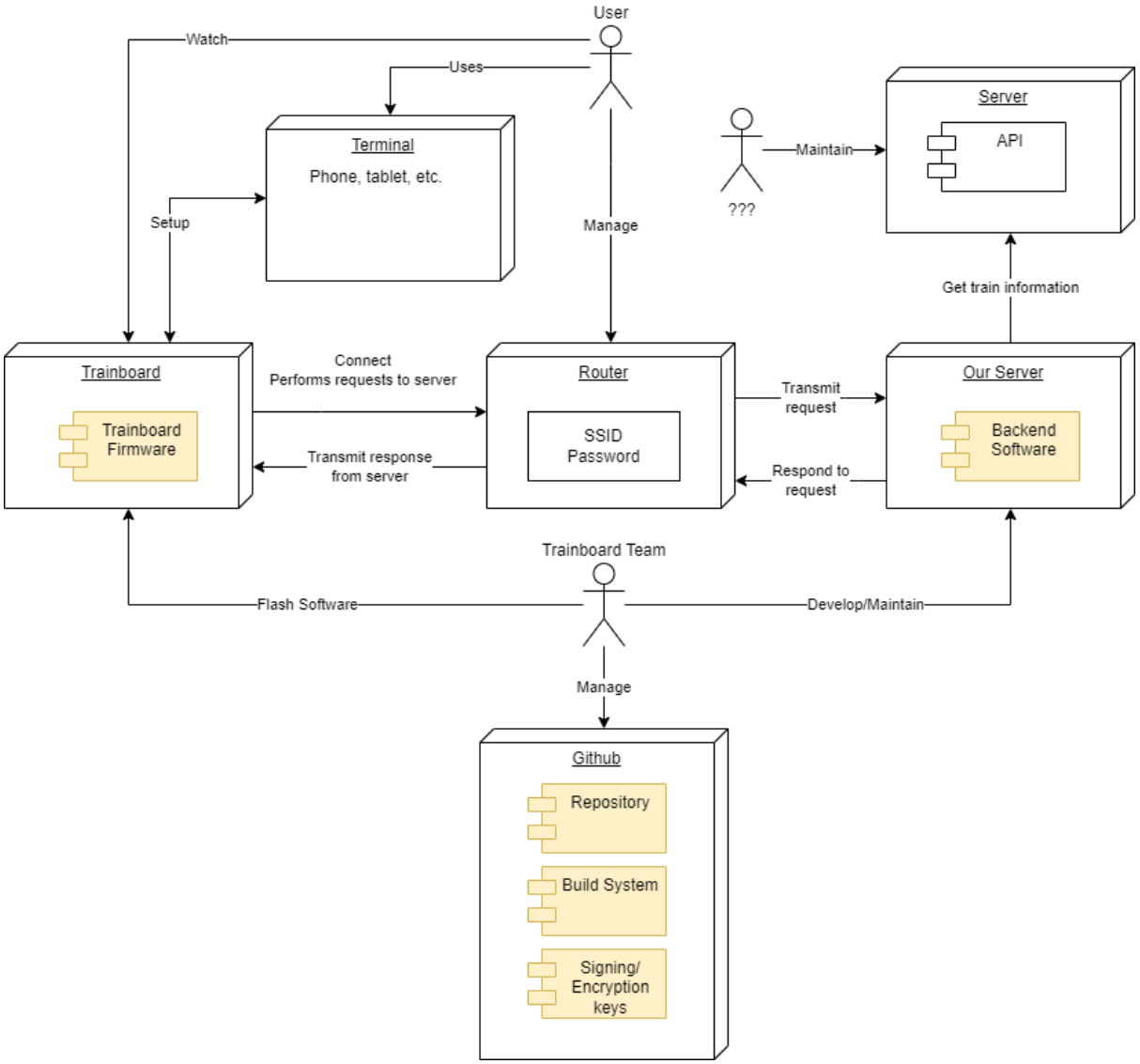
Constraints

Constraint	Description
Language	C++(11/14/17), makes it easier to design flexible software
Libraries	- Use MCU-friendly libraries, e.g. ETL instead of STL (no heap usage) - Use Arduino or Arduino-based framework for low-level software (portability)
Tools/IDE	PlatformIO, ESP-IDF, Visual Studio Code. These are open-source and available on all major OSes.
Connection to home network	No additional app on smartphone or other terminal, we do not have time, resources, nor money to design proper apps.
Versioning	Use Git for version control. Easier to share, maintain, and trace the software

Context and Scope

Business Context

The scope of this document is the Trainboard firmware.



Solution Strategy

Train lines

Control LED strips on board, no knowledge of train lines or train networks. This way there are no definitions to be updated on the boards if or when train lines change.

Client-Server Interface

Client Request

Once a minute, the client (trainboard) sends an https request to the server (see https://stackoverflow.com/a/8496157/21965131).

Request url: https://api.trainboard.ch/tb1_1
Add header to the http request with following information:

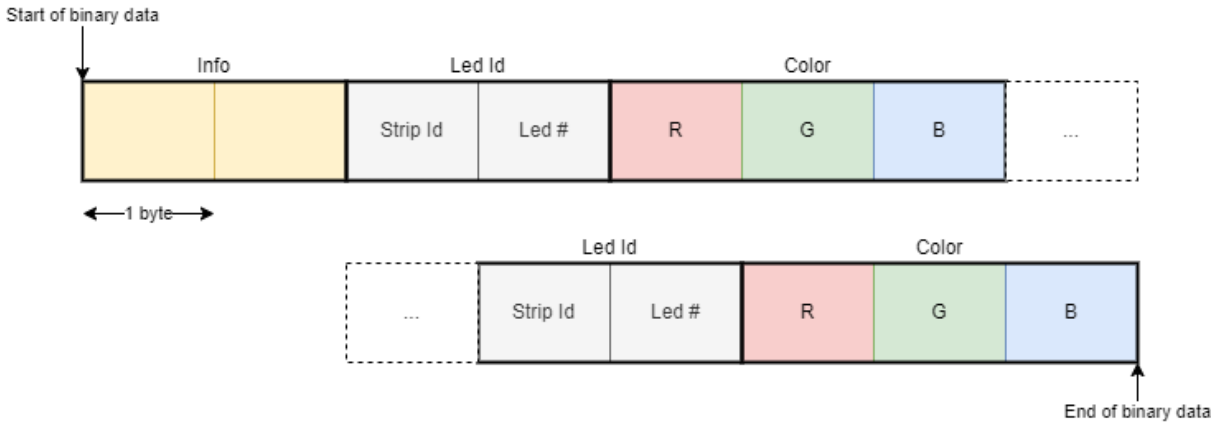
Header Key (String)	**Mandatory or optional**	**Header Value (String)**
fwv	M	Firmware version (x.x.x-hash)
hwv	M	Hardware version (x.x)
mac	O (M if com header is sent)	Ful mac address (xx-xx-xx-xx-xx-xx)
com	O	“history_x”, where x is the number of history frames

Server Response

Normal Request

The server responds to the request from the client with a data stream representing the LEDs to be lit.
The data payload is composed of a series of bytes. The first two bytes (in yellow) contain some information:

- First and bytes : number of LEDs to be enabled. This will be used by the ESP32 software to verify the data. Maximum is the number of LEDs on the board.
- Following two bytes : LED id
    - First byte : strip id (0-255, allows some flexibility in the future)
    - Second byte : led id (0-255, maximum 255 LEDs on one strip)
- Following three bytes: RGB color of the first LED
- These five bytes repeat until the end of the data.



History Request

In response to history requests, all frames are sent sequentially. The number of frames is defined in the request header.

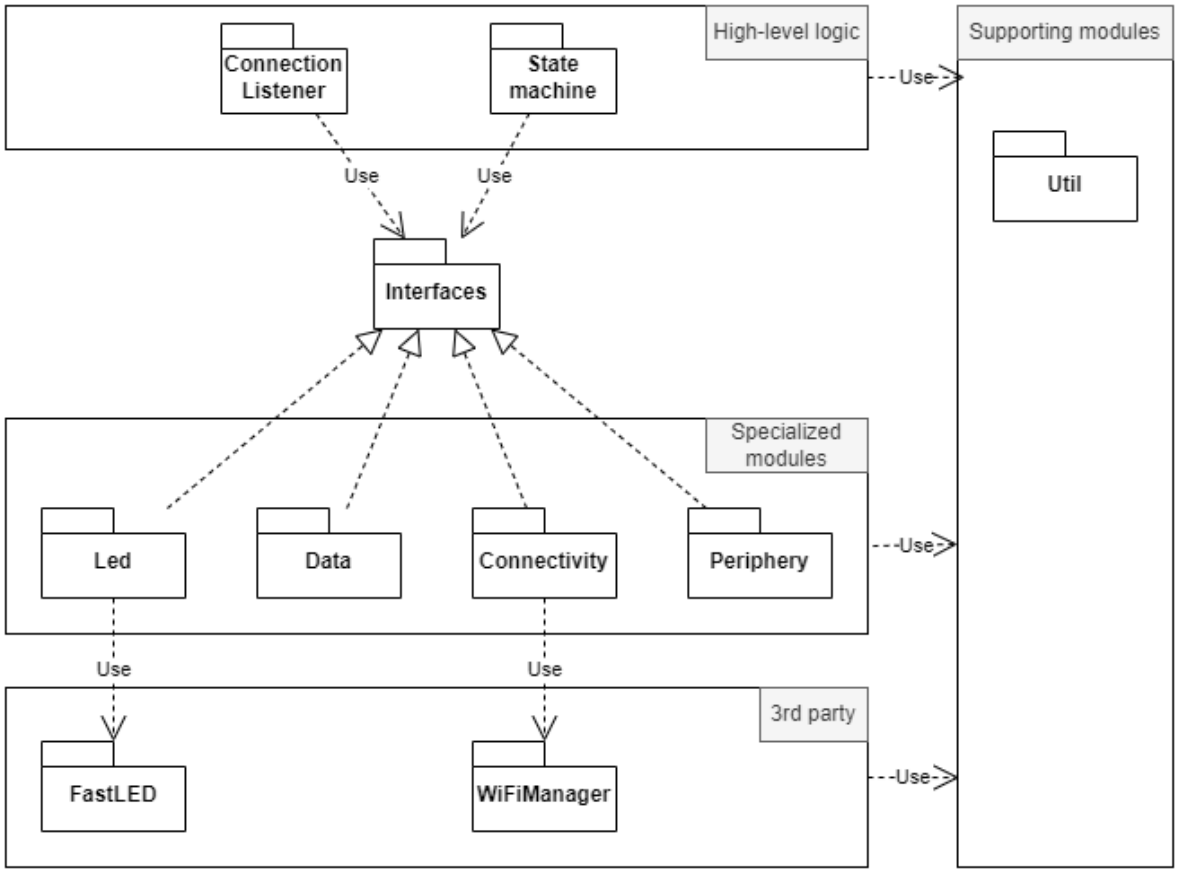
Provisioning

Use a capture portal. An Arduino library is used at first. In the future, we might implement this ourselves.

Building Block View

The high-level logic (business logic) is decoupled from the implementation details through interfaces. These interfaces define the required capabilities of each specialized module.

Each specialized module is independent and does not call any routines from other modules, except dependencies on 3rd party libraries.



State Machine

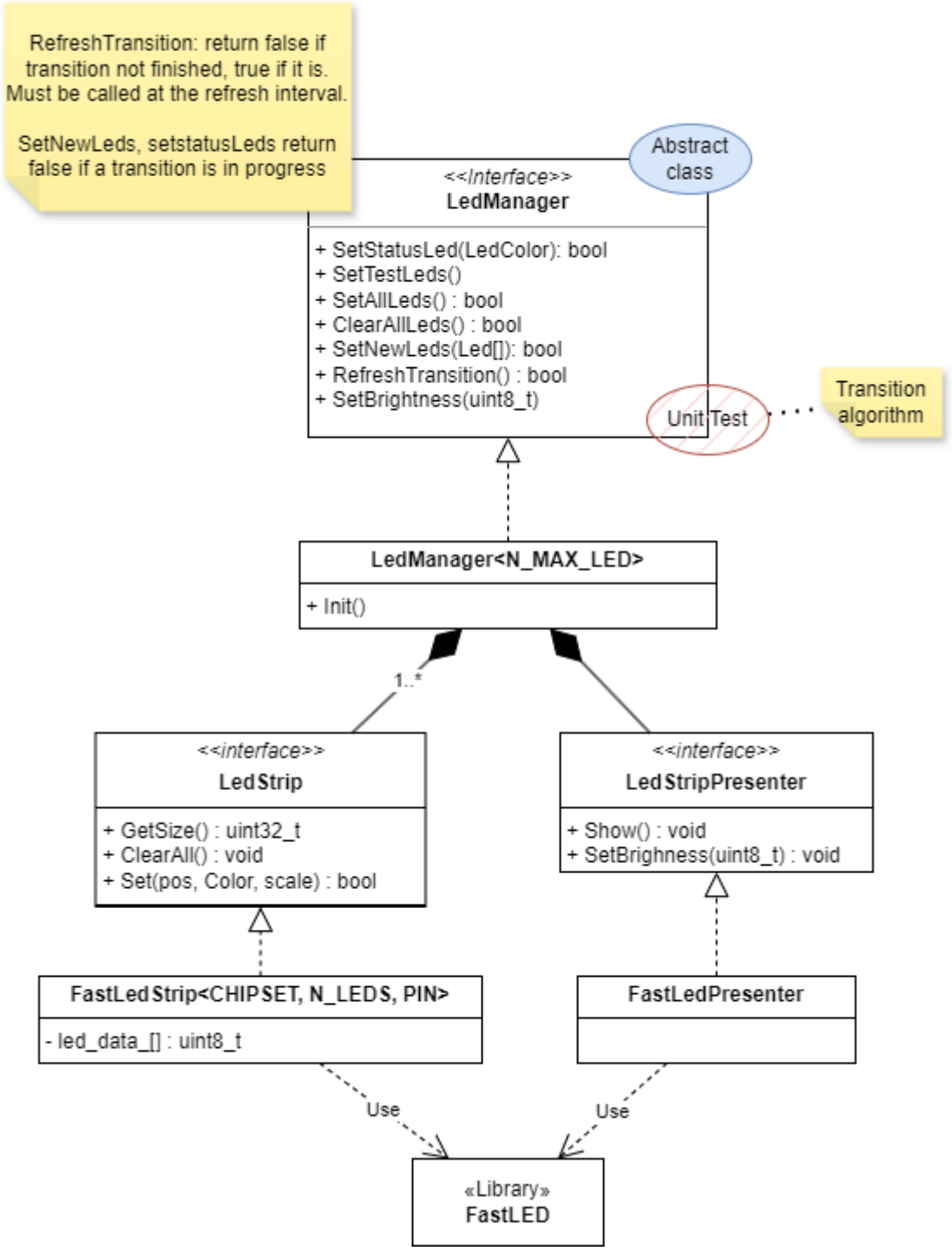
Contains the high-level logic of the board: startup, handling of button press, handling of connection changes, timing of server requests and LED updates.

Connection Listener

Checks whether or not the board is still connected to the Wi-Fi. Sends events when the connection status changes so other modules can respond to it.

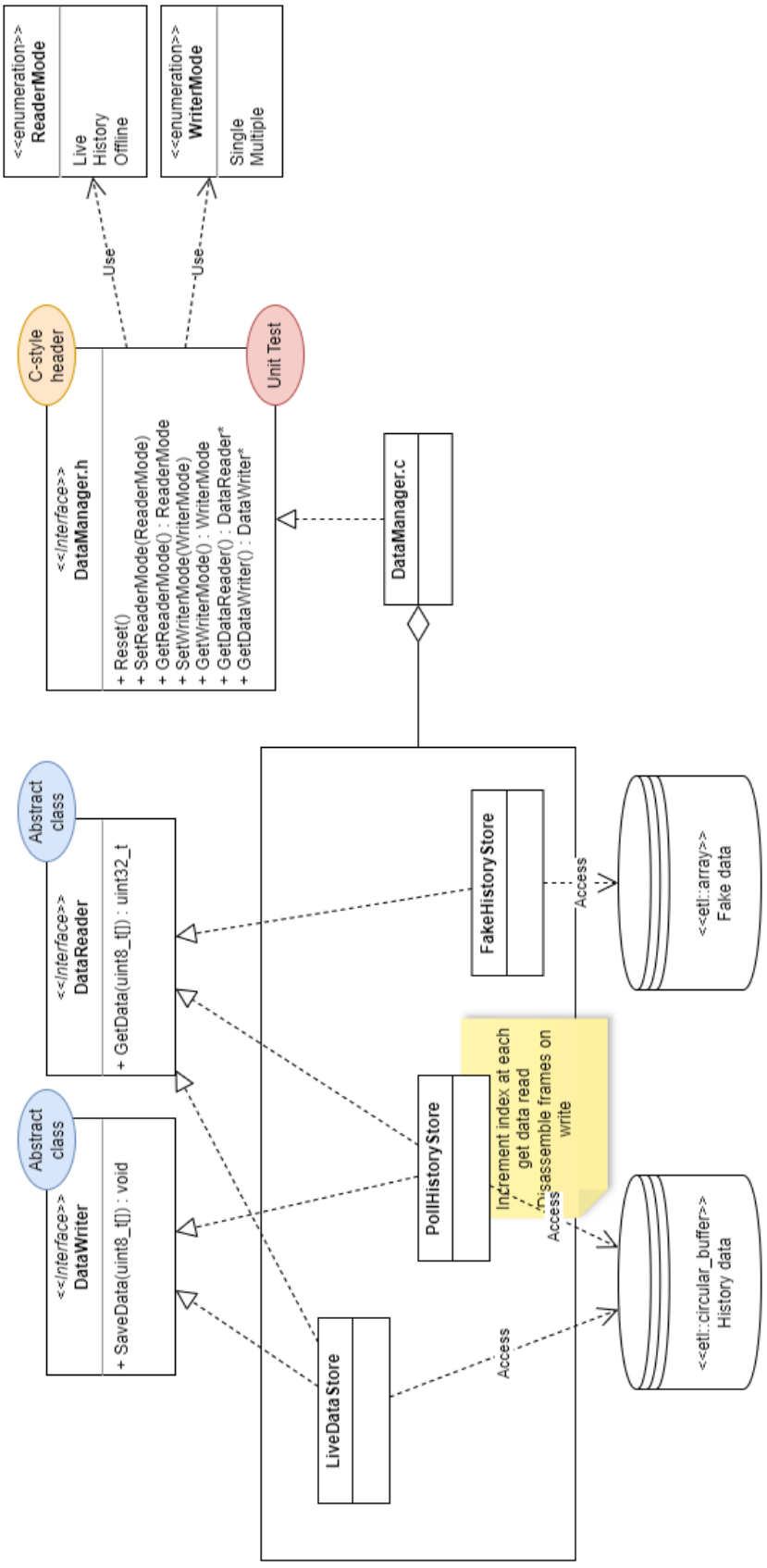
LED

Animates the LEDs when changing from one frame to the next.



Data

Checks data validity and stores new frames to a history buffer. Reads offline frames from non-volatile memory when no Wi-Fi connection is available.



Connectivity

Provides provisioning services and handles server requests.

Periphery

Monitors the push button and the ambient light sensor. Sends events when the button is pushed.

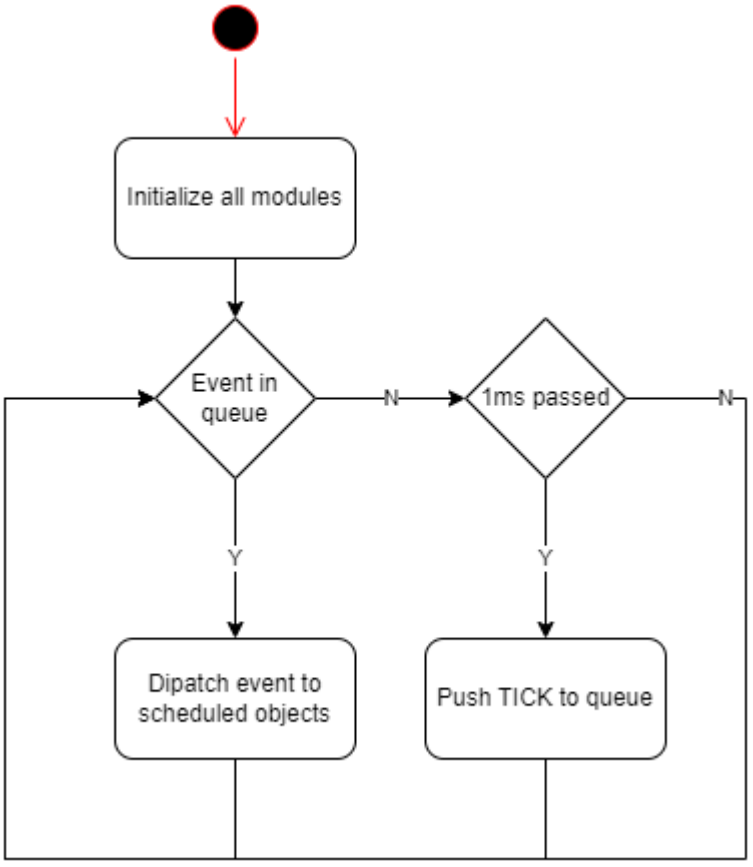
Architectural Decisions

- Use of Arduino framework to reduce development time.
- Simple single-thread scheduler to reduce multi-threading issues (especially when using FastLED and WiFi, there are some cache issues on the ESP32 because of the external SPI Flash).
- Inter-object communication using an event queue to decouple modules.
- Business logic decoupled from implementation details by the use of interfaces.

Cross-cutting Concepts

The scheduling for the application part is single-threaded for the sake of simplicity and to avoid problems with ESP cache.
The communication between different objects is done through an event queue. This allows modules to be decoupled.

The scheduling algorithm is very simple. It is based on an event queue and a 1ms tick. The events are immediately handled. Every 1ms there is a TICK event posted to the queue. The scheduler dispatches the events from the event queue to every active object (state machine, button, etc.) .



Risks and Technical Debt

Dependency on PlatformIO

This development environment lags a bit behind the official releases of the ESP-IDF and Arduino frameworks. It is not guaranteed that the newest versions will be supported. Here we should port the firmware to ESP-IDF + Arduino as a component.

Arduino

This is a hobbyist framework (although more and more used in a professional environment for prototyping and possibly also production). There are no clearly defined safety and security standards. Heavy use of the heap.

ESP-IDF

Releases only supported for two years. This means we would have to frequently update the firmware of the drainboards, possibly containing braking changes to deal with.

Secure Boot

No secure boot is implemented (apart from firmware signature). Here we have a risk if the board is used in environments where it can be accessed by malevolent actors.

Cybersecurity

As an IoT device, there is always a security risk. The analysis of this risk and the action items defined to mitigate it is done in the Security Concept document.

Glossary

Term	Definition
API	Application Programming Interface
LED	Light-emitting diode
RGB	Red Green Blue
SBB	Schweizerische Bundesbahn
SoC	System on Chip