HOCHSCHULE DÜSSELDORF

BACHELOR THESIS

Development of an Integration Platform for IoT Devices

Author:
Cara WATERMANN

Supervisor:
Dr. Christian GEIGER

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Engineering

in the

Research Group Name
Media Technology

Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Medien
Faculty of Media

December 5, 2018

Declaration of Authorship

I, Cara WATERMANN, declare that this thesis titled, "Development of an Integration Platform for IoT Devices" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Contents

List of Figures

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Chapter 1

Introduction and Motivation 1-2p

1.1 Introduction

The Influence of the Internet of Things (IoT) in everyday life has been rising for years. Connected devices are expected to number 20 billion [1] by 2020 in nearly every industry.

According to Verizon [2] the use on digital devices in the media and entertainment industry increased by 120 % in 2016 compared to 2013. The industry was third in terms of accepting IoT, with manufacturing (204%) and finance and insurance (138%) industries topping the chart.

Within the entertainment industry, escape rooms have been a growing sector since the first escape room launched 2007 in Japan. Escape rooms generally follow the same structure: People are locked into a room, have to solve riddles and get out in a defined period of time or will be released by a supervisor who watches the process to support and assist in case of an emergency.

This field offers lots of possibilities for technical development, be it the use of different sensors, the use of Virtual Reality or flexible story-telling (depending on the users actions). In this thesis, my goal was to create a suitable architecture and framework for further technical improvement for an existing escape room.

The faculty provided an escape room with microcontrolled riddles. The former architect who designed the riddles and the set-up of the architecture of the room left the project. Others have since tried to work with the existing architecture but struggled with it for reasons mentioned in Chapter 3. The room supported the existing riddles but modifications were inconvenient to integrate.

This thesis will focus on developing an easy integration system for new riddles from different devices. Furthermore, a user interface which supports communication with existing and new riddles dynamically was developed. The second chapter will provide an overview about the research on topics relevant for this project. The third chapter will analyze the escape rooms architecture concerning the research. The fourth chapter will explain in further detail how the project was implemented.

Chapter five will evaluate the implementation and examine future possibilites for the project.

Chapter 2

Related Work

As explained in Chapter 1, the goal of this project was to extend the existing project. Therefore, possible architectures and practices to integrate to the project were investigated. In the following, research concerning the development of our project is listed and explained.

2.1 Design Thinking

Design thinking is one strategy to design a product or an innovation process. As an innovation process, Plattner, Meinel and Leifer describe this process as "re-defining the problem, needfinding and benchmarking, ideating, building, testing." [3] It seemed to fit the project and was a guide concerning the production phase, further explained in Chapter4. In contrast to mechanical improvements, design thinking tries to emphatize with possible customer needs at all parts of the product. A few of design thinking's key principles are to "engage in early exploration of selected ideas, rapidly modelling potential solutions to encourage learning while doing, and allow for gaining additional insight into the viability of solutions before too much time or money has been spent" and that it "Iterates through the various stages, revisiting empathetic frames of mind and then redefining the challenge as new knowledge and insight is gained along the way." [4] The Stanford Design School, now known as the Hasso Plattner Institute of Design, began teaching a design thinking process with the three steps of understanding, improving and applying a product.

Since then, their approach to design thinking moved on to a widely used, opensourced 5 stage process [5] consisting of the following items:

Empathise
: Emphatising relies on three principles: Observe, engage and immerse with your customers

Define
: Stanford recommends to unpack the priorly collected findings "to needs and insights and scope a meaningful challenge [6]

Ideate

Ideation is the stage one should explore ideas in a "wide open" [6]. The goal is to create ideas that some can be picked from to create a prototype.

Prototype

Apart from testing, prototyping for this definition of the design thinking process serves many purposes. According to [6], one can also profit from prototyping for

- Empathy
- Exploration
- Inspiration
- Testing

purposes. One can receive a deepend understanding by building a prototype (Empathy), explore multiple concepts faster (Exploration), inspire other people for one's ideas (Inspiration) test and refine (Testing).

Test

The testing is an iterative process where one can refine and gather feedback about the product.

Figure 2.1 illustrates the iterative properties of this model.

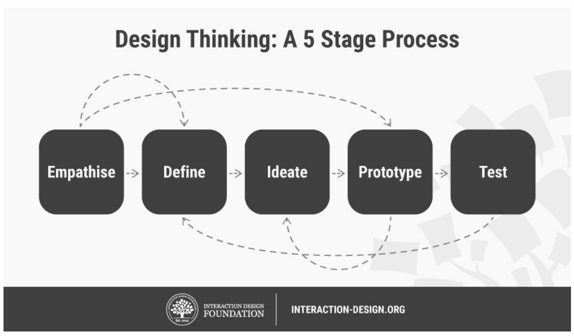

FIGURE 2.1

The model goes on to describe user analysis methods which are not relevant in the context of this project, since the project prioritized the development of a working prototype over extensive user studies.

2.2 Prototyping (1-2p)

A prototype is "An initial model of an object built to test a design." [7] A favored approach to prototyping within the IoT-scape is "Rapid Prototyping" which favors fast production cycles over extensive feature development. As S. Hodges states, "By prototyping and deploying live systems early on in the concept development cycle it is possible to understand the strengths and weaknesses of a particular application, design or specific implementation sooner and feed this information back into an iterative development process." [8] The same paper introduces .NET Gadgeteer, which was a rapid prototyping platform developed by Microsoft but is no longer maintained since 2016. The idea of Gadgeteer was to introduce a plug-and-play mechanism to IoT-development, where the developer had to connect the devices on a visual interface and code would be generated automatically. Due to it's high initial cost (250$ for a starter kit), and it's incompatibility with other shields it was not competitive against the Netduino or the Arduino platform explained in the "Device"-section below. Two generally important concepts in a product lifecycle surrounding a development process are the "Proof of Concept" and the "Minimal Viable Product".

The business dictionary defines a proof of concept as "Evidence which establishes that an idea, invention, process, or business model is feasible." [9] A proof of concept can take many forms depending on the product and the industry it develops in. In a technological environment, proof of concept often take the form of prototypes defined by a set of goals.

A minimival viable product (MVP) is a product with "sufficient features to satisfy early adopters" [10]. Only after considering feedback by customers is the product developed further. MVPs allow companies to publish a product as early as possible which leads to early monitary profit and fast feedback. On this basis, user The concept has been popularized by Eric Ries, a consultant of start-ups.

2.3 Architecture

As there is, at this point in time, no consensus reached for a layer model defined for IoT-architectures [**noModel**] different approaches can be used to analyze and structure an IoT-system. The following describes three proposed layer-models.

2.3.1 Gartner IoT architecture

This project adapted Gartner's take on IoT architecture as it offers a powerful and complex overview about different possibilities of IoT integration. It offers an architecture as well as a taylored layer model.

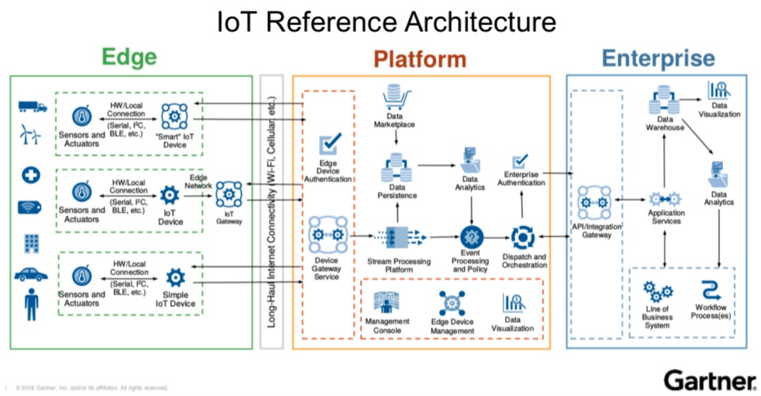

FIGURE 2.2: Tiers of the Gartner IoT architecture

Edge Tier

The left part of Figure 2.2 depicts the edge tier of an IoT-architecture. The "Edge Tier" is where sensors and actuators lie. Wheras a sensor *detects* interaction or changes in a physical environment, an actuator is "a device that is used to *effect* a change in the environment" such as the temperature controller of an air conditioner [11]. Sensors and actutators typically complement each other. The figure describes 3 general forms the edge can take. It is always a combination of sensors/actuators with either:

1. A "smart" IoT device which pre-processes data before it sends data to the device gateway service
2. An IoT device with an edge gateway physically connected which transmits the devices data to the device gateway service
3. A simple IoT device which connects to the device gateway service directly without pre-processing the data from the sensors/actuators

Platform Tier

The middle part of Figure 2.2 shows the platform tier of an IoT-architecture. According to Gartner, the pre-processed or raw data from the edge is processed here. Stream processing, event handling and database implementation take place. Additionaly, edge devices can be overseen with monitoring tools integrated. If needed, further requirements for enterprise authentification and handling are also implemented at that layer. Summing up, this layer is responsible for all data-managment tasks that might arise in an IoT-system.

Enterprise Tier

The "Enterprise Tier", depicted on the right, is the customer part in an enterprise solution for IoT-architectures. It provides the customer with necessary data in a pleasant and clear way. While the customer has access to the platform layer through the application layer, he doesn't get in touch with the platform layer directly.

The following describes Gartner's proposed Layer model for IoT-systems.

Device Layer

The Device Layer is the phyiscal layer in this model. It owns the Edge Tier properties, sensors, actuators and respectively one of the three extending devices. According to Gartner, this is the recommended layer to start when planning an IoT-architecture, as it defines the bandwidth of devices that need to communicate with a gateway and therefore the communication protocols that work with it in the long run. If an edge gateway is present, it's also part of the Device Layer.

Communication Layer

This layer defines how the communication is taking place within an IoT-system. Depending on the Devices within the system, different communication protocols and data models should be implemented. Examples of IoT-protocols are MQTT, Wi-Fi, WPanO6, Zigbee, wheras data models include Apples HomeKit Database, the open-source OpenHab Things model or the SmartThings Capabilities model by Samsung. Different models work with different protocols, so possible future device implementations must be considered.

Information Layer

The Information layer defines how the data is *formatted so it can be interpreted*. Depending on the protocols and data models chosen in the Communication Layer, different strategies to format messages by devices can be applied. Endpoint and edge identification is important to access different features provided by a Device. The messages need to be interpretable possibly by various Devices.

Function Layer

The Function Layer is the core of any IoT-application. It handles event processing, stream processing, analytics and possibly machine learning. Any middleware and back-end tasks are handled here.

Process Layer

The Process Layer is the front-end in this Layer Model. Proccessed data can be displayed and managed by a user. Depending on the use case, several process layers can belong to one IoT-system (e.g. one for a device manager, and one for displaying data).

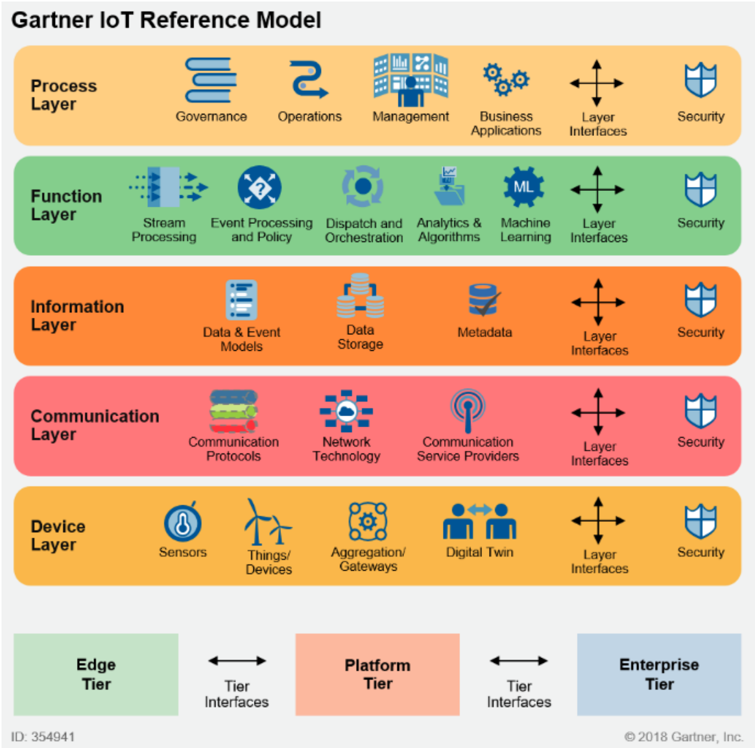

FIGURE 2.3: Gartner Layer Model

2.3.2 Five-Layer architecture

One by the authors of [12, 13] proposed model is a five-layer-model consisting of the following layers.

Perception layer
The perception layer is the physical layer of the architecture. Sensors and actuators exist in this layer.

Transport layer
The transport layer transports data from the perception layer to the processing layer. Different network protocols can be used

Processing layer
The processing layer stores, analyzes and channels incoming data. It is also known as the middleware layer.

Application layer
The application layer is responsible for delivering user-relevant data to a user.

Business layer
The business layer manages the whole system, e.g. applications, business and profit models.

When talking about IoT-architectures, one should mention the difference between cloud and fog/edge based architectures.

Cloud based architectures assume that processing and analyzing of data should happen in an enviroment remote from the devices' location. The network below sends data to the cloud, and above the cloud lie applications working with the processed data with the cloud in the center of this architecture. In the last years, cloud computing has gained popularity, also in the context of IoT architectures [14] because it provides great exibility and scalability.

A newer trend instead are fog or edge based architectures, where the sensors and gateways do parts of the data processing and analytics. A fog architecture [15, 16] presents a layered approach, inserting various layers between the physical (perception) and transport layers (monitoring, pre-processing, storage, security). Wheras fog computing refers to smart sensors and gateways, edge computing refers to not-smart objects like motors, pumps with smart data preprocessing capabilities [**edgeFog**].

2.4 Front-End

As [17] states: "A front-end system is part of an information system that is directly accessed and interacted with by the user to receive or utilize back-end capabilities of the host system. It enables users to access and request the features and services of the underlying information system."

The front-end is the visual component of any application.

Disciplines like UX (User Experience), UI(User Interface) and IxD (Interaction design) have been working on creating a "better" front-end-experience for many years.

Usually, a mixture of HTML, CSS and Javascript is used for front-end development. In recent years, libaries and frameworks that encourage a component-based architecture are becoming increasingly popular. The most popular representatives are currently React.js, Angular and Vue [18] , where the former two are established tools and the later has been on the rise for the last two years. The table in Figure 2.4 displays the differences and similarities between the three tools.

For choosing the right libary or framework, the displayed categories seemed important. The learning curve would enable the author to use the framwork fast, as there was no prior experience in any framework. The relative popularity could dertemine the amount of data one would have access to when researching for information on the framework. "Loved by developers" is an indicator for the long-term ease of use and how more advanced developments would affect especially future developers. Noteworthy users indicates what kind of traffic a website with the framework can handle and how the websites using these frameworks look like.

2.5 Communication

There are different ways to communicate between the client and the server of a website.

HTTP (Hypertext Transfer Protocol) is the most popular way to communicate between a client and a server. It can be used with different languages and frameworks and is usually implemented considering the REpresentational State Transfer (REST) architectural style. The REST style requires a communication to

1. Seperate the client (visuals) from the server (data storage)
2. Communication should be "stateless", which means every time the client requests data from the server, it needs to add every information necessary to send the data back to the client. This is comparable to having to send your phone number with every message while chatting, so the person on the other end, who has no prior messages from you saved because they are immediately deleted once he or she answers, can text you back. You (the client), are the only one who knows the other persons phone number and anytime you want to hear something from them, you have to initiate the conversation.
3. The incoming data should be cacheable. Messages coming from the server are tagged "cacheable" or "non-cacheable" so the client is given the right to reuse that response data for later or to delete it after usage.
4. A uniform interface. "REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state." [19]

Due to its stateless nature, a RESTful HTTP approach always needs the client to request data from the server. There are different techniques to optimize this kind of communication. A Medium article [20] explains the differences with a great analogy: waiting for cake to be ready. Either the client requests cake from the server every possible second (HTTP Short Polling) and asks for the next cake (data) as soon as it gets a response which would look like this in a conversation: *C: "Is the cake ready?" C: "Is the cake ready?" C: "Is the cake ready?" S: "Yes" C: "Is the next cake ready?"* , or the server holds the request open until the cake is ready (HTTP Long Polling) *C: "Is the cake ready?" - pause - S:"Yes" C:"Is the next cake ready?"*. A third option is "HTTP Periodic Polling" which requests new data in a set timeframe. Essentially, it works like HTTP Short Polling but by increasing the time between the requests, less traffic is produced and the server is relieved. AJAX [21] is one of the most famous technologies developed to enable periodic polling. Still, this technique needs a client to request new content instead of listening for new information from the server. Chat-applications e.g. shouldn't require the user to wait for new messages longer than necessary. Those services demand a more reactive and immediate communication. Also, "sending the phone number" meaning the client information with every message creates overhead for every message.

Websockets pursue this task by creating a bilateral environment for clint-server-exchange. This way, the client doesn't need to connect again and again, but listens for events once a connection is established. Now-a-days, most browsers are websocket compatible. A study in 2013 [22] discovered that a WebSockets server consumes 50 % less network bandwidth than an AJAX server. Furthermore, it stated that a WebSockets client consumes memory at constant rate, in difference to AJAX which consumes memory at an increasing rate, and that WebSockets can send up to 215.44 % more data samples when consuming the same amount network bandwidth.

The table below compares RESTful HTTP with Websockets.

2.6 Back-End

According to the Oxford Dictionary, a back-end is "The part of a computer system, piece of software, etc., where data is stored or processed rather than the parts that are seen and directly used by the user" [23]

The back-end can further be separated into the processing and the storing part of the system. The storing and retrieving of data is usually handled by a database management system (DBMS). Different frameworks further process the retrieved data and enable middleware functionalities to connect to other services (like the front-end). Middleware is "software that acts as a bridge between an operating system or database and applications, especially on a network" [24]. The Table in figure 2.6 compares the three most popular framework implementations currently used in web development. The catogries where chosen to match the front-end technologies' categories except for the "Packages available" and "Performance" category.

Any of these frameworks possess a "package manager" which allows users to download external libaries directly into their project. Those libaries are called "packages" and can be uploaded by every developer. Therefore, only a small percentage of packages are professionally well-designed. The goal of a package is first and foremost to ease the development process of an else difficult to implement feature e.g. routing, translation or database-access. The frameworks also have different starting capabilities, Node.js e.g. relies a lot more on external packages than ASP.Net. Still, the "Packages available" indicates, additionally to the "Relative Popularity" category, how much community support is available.

The performance category indicates how fast a framework can handle certain requests. This is important for scalability of a project. In this case, a benchmarking-test was utilized to estimate general performance [25].

2.6.1 Database

A database stores the data, whereas a DataBase-Management-System (DBMS) makes the data accessible from elsewhere. Famous, established examples for DBMS are MySQL and Oracle. Most DBMS are heavily influenced by the standardised SQL-language which was first introduced in 1970 by Edgar F. Codd in context with the relational model. An important idea of the relational model is the concept of database normalization, where related records are linked by a key predicate common to all. Now-a-days, not relational (NoSQL) DBMS like MongoDB gain market share. NoSQL DBMS are mainly used for retrieving and save documents with properties unfit for the SQL-scheme with more unorganized relationships (videos, e-mails, social-media posts). Table 2.7 shows different aspects of the 5 most popular DBMS systems.

2.7 Device

An IoT-Device can take many forms: Sensors and actuators can be used to create nearly any use case, from motion detection for light automation, to tracking the productivity of a machine to automated heating depending on room temperature.

2.7.1 Arduino

A popular choice for self-made technology projects is the Arduino microprocessor.

Arduino is a microcontroller-company from italy which was founded in 2005. It is completely open source and provides its own Integrated Development Enviroment(IDE) [**arduinoIDEDownload**]. The IDE works with nearly all microprocessors on the market. The IDE recommends a structure for all Arduino programs:

Initialization Prior to any function, libaries and necessary variables are declared within an Arduino program.

setup() The setup-function is the first function called in any Arduino-program. This is were variables, supporting hardware or the serialport are initialized and pin modes are set.

loop() The loop-function loops consecutively, which enables the program to change and respond on run-time. Checking for changes usually happens here, whereas consequences of those changes are implemented outside the loop-function.

The Arduino (and comperable microprocessors) are programmed in C.

Right now, the Arduino is not the most typical choice for IoT-devices in particular as Arduinos with included supporting hardware (e.g. Wi-Fi-modules) released just this year, however due to it's popularity many libaries and instructions exist to introduce an Arduino to an IoT-scope. E.g. with additional hardware like a Wi-Fi shield and existing free apps like "Blynk" [26], an Arduino can be tracked and controlled within 30 minutes of set-up.

2.8 Summary

As one can see, a lot of options are available when developing an IoT-project. Different architectures, protocols, devices, and software settings blow up the decision making process. Communication protocols and platforms available for other protocols weren't explained in further detail since they were not appliable to the current project settings. While there is no industry standart, it has to be mentioned that a lot of open source and commercial software is developed for different projects. A good example is "The Things Network" [27] which tries to make IoT accessible for everyone with LoRa-Wan. This technology needs only a few gateways within a city instead of every user having to build their own gateway. Even easier to integrate are IoT-devices using the MQTT protocol with e.g. Amazon Web Services [28] or the Microsoft Azure Cloud technology [29]. For this use case, an integration platform had to be developed by scratch.

Category	React	Angular	Vue
Type	Library	Framework	Library
Learning curve	+	o	++
Relative Popularity	+	++	-
Loved by Developers	++	+	+
Performance (ms)	3037	2743	2820
Size (kb)	193	129	100
Lines of Code	2048	2056	2030
Maintained by	Facebook	Google	Crowd-Funded Development
Noteworthy Users	Facebook, Netflix, Airbnb	Google, Udemy, Nike, AWS	AliBaba, Adobe, Gitlab, Grammarly
Model	Virtual DOM (Document-Object Model)	MVC (Model-View-Controller	Virtual DOM (Document-Object Model)
Language Preference	JSX-Javascript XML	Typescript	HTML Templates and Javascript
Job Offers	++	+	-

	React	Angular	Vue
Learning curve	+	o	++
Relative Popularity	+	++	-
Loved by Developers	++	+	+
Noteworthy Users	Facebook, Netflix, Airbnb	Google, Udemy, Nike, AWS	AliBaba, Adobe, Gitlab, Grammarly
Model	Virtual DOM (Document-Object Model)	MVC (Model-View-Controller	Virtual DOM (Document-Object Model)
Language Preference	JSX-Javascript XML	Typescript	HTML Templates and Javascript

1

FIGURE 2.4: Front-End Comparison

	HTTP (REST)	Websockets
Easy to Implement	+	++
Fast Messaging	-	++
Little Overhead	Moderate overhead per request/ connection	Moderate overhead to establish and maintain the connection, then minimal overhead per message
Bandwidth (for one Data transfer)	~282 Bytes	Initially ~300 bytes, then 54 bytes for
Broad Client Support	Borad support	Modern languages and clients
Messaging pattern	Request-response	Bi-directional
Example for implement	Ajax	Socket.io

Category	Node.Js + Express	ASP.Net	Php	Django	Ruby on Rails
Type	Runtime Environment	Framework	Framework	Framework	Framework
Release	2009	2002	1995	2005	2005
Learning curve	++	o	+	+	o
Relative Popularity	+	++	++	-	o
Performance (by number of JSON responses/s)	+	++	—	o	—
Latency (of JSON responses)	0.9 ms	0.6ms	0.5 ms (HHVM)	4.5 ms	14.0 ms
Maintained by	Open-Source	Microsoft	Open-Source	Open-Source	Open-Source
Noteworthy Users	Paypal, Yahoo, Wall Street Journal	StackOverflow, W3Schools, Microsoft MSDN	Wikipedia, Tumblr, Slack	Youtube, Google, Quora, Reddit	Kickstarter, ask.fm, Hulu, Groupon
Language Preference	Javascript	C#/Typescript	Php	Python	Ruby
Language used server side	1 %	11,9 %	78,9 %	1 %	2,3 %
Market Share in the Alexa top 1 Million	1,03	20 %	20 %	< 0.41%	1,72 %
Packages available on main distributor	> 350,000 (npm)	1385 (nuget)	204 535 (packagist)	90,000 (PyPi)	125,000 (rubygems)

FIGURE 2.6: Backend Framework Comparison

	PostgresSQL	MySQL	Oracle	Microsoft SQL	MongoDB
Primary model	relational DBMS	relational DBMS	relational DBMS	relational DBMS	Document store
Initial release	1989	1995	1980	1998	2009
Licensing	MIT-style license (open-source)	GNU (open-source)	OTN-License (No commercial use)	No commercial use but offers light-weight open-source version	SSPL (open-source) with commercial options
ACID (Atomicity, Consistency, Isolation, Durability)	Yes	Yes	Yes	Yes	Multi-document ACID Transactions with snapshot isolation
Community	++	+	++	+	++
Noteworthy users	Spotify, Apple, Instagram, Uber	Facebook, Airbnb, Youtube	Coca-Cola, Svarowski, Dr.Oetker	StackOverflow, Microsoft, MIT	Sega, Nokia, Gap, EA, Adobe, Cisco, SAP

FIGURE 2.7: DBMS Comparison

Chapter 3

Project Overview

3.1 Analysis

3.1.1 Layer Analysis

Refering to Figure 2.3, we analyzed the existing architecture of the escape room.

Device Layer

The Device Layer consisted of multiple Arduinos and riddle-depending sensors and actuators for each riddle. The gateway device service was an Adafruit Feather 32u4 which was connected via USB to a computer.

Communication Layer

The communication within the room was set-up with RFM69HCW transceiver modules.[30] The RFM69HCW is a very cheap, easy to use and to set-up transceiver. They do packetization, error correction and auto-retransmit which makes them easy to handle. They are designed for point-to-multipoint communication with one transceiver set as a gateway node which sends data to the other transceivers in the room. There are two open-source libaries for the RFM69HCW, the LowPowerlab [31] and Radiohead [32] library. The escape room used the LowPowerlab library since the architect who designed the room prior to our adjustments was familiar with the library. Now-a-days the Radiohead library is the recommended library [33] since it's documented more thorough by an active community, kept up-to-date and is cross-platform friendly.

The gateway transceiver was connected via USB to a computer where the data was forwarded via UART serialport communication. UART is asynchronous and needs to be made synchronous to be interpreted by another device. Therefore, a stop- and start-bit is added to any message and transmission speed needs to be set on both sides. If the transmission speeds differ, the messages can't be interpreted correctly by either side.

Any node (riddle) within the room was recognized by a different nodeID and the gateway was detected with a specified "GatewayID". For security-reasons, password-encryption was used between the nodes.

The Arduinos within the room connected to the RFM69HCW via Serial Pheriphical Interface (SPI). SPI is a serial communication protocol common for micro-processor connection. It uses an extra "Clock" (CLK) line to keep both sides in sync. Only one side generates the clock signal (usually called the "master"). The other side is called the "slave". There can be multiple slaves, but only one master. The clock is an oscillating signal that tells the receiver when to sample the bits on the data line exactly. Bits are send either on the "high to low" or "low to high" edge of a CLK signal. If the master wants to send data to a slave, it's send via a MOSI line, for “Master Out / Slave In”. If a slave wants to send data to the master the data will be put on a third line called MISO for "Master In/Slave Out". The master will continue to generate a prearranged number of clock cycles, before the message is read by the master. As there is a MISO and a MOSI line, full-duplex communication where data is simultaniously send and received is possible with SPI. The fourth line needed to enable SPI communication is the "Slave Select"(SS) line which opens the communication channel. If there is only one slave, the "SS" line is kept low (its active state) as long is the device is on.

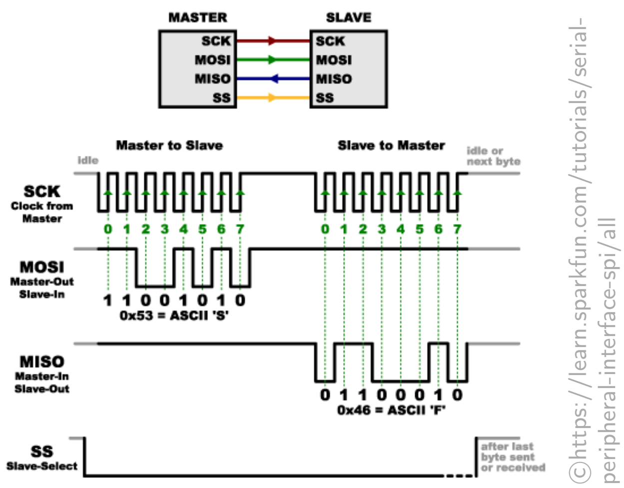

FIGURE 3.1: Visualization of SPI communication

Information Layer

The Information Layer was build around a simple event model: the transceivers would send and receive numeric codes in a "n1/n2/n3...\n"-structure, e.g. "16/2/0\n". A riddle could be identified through the string it sent, since the first number would be the riddle's adressing number. Further numbers, separated by a backslash, would show the active state of the device. The Arduino could process the incoming data in a matching "Switch-Case" scenario shown in figure 3.3

Function Layer
A C++-Server would broadcast all incoming messages to any TCP-client. The server would forward any messages coming from a TCP-client back to the serialport just as well.

Process Layer
If an incoming message matched with a list in Unity, a Video in Unity would be played and Unity would send a "reset"-message to the matching riddle. Apart from the main functionality, the C++-Server offered a communication window where serialmessages could be seen and sent manually shown in figure 3.4.

Figure 3.2 provides further insight into the architecture

3.1.2 Workload analysis

In 2009, Kim Goodwin stated that „Interactive products and services tend to require four different types of work from users: cognitive, visual, memory, and physical“ [34]. While an IoT-System is not always interactive, the goal of this thesis was very interaction-orientated: The goal was to help engineers expand the existing room which would require interaction with the Device-Layer (if they wanted to add hardware-functionality) and the Function-Layer (if they wanted to add software-functionality). Therefore it makes sense to analyze this specific project with those aspects in mind.

It should be mentioned that the room was well designed for the target user, which is an escape room customer, and the four areas of cognitive, visual, memory and physical work involved would already be pretty low. Because this analysis will focus on an engineers point of view, the mentioned user in this case is the engineer who wants to extend or modify the room in some way.

Cognitive Work

Engineering products usually demands some kind of cognitive work. Still, there are hurdles that can be avoided, like finding out wether to click "yes" "no" or "cancel" in a confirmation dialog can be made clearer if the question is phrased easily. In this case, the cognitive work needed to add a riddle or a functionality was very high:

Device Layer:

1. The engineer needs to understand the transport protocol and therefore needs to
   (a) Set-Up the RFM69HCW with an Arduino or an Adafruit Feather, which requires drivers and another library
   (b) Understand the communication within the LowPowerlab-library

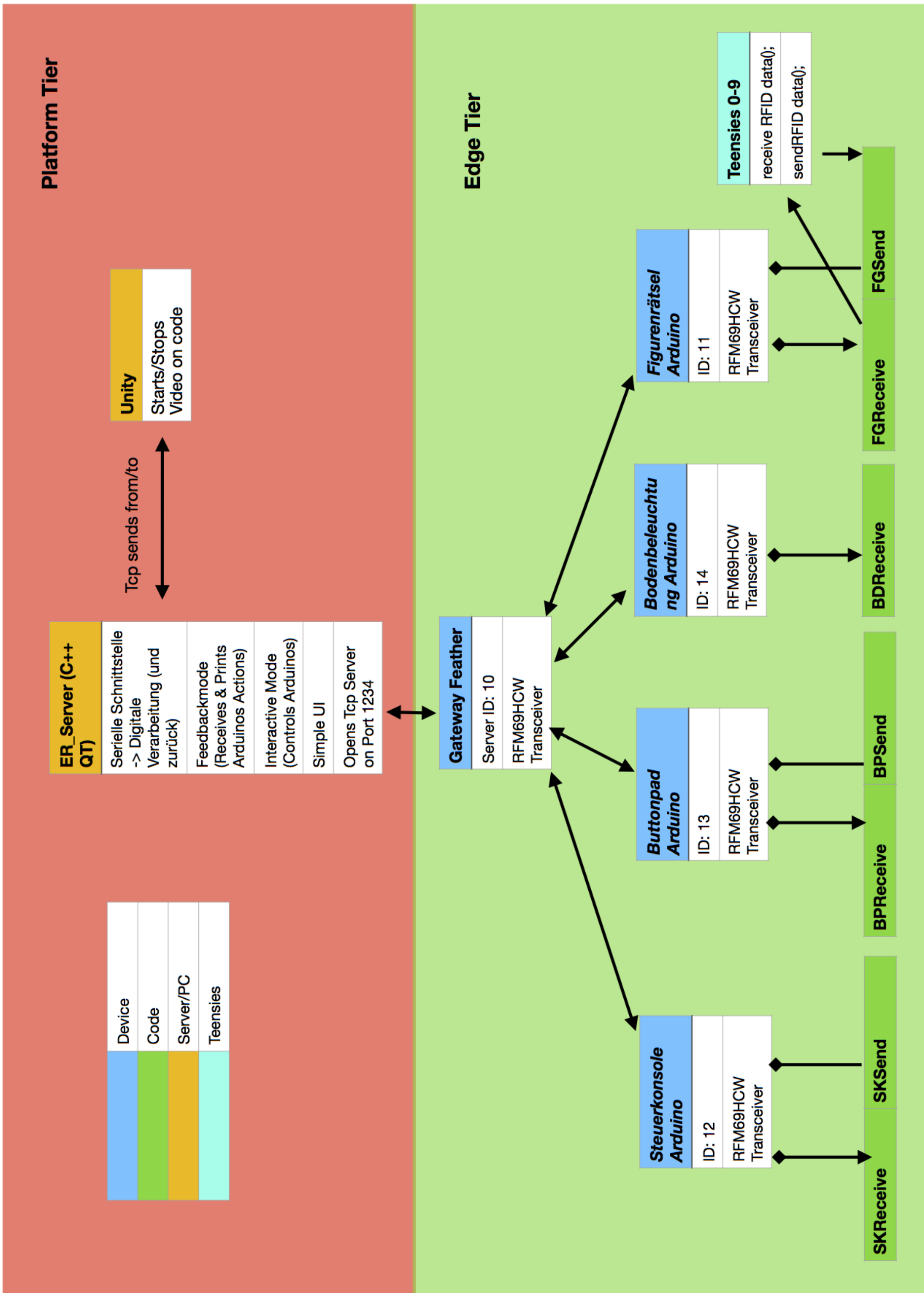

FIGURE 3.2: The old escape room architecture

```
void execCommand( int exec[]) {

 switch (exec[0])
 {
   case 1: // Mode
     { // okay
       if (exec[1] == 0) { // Demo-Mode
         mode = 0;
         String message = "0/0/\n";
         rf69send(message);
         resetDemo();
       }
       if (exec[1] == 1) { // Demo Mode with Feedback
         mode = 1;
         String message = "0/1/\n";
         rf69send(message);
         resetDemo();
       }

       if (exec[1] == 2) { // Interactive Mode
         mode = 2;
         String message = "0/2/\n";
         rf69send(message);
         resetOff();
       }
       break;
```

FIGURE 3.3: Example of the messages defined in the Arduino

FIGURE 3.4: Original Serial Window

   (c) Look up the other riddles codes to avoid using a

2. Understand Arduino-coding
3. Understand working with the "Switch-Case"-scenario used for communication

Process Layer:

1. Understand C++ to change the Server (f.e. to communicate with an upper-level protocol)
2. Understand C# and Low-Level-Socket communication to make changes in Unity and gain an overview about the communication since it's in separate files (one File per Video and one for the Tcp Socket)

Visual Work

Visual work means how much the user needs to search for features in a product visually. The visual work within the architecture was low, as the interface to work with was simple. It didn't offer needed functions for the user, like buttons for often used features(reset, "send feedback"...).

Memory Work

Memory work is measured in how much a user has to remember to succeed in a task. Typical examples are passwords, command names, and file names. In its former state, the architecture demanded a high amout of memory work from a developer:

1. The developer has to translate "n1/n2/n3...\n"-codes whenever he wants to understand the serialport-messages
2. The developer has to remember a different code when he wants to send an order to the device
3. Anyone who starts the room has to remember to first start the c++ server and afterwards the unity application on the desktop, or connection will fail.
4. The developer has to enable the "enable feedback" mode for each node manually if he wants to see all messages send
5. The developer has to enable another mode if he wants to interact with the riddles, and the riddle won't react hardware-wise in that mode.

Physical Work

IoT-Projects always involve some kind of physical work for a developer: the developers need to switch between hardware and software to test the devices for example. The escape room fits into that scenario but doesn't make testing physically harder than it needs to be in most aspects. One aspect that makes changes harder is seemingly unavoidable - most of the hardware is hidden, therefore mostly difficult to access. Since an escape room is made for customers who expect the illusion of in this case, a spaceship, touch-sensitive hardware would impact that illusion.

Chapter 4

Implementation

After introducing the project as it existed before this thesis, this chapter will concern the planning and implementation of the extension that was build. The implementation used the aforementioned tools as well as some smaller libaries that will be explained in the following sections.

4.1 Project Design

The project's design followed the in Chapter 2 design thinking approach. The following shows the different steps that were taken that lead to the changes that were planned.

Empathise

As a first step, interviews and tests with the existing project were executed. Orignially, the author wanted to build another riddle for the room. It was quickly discovered that constraints would complicate that task. Three people who had worked with the room reported they experienced severe difficulties on trying to change the existing pattern. Frequently mentioned was the overall lack of understanding the room as a whole. Some parts, like the TCP-socket in Unity, were relatively easy to modify, others, like the riddles themselves were disclaimed "not to be touched or they might break". As the room had many visitors (a few a hours a day, 2-4 times a week), changes which would affect the look of the room or make it unstable for a longer period of time were not welcomed.

In the time following these interviews, a deeper occupation with the project seemed necessary to develop alternative ideas for this thesis. The author's impression was that the project lacked documentation and explanations on many sides. Understanding the processes and the different parts of communication proved to be difficult, as it didn't seem to follow any standartized structure or protocol. The result of this examination is further explained in Chapter 3.

On questioning the former architect about his choices, he stated that the project was build under time pressure and everything was therefore implemented best to the architect's knowledge, but without further scientific research or extensive planning.

Define

The "Empathise" stage lead to the impression that a new riddle would be considered "nice-to-have" whereas extensions to improve the flexibility and comprehensibility of the room would be welcomed. Derived from the workload anyalysis in Chapter 3, it was determined that the cognitive and memory work required from a developer were the most critical points that one might spend a lot of time on, or might decide not to join the project at all. It was defined that:

To reduce cognitive (C) work, the process of learning and discovering the project must be simplified.

To reduce the memory (M) work, the amount of commands to remember to control the room must be reduced and the start-up of the room must be simplified.

Ideate

The next step was to devise ideas and to estimate their workload to decide which should be included in the prototype. Since the author had little experience as a software developer, complicated coding tasks (judged by the author's supervisor) were cut, like a graph-editor for the front-end.

After discussing possible approaches, a few specific tasks were set:

1. Developing a web interface that would:
    - Enable more overview and control for the existing riddles (M)
    - Ease the testing process for new riddles by displaying them dynamically (C)
    - Allow remote access (P) and control (C) within the lokal w-lan viroment
2. Retrieving information from the room must work automatically, therefore should the "feedback mode" be enabled on start-up (M)
3. Providing a thorough documentation for future developers (C/M)

The summarized goals for improvement in the different areas can be seen in figure 4.1.

Prototype

Afterwards, prototyping began. The development process of the prototype is explained in the following sections of this chapter. Proof of Concepts were designed for each stage of the implementation (4.2).

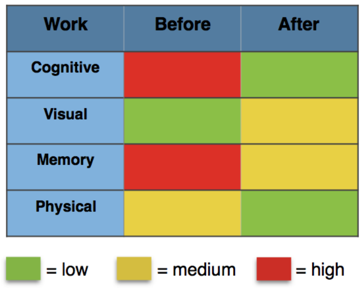

FIGURE 4.1: Workload of the different aspects color-coded

Test

Testing happened regulary at least once a month, later every two weeks. Extensive testing was difficult to achieve, as the room did not provide an internet connection needed to install modules (e.g. Node.js) on the PC and missed basic testing tools like a coding enviroment. Additionaly, due to the room's physical set-up, testing directly on the PC turned out to be inconvenient. The PC was hidden behind a wall and difficult to access, which meant mouse and keyboard could barely reside outside the wall's recess. Consequently, most testing was conducted on two laptops brought by the author: One Macbook Pro from 2009 with OSX 10.11.6, and a Windows laptop with Windows 10 installed. Throughout testing, new features were designed and iterated.

Our final result is a MVP which provides the bare functionalites but lacks in design and more extensive features which we will eleborate in the evaluation.

4.2 Architecture

As happening in other contexts, a Service Orientated Architecture (SOA) approach for middleware was proposed in many IoT - architectures from the last few years [35]. SOA encourages decomposition of a complex system into simpler and isolated components. Thus, reusability and changeability is increased. Especially an IoT-scenario with flexible gateway components and on-going extensions can profit from such architecture as changes are demanded frequently. [35]

The goals were set with that architecture in mind and resulting architecture changes designed to become loosely coupled to encourage improving a specific module of the project.

For example, the TCP- and serial-connection were set-up in a general way (send/ receive all) to separate the data processing from the transport channels. Also, communication was designed not interfere with other each other, e.g. communication to Unity should still work if a connection to the front-end failed. For that reason, a backend was implemented which ensured reliable data storage.

Proof of Concept Steps	Time Needed (hours)	Difficulty (where „+“ = easy)	Efficiency (where „+“ = efficient)
Device:			
Radio communication between two Arduinos	5	-	-
Sending messages to the Gateway	0,5	+	+
Receiving messages from the Gateway printing them to the Serialport	6	o	-
Receiving messages and reacting to them with a physical component dynamically (blink)	1	+	o
1st Gateway (C#):			
Sending and receiving messages via TCP	8	o	-
Sending and receiving messages via Serial	5	+	+
Sending and receiving messages via Socket.io	0,5	+	+
Sending messages from each to each	30	-	-
2nd Gateway (Node.js) :			
Sending and receiving messages via TCP	1	+	+
Sending and receiving messages via Serial	1	+	+
Sending and receiving messages via Socket.io	0,5	+	+
Sending messages from each to each (Async)	20	o	-
Webserver:			
socket.io communication with the web interface	2	+	+
Processing gateway functionalities	20	o	o
Database connection	2	+	+
Database a)select, b)insert, c) update, d) batches	1/2/1/8	o	+
Web interface:			
Drag and Drop functionality	30	-	o
Basic Chat functionality	10	+	+
Pop-Up functionality	5	+	o
Socket.io event triggering (alert)	2	+	+
Database updates a) on event trigger b)on load	3/10	+	o

FIGURE 4.2: List of our PoC steps

In that way, changes reflect the composition of a fog architecture, though there is no cloud available or planned which is an important part of larger IoT project architecture.

The front-end connection received it’s own namespace for Socket.io events so front-end relevant data would only be proccessed in it’s set namespace component.

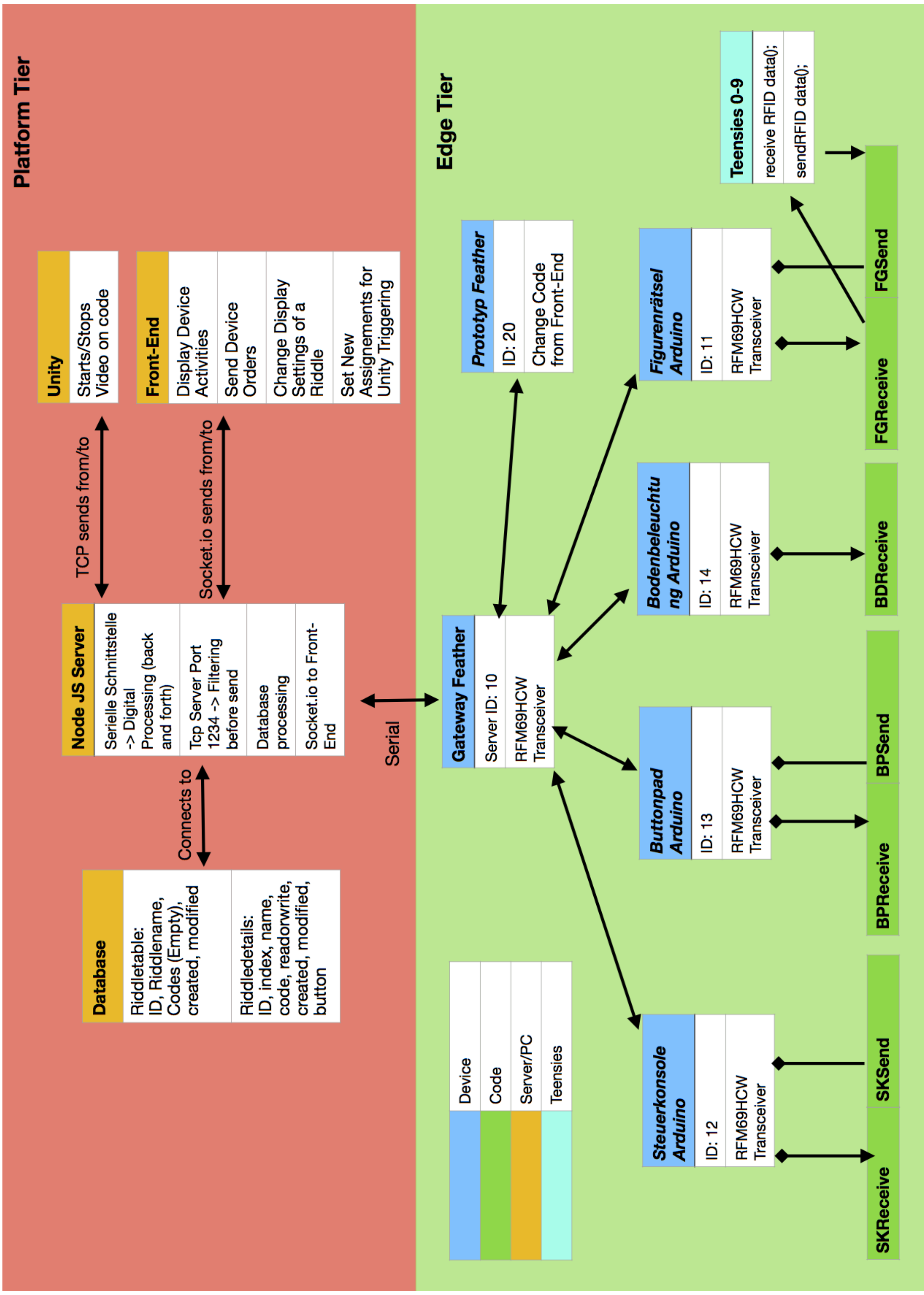

FIGURE 4.3: The new escape room architecture

4.3 Device

Since the room consisted of microcontroller-driven-riddles only at the time of this thesis, we decided to design a prototype and a template for integration of future microcontroller-driven-riddles. The principle concepts though are appliable to any device.

4.3.1 RFM69HCW Wiring

As most riddles are connected to RFM69HCW modules manually, the first set-up of the prototype consisted of an Arduino Uno connected to a RFM69HCW. As one can see in figure 4.4, the wiring looks chaotic.

Later, the set-up was replaced with an Adafruit Feather 32u4 which is a microprocessor with an integrated RFM69HCW module an therefore reduces the complexity of the wiring needed for the riddle. This prototype included an external battery as the Feather doesn't support 5V output. The Adafruit Feather 32u4 needs an external libary to work, the only difference though (apart from the radio set-up) is the assignment of the SPI lines.

The pictures were drawn with a tool called "Fritzing". Fritzing is a popular open-source software for the design of electrical microporcessor-hardware. It and enables prototyping and e.g. generates circuit diagrams automatically and testing code in an interface. To achieve that, someone needs to set the specifications (I/O, behaviour) of a part. The developed part is called a "Fritzing-part" opposed to a "Fritzing-app" which is a project containing several connected Fritzing-parts. Projects can be shared in the Fritzing forum [**fritzingForum**]. It was developed by the University of Applied Sciences in Potsdam

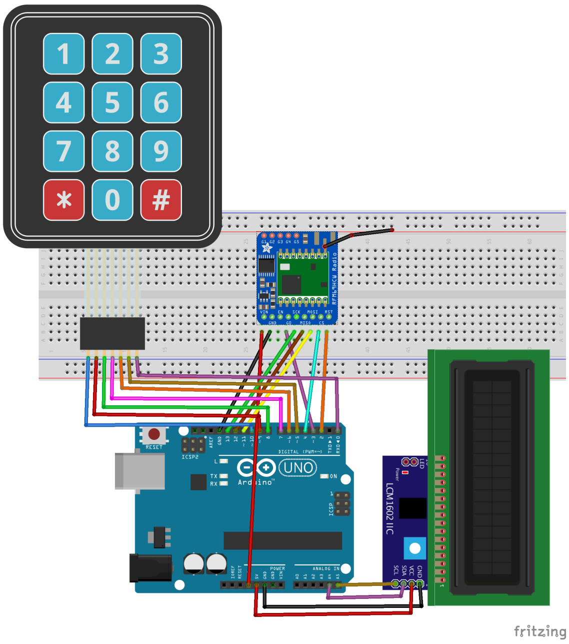

FIGURE 4.4: Prototype, Version 1

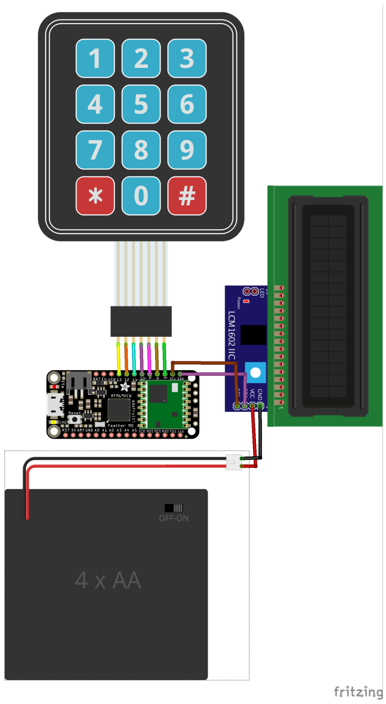

FIGURE 4.5: Prototype, Version 2

4.3.2 Template

The template was designed to simplify the process of developing a riddle. The escape room provided in its prior form no support for new riddle-developers. We decided to modify the existing communication protocol, but had to be careful not to impact communication to the existing riddles.

Still, we wanted simplify the communication system for riddle-developers.

The existing communcation protocol followed a "string-to-chararray-send" and "receive -chararray-to-string" structure that we applied to our implementation. In contrast to the existing puzzles though, we decided to separate the code into several parts, named by the functionality they supplied.

The template is devided into 3 parts: "Groundwork", "Riddlefunctionality" and "Remote Functionality".

Groundwork
Section to be filled with libaries, variables and definitions.

Riddle Functionality
To contain the riddles functionalities separated from nearly any communication. The only communication that needs to be defined there is when the Microcontroller should send messages and which. That is executed by writing a single line command containing the desired string. If the string's value is defined in the "registerRiddle()" function of the "Remote Functionality" section, it will be translated in the web interface.

Remote Functionality
To contain any remote commands for interaction with the web interface and the server. The "Remote Functionality" section consists of two functions:

registerRiddlle()
Here is where strings to be send once on starting the device are defined. These strings set the configuration of the variables in the web interface. To work, they need to follow a specific structure:

1. an Index for the riddlevariable (to order the variables)
2. a "readonly" or "write" command (to make it static or dynamic)
3. the name of the variable (to translate)
4. the value of the variable (needs to be converted into a String)
5. an optional button value (if it was present, a button would show)

That structure is meant to be applieable for any variable.

remoteCommand()

Designed to contain processing of incoming messages from the gateway/server. It's connected to the radio functionality further down in the code, nevertheless allows the user not to care about how the messages are processed.

The developer is advised to use a "Switch-Case" structure to define the microcontroller's reactions to radio messages, to keep the processing clean and standartized. For any reaction concerning the defined variables, the case should match the index of the variable in order for the buttons within the web interface to work.

```
//__________________________________________
//Remote Functionality
//__________________________________________
void registerRiddle(){


rf69send("1/readonly/won/" + won);
delay(3000);
rf69send("2/readonly/lost/" + lost);
rf69send("3/write/user/" + user);
delay(3000);
rf69send("4/write/code/" + code);
//rf69send("riddle 17 just registered");


}
```

FIGURE 4.6: "registerRiddle" definition in the Arduino IDE

The documentation provided explains the template in further detail.

4.3.3 Prototype

For our prototype, an Arduino Uno with a RFM69HCW module, a keypad, and an I2C-display were used. All the parts required specific libaries. The riddles challenge would be to guess a code and enter it. The use case would be that a riddle could to provide different difficulties by adjusting the codes length. Moreover would the riddle possess static "won", "lost", and "reset"-values to track and control the riddle's state.

4.4 Back-End

For this database, PostgreSQL was used as DBMS. PostgreSQL is a light-weight open-source object-relational database system. Companies like Netflix, Spotify or Instagram [36] rely on the flexible database system which allows SQl and noSQL design. It is easy to set-up and maintain.

The relational model was implemented for the back-end. Two tables were enough to fit our needs. One table manages the location, name and other general information about the riddle displayed in the main view, whereas the other one is responsible for saving and editing the information displayed in the pop-up window. The key predicate was the id of the riddle, which was common to both tables. This seperation simplifies database changes, clearifying the tasks happening on the Node.js server. The Node.js server connects the information when sending to the front-end by assigning the details with the riddle's id to the corresponding riddle.

4.5 Comunication

For this project, a websocket implementation seemed the most fitting option as realtime-communication between several clients would be required.

It was decided to use Socket.io for client-middleware-communication. Socket.io is a Javascript-Libary designed for realtime communication build on top of a websocket-protocol. It enables a bi-directional communication channel between client and server and offers a fallback mechanism to long polling when websockets are not available. There are several fallback mechanisms available that are determined dynamically by Socket.io:

- Websocket
- Adobe Flash Socket
- AJAX long polling
- AJAX multipart streaming
- Forever Iframe
- JSONP Polling

The server-side of Socket.io is developed specifically for Node.js wheras for the client different implementations (e.g. .Net, Swift, C++)[37] are available. Once a connections is established it's maintained and uses a diminishing small amount resources to communicate. It uses an event-based system where one participant listens and another emits an event. Both Client and Server can emit and listen for events.

4.5.1 Webserver

It quickly became apperant that the webserver would use Node.js for middleware and processing functionalites between our client front-end and the database-backend due to the reasons mentioned in Chapter 3 and listed below.

Node.js is an open-source, cross-plattform, javascript-runtime-enviroment. With 49.6% it is this years "Most Popular Framework, Libary or Tool" on this years Stackoverflow-survey [38]. According to Google Trends, interest is rising since 2012[39]. One explanation for that might be that it's written in Javascript. The transition for front-end Javascript developers to developing back-end is eased, because they don't have to learn a new language. It uses an event-driven architecture which operates on a single threaded event loop using non-blocking I/O calls. Commands use callbacks to signal they are completed or failed. A downside is, that it doesn't allow vertical scaling by increasing the number of CPU cores of the machine it is running on. On CPU-intensive applications, that might become a problem - but modules like IPC or pm2 can add that functionality [**pm2**]. Node.js commands are non-blocking and execute concurrently or in parallel. It's build on the Google V8 JavaScript engine which compiles Javascript to machine code instead of interpreting it in real time. There are thousands of open-source libaries and web frameworks available for Node.js.

The decisive factor for using Node.js in comparison to other middleware and back-end solutions was that it would be easier to develop and understand a web server written in the same language as the front-end.

As all the website operations were processed on the client-side, Node.js main operation was the database and handling. The "pg-promise" libary [40] was used for database integration with PostgreSQL.

Depending on the event emitted by the web interface, database-queries to select, update or delete entries could be triggered.

If the gateway emitted a message, a control mechanism would check if the riddle was known and either add a new riddle, update an existing riddle (if new variables where recognized) or translate the incoming data.

With Socket.io, the client would register whether a front-end or another client would register and forward the needed data from the database. By namespacing (creating different channels for different clients), we tried to avoid dispensable traffic. The gateway would receive the changeable ("write") values and send them to the connected riddles. The front-end would receive the sorted database data in a sorted json fit to the front-ends data-handling.

4.5.2 TCP/Serial/Socket.IO-Client

This part of the middleware changed several times during our development process.

Since the front-end allowed a reassignement of the messages that would trigger an Unityevent, filtering the incoming serialmessages before they were sent to Unity via TCP was required. Furthermore would they need to be checked for eventual new riddleinformation or messages to be translated or displayed in the front-end. Consequently, this part of the middleware would filter relevant "Finish"-serialmessages through a PostgreSQL-database, and activate a "checkSerialMessage" function which would decide on further processing. First was tried to use the existing C++-server for the architecture but it was quickly discovered that understanding and extending the existing code would probably take longer than recreating the features.

Then, because many developers within the faculty were proficient in C# from Unity development, an implementation the functionalites was tried with a .NET WPF-App. This proved to be difficult as well, as it required multi-threading and communication between the threads. Both are well documented at the MSDN [41], however due to the mass of different techniques it was hard to get an overview. The resulting code was easier than the C++ code, though not comparable to the readability of the Javascript-code of the Node.js implementation. It took roughly 40 hours to implement the desired functionalites with .NET .

Finally, the C# code was revisited and implemented it in Javascript to compare the workload and readability. The same functionalites took about 20 hours to implement, though the author did by then have little experience in Javascript and started the C# implementation with Unity-C#-Knowledge. The async-capabilities of Node.js revealed a tremendous advantage compared to the threading difficulties experienced with the .NET project in this project.

4.6 Front-End

For this project, React.js was used. React.js will further be called by it's commonly referred name "React". React was chosen for it's flexible functionalities and it's big community support that outweighs the other two presented front-end solutions. Angular too has a big community, but supposedly a flatter learning curve. React is an Open-Source Javascript-Libary. After developing React in 2011, Facebook soon discovered that it's performance was faster than other implementations of its' kind [42]and made it Open-Source in 2015. At present, React is used by major companies for their front-end like Airbnb, Netflix and Reddit [43]. In this years' Stackoverflow-Survey, React came third in "Most Popular Framework, Libary or Tool" [38] and is the most popular front-end-framework according to this survey. This year, over 100,000

Challenge	Workload (where „+" = high)	Efficiency (where „+" = efficient)	Tool	How/Why Not? DELETEEEEE
Implement connection from Serial and TCP to Webserver	o	+	Socket.io	Implementation with Socket.Io
New riddles register automatically	+	o	n/n/n/n/n - scheme	If the riddle-device is configured accordingly, the registration works
Send messages from Web Interface to the corresponding riddles	-	+	Riddle-dependant automatic code generation	Messages are build with the riddles information stored in the
Translation	+	-	Database-filtering	Integrated for the new riddle
Optional:				
Database integration	o	+	PostgreSQL, pg-promise library	Database with PostgreSql, updates with queries from the Node.js backend that react to events
Backend sends the updated, changed codes to the corresponding riddle automatically	-	o	Riddle-dependant automatic code generation	When the Node.js backend is started, it will try to send all values with a „readOrWrite" property set to „write" - it doesn't check if it was changed though

FIGURE 4.7: Overview of the back-end challenges

developers participated in the survey. A lot of libaries are available for React. Its main concepts are:

Components

React motivates its users to write encapsulated components with single responsibilities. Components combine the HTML-markup and Javscript-functionality of a responsibility. They are supposed to increase reusability.

Composition

The user can reuse and composite elements as he needs to. The isolated components make code easier to maintain.

Uni-Directional Dataflow

Properties should not be changed in other components, but passed down as read-only variables. React doesn't want children to affect their parent components. That makes maintainability easier, as there's a clear downward structure in a well designed React project. If a user needs to pass changes to a parent component, it's executed with callbacks.

Virtual Dom

A Document Object Model (DOM) is a logical structure of documents in HTML, XHTML, or XML formats. Web browsers are using layout engines to transform or parse the representation HTML-syntax into document object model that we can see in a web browser. Usually, when one of these elements changes, the whole structure has to be calculated again. React uses a Virtual DOM as a negiator to enable calculating only the parts that need calculating. That's also possible because of Reacts' isolated component structure.

JSX

Javascript XML (JSX), extends the ECMAScript JavaScript syntax with XML/HTML-like elements. It is React's recommended language of choice, though React supports standart Javascript syntax too. "The syntax is intended to be used by preprocessors (i.e., transpilers like Babel) to transform HTML-like elements into JavaScript objects that a JavaScript engine will parse" [44]. The JSX-syntax supports React's idea of isolated components, because HTML and Javascript is defined in a single file instead of multiple files. JavaScript functionalities between HTML-code can be used by putting them in curly brackets (""). The generated code runs faster than an equivalent code written directly in JavaScript [45]. Most JSX-components in react follow a specific structure that can be seen in figure 4.8. Usually, a class describing the components functionality is defined. In the class, first, a constructor sets the initial internal values like functions and variables used within the class, second, functions are stated, third, the return-function defines what shall be returned and finally the render-function defines what the returned HTML should look like.

For setting up the front-end, the Create-React-App was used, which provides a front-end build pipeline with Babel and Webpack. React recommends to start there for single-page applications [46].

It provides a package.json file in which modules and their versions are defined and set. This prevents unwanted updates so the existing code won't risk becoming deprecated.

The npm packet manager (which is the standard packet manager for Node.js) automatically generates a package-lock.json file which saves the dependency tree in further detail.

For the file-structure, the recommended approach to group by filetype [47] in combination with the Create-React-App-structure was used.

Starting out, it was planned to implement a node-editor to connect riddles in all thinkable ways. Wheile listing the wished functionalities (Changeable riddleassignments with "Single", "AND" and "OR" connections to the Unity-Events) it was decided that a drag-and-drop table would supply those functionalites (Changeable assignements, OR connections) without creating a difficult User-Interface. The React-dnd libary [48] was used to implement the drag-and-drop functionality in React. Currently, it is not mobile-optimized since that was thought to be the less popular use case, but adding a mobile implementation for the module is possible.

When a user dropped a riddle into a "Video" field (and saved), the riddle's "Finish"-command would be reassigned on input to the corresponding Video-Trigger-command. For example, the "Video1" command was originally triggered by "Riddle1". If a user wanted to make "Riddle2" trigger "Video1", he needed to replace "Riddle1" in the

"Video1"-List with "Riddle2". Whenever "Riddle2" would now signal it's finished, the "Finish"-code of "Riddle1" would be sent to Unity via TCP.

If a Riddle was newly registered, it would be named "NewRiddle" and appear in the "Unassigned Riddles"-List on the web interface. We designed an "Edit"-function which enabled changing the name of the riddle and deleting it in case it got corrupted (or deleted in real-life).

Another aspect was the popup-window for the riddles. It was planned to show enough information, yet keep it simple. Consequently, our layout for the popup-window was designed flexibly to adapt to a desirable output depending on the use-case:

Each variable would be displayed in respect to its in the Arduino defined values. If a variable was set "readonly", but didn't have a button value defined, the information would be listed plainly. If a variable was set "write", but didn't have a button value defined, the information would be listed plainly. Additionaly, an input field would enable changing the defined value and sending it to the Arduino automatically next time the Server would start.

If the microcontroller was programmed to interpret the incoming value, a variable could be changed that way (e.g. a password in a riddle). If a button component was set in a variable, a button would appear instead of plain information about the variable. The user would be able to click the button to send the code immediately to the riddle. This functionality was especially designed with "Finish" and "Start" functionalities in mind, where a supervisor of the escape room might want to trigger these functionalites during a game if customers get stuck.

To increase the general overview for a supervisor, the color of a riddle would change to green once it's "Finish"-code arrived.

```
import React, { Component } from "react";
import { Button } from "reactstrap";

//Small JSX Component exporting a button that changes looks when it's clicked
class EditButton extends Component {
  //Constructor for defining start settings in this.state and binding functions

  constructor(props) {
    //properties given to us by other components are connected with "props"
    super(props);
    //You need to bind a function in the constructor to call it throughout the class
    this.onEdit = this.onEdit.bind(this);

    //This is were our start settings are defined:
    //We want our Edit button to show "off"/false
    this.state = {
      isEditing: false
    };
  }

  onEdit(ev) {
    //New State is set
    this.setState( () => ({ isEditing: !this.state.isEditing }));
  }
    //Here is where our HTML-Markup is designed, in this case just our Edit Button
  render() {
    //The value of isEditing is called from the state
    const { isEditing } = this.state;
    //Our text is called with this.props;
    const{text}= this.props.text;
    //Here starts our HTML, Javascript is marked with "{}" brackets.
    return (
      <div>
        <Button color="info" onClick={this.onEdit}>
          {isEditing ? "Done Editing?" : "Edit"}
        </Button>
        <p>{text}</p>
      </div>
    );
  }
}

export default EditButton;
```

FIGURE 4.8: React example code

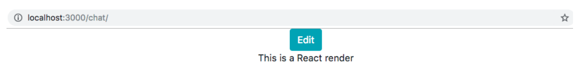

FIGURE 4.9: Resulting Output of example code

Challenge	Workload (where „+“ = high)	Efficiency (where „+“ = efficient)	Tool	How/Why Not? DELETEEEE
GUI	+	o	React	Table view, since it was easier to implement
Which riddle activates which Unityevent?	o	-	React-DND	Managable and can be seen in the View
Show new riddles	o	+	✘	The View shows new riddles automatically
Integrate new riddles in existing structure	o	+	n/n/n/n/n - scheme	New riddles can be assigned to the Unity events just like the older ones
Make new riddles more changeable through the web interface	o	o	assigned „write“ property	If a „write“ property is assigned to a riddledetail, an input field will appear
Debug Window (like the prior C++ view)	-	o	✘	An alternative view with Chat messages is available
Optional:				
Buttons	-	+	✘	If a „button“ property is assigned to a riddledetail, a button will automatically appear for the code
Changes Color when finished	-	o	✘	The riddle appears green when it's finished until the page is refreshed

FIGURE 4.10: Overview about our front-view tasks.

Chapter 5

Evaluation and Conclusion

5.1 Evaluation

To evaluate the quality of the changes we made, we evaluated the room with the same criterias we picked when judging the room before the changes.

5.1.1 Layer Analysis

Refering to Figure 2.3 again,the existing architecture of the escape room in respect to the changes we made was analyzed.

Device Layer
: We didn't make many changes to the Device Layer, as we had instructions not to interfere with the existing structure of the room. Except to adding a prototype implementing the new communication structure, the existing riddles and the gateway Adafruit Feather 32u4 were not manipulated in any way.

Communication Layer
: The Devices do still communicate with RFM69HCW modules and the LowPowerLab-library via radio communication, however, the new webserver functionalities could expand the communication for the software-side of the room. if the server gets a steady network connection. An overview from anywhere within the network could be achieved.

Information Layer
: For newer riddles, an advanced communication protocol was developed where a developer of new riddles can use defined strings to trigger events on the device. As the new system should be compatible with the old one, the basis of the communication architecture was not meant to change completely. Instead, the given system of strings separated by a slash ("/") to trigger more actions was expanded.

Function Layer
: The Function Layer received the most changes. A PostgreSQL database was

added to the architecture for filtering and overview purposes. A middleware-implementation using the database and Socket.io was established to connect to a front-end using React. The incoming strings are now translated and send to a web interface. Depending on the string, they trigger a database-query and are interpreted and saved in the database, or update an existing entry. Additionally, the incoming strings are now filtered before they reach the TCP-client(Unity)by the database to ensure proper trigger assignment.

Process Layer

There is now a web interface showing the existing riddles, enabling an immediate interaction with them, and providing a scalable overview about the room. Unity triggering is still the main objective of the system.

5.1.2 Workload Analysis

The changed amounts of workload were analyzed again. As there wasn't an opportunity to test the findings, the listed views are highly subjective and should only be interpeted as the authors impressions, backed by the research mentioned in Chapter 2.

Cognitive Work

The cognitive work needed to develop parts of the software and hardware is expected to be considerably lower with the changes applied. In the following, prior concerns are compared to our perception now.

1. The developer no longer needs to understand the transport protocol in detail
   (a) He still needs to Set-Up the RFM69HCW with an Arduino or an Adafruit Feather 32u4, which requires drivers and maybe another library, but has thorough instructions through a documentation
   (b) doesn't need to understand the communication LowPowerLab library if he uses the designed template
   (c) doesn't need to look up the other riddles information as he can look most of it up in the WebInterface
2. Understand Arduino-coding
3. Understand working with the "Switch-Case"-communication-protocol used for communication (with instructions)

Process Layer:

1. Understand Javascript to make changes on the server (in a cleaned up, documented and component-based code with improved readability)

2. Understand Javascript and HTML to make changes to the React-front-end (also component-based)
3. There were instructions not to change the Unity application, so the problem of Unity-programming still exists. It would be easier to change though, by replacing the TCP connection with a Socket.io connection. There are multiple client implementations for Unity available [49, 50, 51] That reportedly work (at least) up to the current version of Unity. Since the Unity version in the escape room won't be upgraded regularily, deprecation issues should not arise.

Visual Work

There was little time to research web-design to an extend where the author could state confidentely that the project's design is especially user-friendly. Alternatively though, the website offers an improved version of the prior window with an integrated translation of the incoming codes. Instead of coded strings, the strings are now depicted as readable messages if they are stored in the database. It also offers a lot more information and possibilites to interact with the enviroment, which reduces the workload in other areas.

Memory Work

The memory work is reduced immensely, especially for new riddles. The developer has the possibility to save its values as buttons so that he only needs to interact with an abstraction of the machine-code.

Buttons
: By clicking buttons instead of remembering the strings, the developer can take the shortest way to activate a functionality

Information Display
: By displaying all riddle information in one place, the developer can gain a better understanding of the riddles functionalities

Automatisation
: Tasks like activating the feedback mode of the riddles and sending and starting the applications in the right order were automated.

Physical Work

The web server reduced the physical interaction needed to work with the room silghtly. A user can access the room remotely and doesn't need to start the applications manually.

Challenge	Before	After
Cognitive:		
Device Layer:		
Set-Up the RFM69HCW with an Arduino	~	✔
Set-Up the RFM69HCW with an Adafruit Feather 32u4	~	✔
Understand the library commands	✘	✔
Researching other riddles information	✘	~
Writing Arduino code	~	~
Working with the communication protocol	✘	✔
Process and Function Layer:		
Changing or extending the back-end	✘	~
Changing the front-end	✘	~
Unity-Integration	~	~
Visual:		
Clearity of the visual interface	✔	~
Memory:		
Translation	✘	~
Code-memorization	✘	~
Start-up Settings	~	✔
Physical:		
Physical work required	~	~

FIGURE 5.1: Overview about the workload analysis

5.1.3 Limitations

Even though we, to our judgement, achieved our set goals, we should mentions the limitations our project might have suffered from.

We were working with an already existing project which we were instructed not to change from its core. Instead of designing a new architecture by scratch, we acted therefore within the given frame. An architecture from scratch would e.g. have used a simpler communication system for the Arduinos and would have changed the Unity communication to an event-based, more dynamic protocol.

Due to the current Unity communication, resetting the riddles depending on their assignement is currently not possible, and riddles might be involuntary resetted on activating a trigger in Unity.

The code of the project was build within a 3-month period by the single, unexperienced author. The most thorough research can not replace hands-on experience, just like the most motivated person can usually not substitute the intertwined ideas a team can develop over a course of time.

Because of the tight schedule, there wasn't time to evaluate the user-experience with the new system so the value the framework produces can only be estimated. Also, lots of features are still missing, listed in the "Future Work"-section below. A more finished product would have been desirable.

5.1.4 Future Work

This section is meant to list possible ways to expand the built framework.

Fully Event-Based-Processing

By replacing the TCP-Connection to Unity with a Socket.io client, one could take the entire processing to the Node.js server. Since the Node.js server handles the database-queries, the events would scale dynamically. Developers wouldn't need to change the Unity code manually everytime a riddle is added. Possible use cases are:

1. Resetting the room could be implemented by selecting all riddle details with "Finish" as an infovalue when Unity sends a "resetRiddles" event.
2. Resetting the riddle that activated an event in Unity could be implemented by storing the incoming "Finish" value and retrieving its fitting "Reset" value with the ID of the string.

Node Editor

Developing the front-end further, one could implement a node-editor to create reaction-chains. Such a reaction chain could be that a riddle changes the color of the floor when it's finished by combining two buttons. The data already exists as JSON in the front-end, but creating a flexible drag-and-drop interface with the internal processing went beyond the scope of the project.

Security

Within the scope of this project, security protection like password authentication and encryption was not included. Since the web server is hosted in a local network in a password protected enviroment we didn't prioritize an authentication system. Another aspect was that an escape room doesn't contain privacy sensitive data in our point of view.

Component Isolation

Though it was tried to keep the components separated, there is always room for improvement. Especially the Node.js middleware shows room for further splitting and transparency.

Gateway Configuration

As changes didn't affect the communication protocol, scalability issues might arise depending on the number of incoming messages at the gateway component, especially in combination with the current serialport communication. The serial port's UART is running with 9600 baud (which represent the bits per second communicated), 960 byte/s, whereas the RFM69HCW is able to send 300kb/s. That's an avoidable bottleneck, if UART were replaced with SPI. Alternatively, the baud rate could be increased. A Raspberry Pi could be set up as a server with a Wi-Fi module. There is an regulary updatet (last update is 6 months old in a branch) python wrapper [52] with thorough documentation [53] available for connecting a Raspberry Pi to a RFM69 module running with the LowPowerLab library. The Raspberry could host the websever in a local, protected Wi-Fi environment and supply the server hosting Unity with the needed events within the closed network. Ideally, Unity would have an implemented Socket.Io connection by then. This way, only authenticated users would have access to the escape rooms properties, the architecture would be separated and therefore clearer for developers, and easier to change. [53]

Mobile integration

With the webserver running, mobile devices like cellphones or tablets could communicate with the server via Socket.io. This introduces a whole new world riddles that could interact with the cellphone. One could build an alternative front-end for customers, and e.g. let them scan something with their camera trough the webpage they access. The database could provide further information to a riddle, if e.g. a code should be adressed.

Smart Capabilities

A smart room with more riddles and tracking functionalities would be gratifying. Immersion in an escape room can be increased by tracking body functions, One could track body functions like heartbeat when controlling the spaceship to start individual audio messages (approval, motivation, disapproval) enviromental factors like the temperatue of the room could impact the story telling ("Brrr It's cold in here"/"It's getting hot in here" - audiomessages)

//Picture of Planned architecture (Raspberry, webserver to unity bla)

5.1.5 Big Picture

In the context of other IoT implementations, a rather unusual approach was taken, reasoned with the initial set-up of the room. Researching, little IoT-implementations apart from the LowPowerLab framework were found with the RFM69HCW module. Though it serves it's purpose well, beeing cost-effective, consuming very little power and beeing fairly reliable, other communication protocols would have been easier to implement and are more flexible in the long-run. For example a related module which costs 5 euros more, the RFM9X, is becoming popular with the rise of LoRa-WAN in IoT technology. LoRa-WAN enables easy access to a Web-Interface with The Things Network (TTN) [**TTN**]. It would also simplify testing, as any "Thing" integrated in TTN can access the gateway within a 2-5 mile range within a city. Protocols like Zigbee and LowPower-Bluetooth are an alternative for short-range communication and offer other advantages like ZigBees mesh-communication or BLE's easy cellphone communication. More popular protocols offer more support by a community and are therefore easier to access and update. That might be on the one hand due to the we supplied little hardware extensions.

5.2 Conclusion

This project extended the functionalities of an existing escape room. A framework which might help future developers in integrating new riddles was developed. We hoped to set a basis for a smart escape room with the new features.

Struktur: "Why should anybody care"

mobile integration would be possible by accessing the node.js server via socket.io Erster schritt richtung smart escape room

Narration gets more immersive if a room can react to enviromental factors with sensors. Body functions could be tracked for specific tasks, while the room could react to the general status of the room. If the room "knows" how many riddles were done, the temperature of the room, the number of people within the room, where everybody is with motion or touch detection, it can react appropriately. Currently, escape rooms need a supervisor to send hints to people within the room. A fully developed smart escape room.

Bibliography

[1] Gartner. *Gartner Predicts 20.4bn Connected 'Things' by 2020*. URL: Feb.%209, %202017..

[2] Verizon. *2017 State of the Market: IoT Report*. URL: https://www.verizon.com/about/sites/default/files/Verizon-2017-State-of-the-Market-IoT-Report.pdf (visited on Nov. 18, 2018).

[3] Hasso Plattner, Christoph Meinel, and Larry J. Leifer. *Design thinking: understand, improve, apply. Understanding innovation*. Springer-Verlag, 2011. URL: doi:10.1007/978-3-642-13757-0.

[4] Rikke Dam and Teo Siang. *Design Thinking: A Quick Overview*. URL: https://www.interaction-design.org/literature/article/design-thinking-a-quick-overview (visited on Nov. 24, 2018).

[5] Hasso Plattner Institute of Design at Stanford University. *A Virtual Crash Course in Design Thinking*. URL: https://dschool.stanford.edu/resources-collections/a-virtual-crash-course-in-design-thinking (visited on Nov. 24, 2018).

[6] Scott Doorley et al. *The Design Thinking Bootleg*. URL: https://static1.squarespace.com/static/57c6b79629687fde090a0fdd/t/5b19b2f2aa4a99e99b26b6bb/1528410876119/dschool_bootleg_deck_2018_final_sm+%282%29.pdf (visited on Nov. 24, 2018).

[7] Amy Hackney Blackwell and Elizabeth Manar, eds. *Prototype*. Farmington Hills, MI, Nov. 2015. URL: http://link.galegroup.com/apps/doc/ENKDZQ347975681/SCIC?u=dclib_main&sid=SCIC&xid=0c8f739d.

[8] S. Hodges et al. "Prototyping Connected Devices for the Internet of Things". In: *Computer* 46.2 (Feb. 2013), pp. 26–34. ISSN: 0018-9162. DOI: 10.1109/MC.2012.394.

[9] WebFinance Inc. *Proof of Concept*. 2018. URL: http://www.businessdictionary.com/definition/proof-of-concept.html (visited on Nov. 24, 2018).

[10] Techopedia. *Minimum Viable Product (MVP)*. URL: https://www.techopedia.com/definition/27809/minimum-viable-product-mvp (visited on Nov. 27, 2018).

[11] Pallavi Sethi and Smruti R. Sarangi. "Internet of Things: Architectures, Protocols, and Applications". In: *J. Electrical and Computer Engineering* 2017 (2017), 9324035:1–9324035:25.

[12] Rafiullah Khan et al. "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges". English. In: *2012 10th International*

Conference on Frontiers of Information Technology (FIT): Proceedings. Institute of Electrical and Electronics Engineers Inc., 2012, pp. 257–260. ISBN: 978-1-4673-4946-8. DOI: 10.1109/FIT.2012.53.

[13] Ibrahim Mashal et al. "Choices for interaction with things on Internet and underlying issues". In: *Ad Hoc Networks* 28 (2015), pp. 68–90. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2014.12.006. URL: http://www.sciencedirect.com/science/article/pii/S1570870514003138.

[14] Jayavardhana Gubbi et al. "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions". In: *CoRR* abs/1207.0203 (2012). arXiv: 1207.0203. URL: http://arxiv.org/abs/1207.0203.

[15] I. Stojmenovic and S. Wen. *The fog computing paradigm: scenarios and security issues*. Tech. rep. Warsaw, Poland: Proceedings of the Federated Conference on Computer Science and Information Systems (Fed- CSIS '14), pp. 1–8, IEEE, Sept. 2014.

[16] F. Bonomi et al. "Fog computing and its role in the internet of things". In: *in Proceedings of the 1st ACM MCC Workshop on Mobile Cloud Computing, pp. 13–16,* 2012.

[17] 2018. URL: https://www.techopedia.com/definition/3799/front-end-system (visited on Nov. 28, 2018).

[18] TechMagic. *ReactJS vs Angular5 vs Vue.js - What to choose in 2018?* URL: https://link.medium.com/RCl9Ku3N1R (visited on Nov. 21, 2018).

[19] Dec. 2018. URL: https://restfulapi.net/.

[20] Thilina Ashen Gamage. *HTTP and Websockets: Understanding the capabilities of today's web communication technologies*. URL: https://medium.com/platform-engineer/web-api-design-35df8167460.

[21] *jQuery Doc - Ajax*. URL: http://api.jquery.com/category/ajax/.

[22] D. G. Puranik, D. C. Feiock, and J. H. Hill. "Real-Time Monitoring using AJAX and WebSockets". In: *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. Apr. 2013, pp. 110–118. DOI: 10.1109/ECBS.2013.10.

[23] Cambridge University Press. *Back-End*. URL: https://dictionary.cambridge.org/dictionary/english/back-end (visited on Nov. 27, 2018).

[24] Oxford University Press. *Middleware*. URL: https://en.oxforddictionaries.com/definition/middleware (visited on Nov. 27, 2018).

[25] TechEmpower. *Web Frameworks Benchmarks*. URL: https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=plaintext&p=zik0zj-zijocf-zijocf-5m9r (visited on Nov. 21, 2018).

[26] Blynk. *Blynk*. URL: https://www.blynk.cc/ (visited on Nov. 27, 2018).

[27] *The Things Network*. URL: https://www.thethingsnetwork.org/.

[28] Amazon. *Amazon IoT developer book*. URL: https://docs.aws.amazon.com/de_de/iot/latest/developerguide/what-is-aws-iot.html.

[29] Microsoft. *Support additional protocols for IoT Hub*. URL: https://docs.microsoft.com/de-de/azure/iot-hub/iot-hub-protocol-gateway.

[30] lady ada. *Radio Range F.A.Q.* URL: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts.pdf (visited on Nov. 19, 2018).

[31] *LowPowerLab Libary*. URL: https://lowpowerlab.com/.

[32] *Radiohead Libary*. URL: https://www.airspayce.com/mikem/arduino/RadioHead/.

[33] Adafruit. *Using the RFM69HCW Radio*. URL: https://learn.adafruit.com/adafruit-feather-32u4-radio-with-rfm69hcw-module/using-the-rfm69-radio.

[34] Kim Goodwin. *Designing for the Digital Age: How to Create Human-Centered Products and Services*. Wiley Publishing, Inc., 2009.

[35] L. Atzori et al. "The Internet of Things: A survey". In: *Computer Networks, Elsevier* (Oct. 2010).

[36] Postgres. *Postgres User*. URL: https://stackshare.io/postgresql (visited on Nov. 20, 2018).

[37] Socket.io. *Other Client Implementations*. URL: https://socket.io/docs/ (visited on Nov. 20, 2018).

[38] StackOverflow. *Developer Survey Results 2018*. URL: https://insights.stackoverflow.com/survey/2018#technology (visited on Nov. 20, 2018).

[39] Google. *Google Trends Node.js*. URL: https://trends.google.com/trends/explore?q=nodejs&date=all (visited on Nov. 20, 2018).

[40] vitaly-t. *pg-promise*. URL: https://github.com/vitaly-t/pg-promise (visited on Nov. 20, 2018).

[41] Microsoft. *Thread Class*. URL: https://docs.microsoft.com/de-de/dotnet/api/system.threading.thread?redirectedfrom=MSDN&view=netframework-4.7.2 (visited on Nov. 28, 2018).

[42] URL: https://stefankrause.net/js-frameworks-benchmark8/table.html.

[43] stackshare. *Companies using React*. URL: https://stackshare.io/react (visited on Nov. 21, 2018).

[44] URL: https://www.reactenlightenment.com/react-jsx/5.1.html.

[45] *JSX*. URL: https://jsx.github.io/.

[46] React. *Create React App*. URL: https://reactjs.org/docs/create-a-new-react-app.html (visited on Nov. 20, 2018).

[47] React. *File Structure*. URL: https://reactjs.org/docs/faq-structure.html (visited on Nov. 20, 2018).

[48] *React DND*. URL: https://react-dnd.github.io/react-dnd/about (visited on Nov. 20, 2018).

[49] Fabio Panettieri. *Socket.IO for Unity*. URL: https://assetstore.unity.com/packages/tools/network/socket-io-for-unity-21721 (visited on Nov. 22, 2018).

[50] floatinghotpot. *socket.io-unity*. URL: `https://github.com/floatinghotpot/socket.io-unity` (visited on Nov. 22, 2018).

[51] smallmiro. *socket.io-client.unity3d*. URL: `https://github.com/nhnent/socket.io-client-unity3d` (visited on Nov. 22, 2018).

[52] Eric Trombly. *Python RFM69 library for RaspberryPi*. URL: `https://github.com/etrombly/RFM69` (visited on Nov. 22, 2018).

[53] *RPI RFM69 documentation*. URL: `https://rpi-rfm69.readthedocs.io/en/latest/` (visited on Nov. 23, 2018).