HOCHSCHULE DÜSSELDORF

BACHELOR THESIS

Development of an Integration Platform for IoT Devices

Author:
CaraWATERMANN

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

November 24, 2018

Declaration of Authorship

I, CaraWATERMANN, 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:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

Acknowledgements

The acknowledgments and the people to thank go here, don’t forget to include your project advisor…

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 (*Gartner Predicts 20.4bn Connected 'Things' by 2020*) by 2020 in nearly every industry.

According to Verizon (*2017 State of the Market: IoT Report*) 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 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

Research 10P

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

2.1 Design thinking

Design thinking is one strategy to plan an innovation process. It seemed to fit our working process and was a guide concerning our production phase, further explained in 4. 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." (*Design Thinking: A Quick Overview*) The Stanford Design School, now known as the Hasso Plattner Institute of Design began teaching a design thinking process with the thre steps of understanding, improving and applying a product.

Since then, their approach to design thinking moved on to a widely used, open-sourced 5 stage process (*A Virtual Crash Course in Design Thinking*) consisting of the following items:

Empathise Emphatizing 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 (*The Design Thinking Bootleg*)

Ideate Ideation is the stage one should explore ideas in a "wide open" (*The Design Thinking Bootleg*). 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 (*The Design Thinking Bootleg*), 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). Empathy gaining.

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.

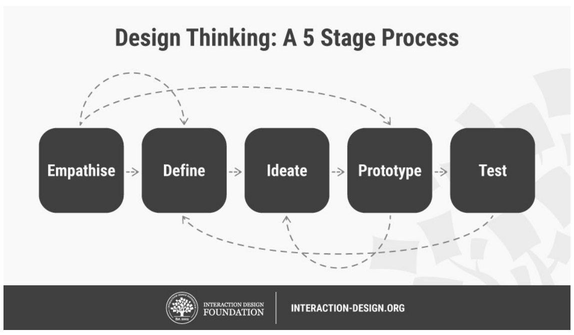

Design Thinking: A 5 Stage Process
Empathise
Define
Ideate
Prototype
Test
INTERACTION DESIGN FOUNDATION
INTERACTION-DESIGN.ORG

FIGURE 2.1: Author/Copyright holder: Teo Yu Siang and Interaction Design Foundation. Copyright terms and licence: CC BY-NC-SA 3.0

The model goes on to describe user analyzing methods which are not relevant in the context of this project.

2.2 Prototyping (1-2p)

A prototype is "An initial model of an object built to test a design." (Blackwell and Manar, 2015) Basic vocabulary will be explained in this section for following chapters.

2.2.1 Proof of Concept

2.2.2 Minimal Viable Product

2.2.3 Rapid Prototyping

2.3 Architecture (2-3p)

//IoT-system, architectures, layers, models, fog/cloud computing //Sensors, actuators

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 Five-Layer architecture

One by (Khan et al., 2012; Mashal et al., 2015) 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 architecture (Gubbi et al., 2012) 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 (Stojmenovic and Wen, 2014; Bonomi et al., 2012) 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**)

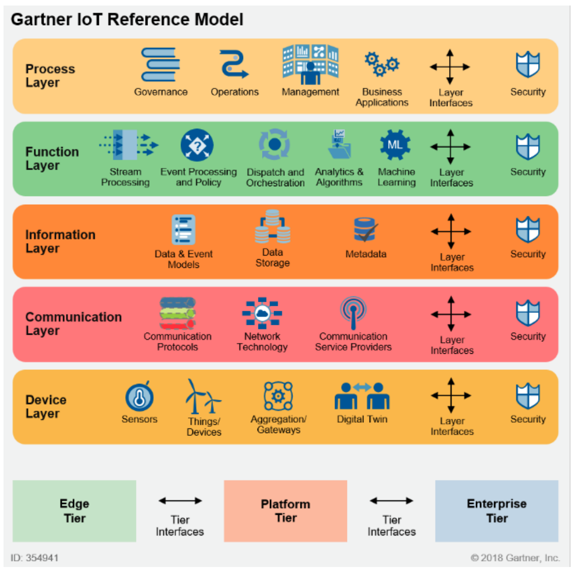


FIGURE 2.2: gartnerIoT1

2.4 Communication Protocols

The project didn't allow changing the communication protocol without disrupting the existing enviroment. For future implementations, other communication protocols could be considered for enviroments like this (small room, fast transport).

2.4.1 W-LAN

2.4.2 WPan6

2.4.3 ZigBee

2.4.4 MQTT

2.5 Front-End

As (**frontend**) 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 that combine HTML and Javascript capabilities like React.js, Angular or Vue have been designed to support a component based architecture.

For this project, we used React.js.

Reactjs is an Open-Source Javascript-Libary. After developing Reactjs in 2011, Facebook soon discovered that it's performance was faster than other implementations of its' kind and made it Open-Source in 2015. At present, React is used by major companies for their front-end like Airbnb, Netflix and Reddit (*Companies using React*). In this years' Stackoverflow-Survey, React came third in "Most Popular Framework, libary or Tool" (*Developer Survey Results 2018*). This year, over 100,000 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 reusability.

Composition
The user can reuse and composite elements as he needs to. The isolated components makes 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, or, for more complicated architecture with a Flux-supporting libary like Redux.

Virtual Dom
A DOMis 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 JSX, short for Javascript XML, is an implementation of Javascript which is usually used to write in ReactJs. It looks a lot like HTML but enables Javascript functionality. Javascript functionalities can be used by putting them in curly brackets ("").

2.6 Communication

There are different ways to communicate between clients and middleware. Communication on the web is usually unsynchronized. The client requests and the server responds. Problems arise if real-time communication is required. Chat-Applications shouldn't require the user to reload a page anytime he wants to see a new message. Those services demand a more reactive and immediate communication.

For this, web development techniques like AJAX (**ajax**) were invented. Here, the client requests data automatically, establishing a new connection to the server.

This technique needs a client to request new content instead of listening for new information which creates unnessecary overhead and a higher latency than other communication enviroments like WebSockets.

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. Now-a-days, most browsers are Websocket compatible.

For this project, Socket.io was used 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++)(*Other Client Implementations*) 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.

2.7 Middleware

//Was ist middleware? in dme fall node js 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 (*Developer Survey Results 2018*). According to Google Trends, interest is rising since 2012(**gogleTrendNode**). One explanation for that might be that it's written in Javascript. The transation for front-end Javascript developers to developing back-end is eased, which saves companies learning costs. Node.js is estimated to have a high learning curve (**nodeLearningcurve**).

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. 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. That makes it faster than (some) other engines. There are thousands of open-source libaries and web frameworks available for Node.js.

2.8 Device

//Was ist ein device im iot kontext? die microcontroller 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.

loop()

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

2.9 Back-End

//Back end was ist das in diesem fall wurde das benutzt Postgres is a light-weight open-source object-relational database system. Companies like Netflix, Spotify or Instagram () rely on the flexible database system which allows SQl and noSQL design. It is easy to set-up and maintain.

Chapter 3

Project Overview5-10p

3.1 Analysis

3.1.1 Layer Analysis

Refering to 2.2, we analyzed the existing architecture of the escape room.

Device Layer

The Device Layer consisted of different Arduinos as Actuators, and riddle-depending sensors for each riddle. The gateway device was an Adafruit Feather 32u4.

Communication Layer

The communication within the room was set-up with RFM69HCW Transceiver modules.(*Radio Range F.A.Q.*) The RFM69HCW is very cheap and easy to use and setup Transceiver. They do packetization, error correction and auto-retransmit which makes them easier to handle than other communication systems. They are designed for point-to-multipoint communication with one Transceiver set-up 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 (**LowPowerlab**) and Radiohead (**Radiohead**) libary. The escape room used the LowPowerlab libary since the architect was familiar with the libary. Now-a-days the Radiohead libary is the recommended libary since it's documented more thorough and kept up-to-date. The gateway transceiver is connected to a PC through the Serial Port. Any node is recognized by a different nodeID and the Gateway with a specified GatewayID. For security-reasons, password-encryption is used between the nodes. From there, a C++-Server would forward those messages to any TCP-Client listening. The server would forward any messages coming from the TCP-Client back to the serialport just as well.

Information Layer

The Information Layer was build around a simple event model: the Transceivers would send codes in a "Number/Number/Number..."-structure. A node could be identified through the string it sent, since the first number would be the node's adressing number. The other numbers would represent the current state of the Arduino with a "Switch-Case" scenario 3.2.

Function Layer

If the Code matched with the Tcp-Clients list (in our case, in a Unity application), a Video in Unity would be played and Unity would send a "reset" code to the matching riddle. Both lists were defined statically.

Process Layer

Apart form the main functionality, offered the C++-Server a communication window where serialport-messages could be seen and sent manually. 3.3.

3.1 provides further insight into the architecture

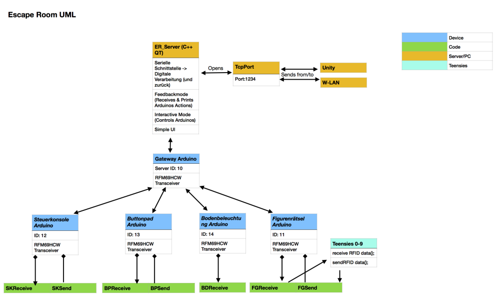


FIGURE 3.1: The old escape room structure.

FIGURE 3.2: Example of the messages defined in the Arduino

FIGURE 3.3: Example of the messages defined in the Arduino

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“ (Goodwin, 2009) While an IoT-System is not always interactive, the goal of this thesis was very interaction-orientated: We wanted to help engineers expand the existing room which would require interaction with the Device-Layer (if they wanted to add hardware-functionality) and the Process-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 user, and the four areas of cognitive, visual, memory and physical work involved for an escape room customer would already be pretty low.

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 user 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 libary
   (b) Understand the communication within the LowPowerlab-libary
   (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 get 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 has to search for features in a product visually. The given interface was clear, but lacked in buttons for often used functionalities (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 it's current state, the architecture demanded a high amout of memory work from a developer:

1. The developer has to translate "Number/Number/Number..." 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.

3.2 Consequences

This analysis showed the projects strengths and weaknesses. The next step was to estimate the workload and possibilities we had with our limited resources to improve the project for future developers. As mentioned earlier, IoT-Projects typically follow a SoA to keep the project flexible and expandable.

We orientad our goals with that architecture in mind and planned to keep whatever we implement loosely coupled to encourage improving a specific module of the project.

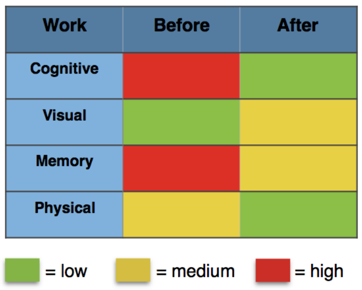
The cognitive and memory work required from a developer were the most critical points that one might spend a lot of time on, or might decide not to join the project at all. As a consequence, we looked for ways to reduce those workloads. To reduce cognitive (C) work, the process of learning and discovering 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.

After discussing possible approaches, we decided on a few specific tasks:

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 enviroment
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)

We summarized our goals for improvement in the different areas in 3.4.



Work	Before	After
Cognitive	🟥	🟩
Visual	🟩	🟨
Memory	🟥	🟨
Physical	🟨	🟩

🟩 = low 🟨 = medium 🟥 = high

FIGURE 3.4: Workload of the different aspects color-coded

Chapter 4

Implementation 10

After introducing the project, we will now talk about our implementation. Our implementation used the aforementioned tools as well as some smaller libaries that will be explained in the following section.

4.1 Architecture

We tried to integrate a SoA, so that our components would be self-contained and reusable. For example, the communication to Unity should still work if a connection to the webserver failed.

The Tcp and Serial connection whre set-up in a general way (send/receive all).

As a consequence, the processing of the data would happen in another component.

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.

4.2 Prototyping

Our reasearch inspired us to try Rapid Prototyping in our project. We designed a Proof of Concept for each stage of our implementation (4.1).

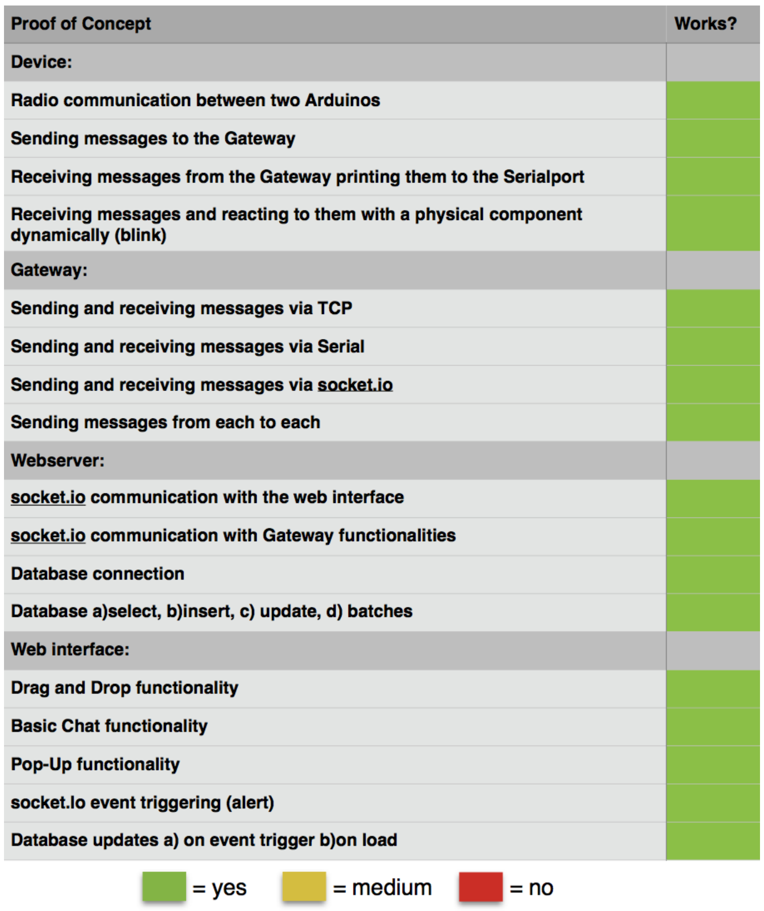
Our final result is a MVP which provides the bare functionalites but lacks in design and more extensive features which we will eleborate in the evaluation.

4.3 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.

4.3.1 Template

The template was designed to simplify the process of developing a riddle. The escape room provided in it's prior form no support for new riddle-developers. We



Proof of Concept	Works?
Device:	
Radio communication between two Arduinos	🟩
Sending messages to the Gateway	🟩
Receiving messages from the Gateway printing them to the Serialport	🟩
Receiving messages and reacting to them with a physical component dynamically (blink)	🟩
Gateway:	
Sending and receiving messages via TCP	🟩
Sending and receiving messages via Serial	🟩
Sending and receiving messages via socket.io	🟩
Sending messages from each to each	🟩
Webserver:	
socket.io communication with the web interface	🟩
socket.io communication with Gateway functionalities	🟩
Database connection	🟩
Database a)select, b)insert, c) update, d) batches	🟩
Web interface:	
Drag and Drop functionality	🟩
Basic Chat functionality	🟩
Pop-Up functionality	🟩
socket.Io event triggering (alert)	🟩
Database updates a) on event trigger b)on load	🟩

🟩 = yes 🟨 = medium 🟥 = no

FIGURE 4.1: List of our PoC steps

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:

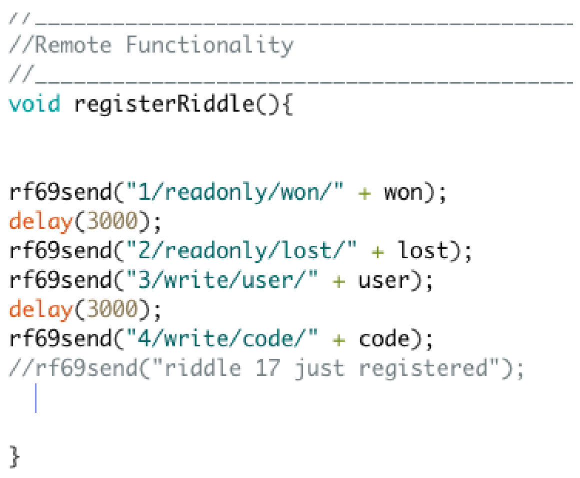
registerRiddlle() Here is where strings to be send once on starting the device are defined. These strings set the configuration of the variables in the web interface. To work, they need to follow a specific structure:

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/PC. 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 4.2: "registerRiddle" definition in the Arduino IDE

The documentation we provided explains the template in further detail.

4.3.2 Prototype

For our prototype, we used an Arduino Uno with a RFM69HCW module, a keypad, and an I2C-display. 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

We used a relational model to manage our database. In the relational model, related records are linked by a key predicate common to all. In our project, we found that two tables 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. 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 Middleware

4.5.1 Web Server

It quickly became apperant that our middleware web server would use Node.js due to the reasons mentioned in 3. The decisive factor for this decision was that it would be easier to develop and understand a web server programmed in the same language as the front-end.

As we did all website operations client-side, Node.js main operation was the database and handling. We used the "pg-promise" libary (*pg-promise*) for our database integration with postgres.

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

If the gateway emittet 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

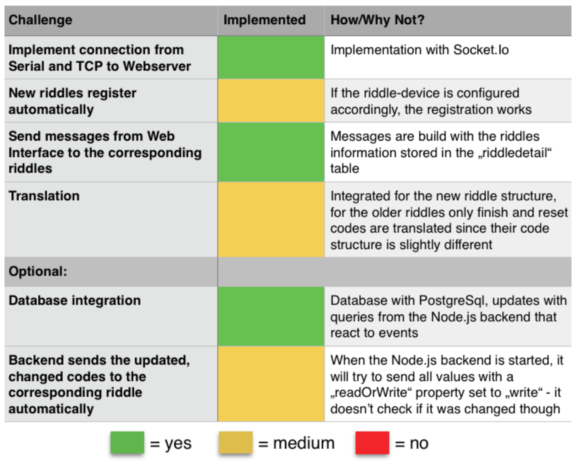
This part of the middleware changed several times during our developing process. Since the front-end allowed a reassignement of the messages that would trigger an

Unityevent, a middleware implementation was needed. We decided to filter the relevant "Finish"-serialmessages through a PostgreSQL-database, which is compatible with different languages and frameworks. Additionaly, we needed to implement a connection from the serial port to our socket.io connection.

First, we tried to use the existing C++-server for our architecture but discovered quickly that understanding and extending the existing code would probably take longer than recreating the features.

Then, because many developers of our target group were proficient in C# from Unity development, we tried to implement the functionalites with a .NET WPF-App. This proved to be difficult, as it required multi-threading and communication between the threads. Both are well documented at the MSDN (**MSDN**), however due to the mass of different techniques it was hard to get an overview. The code we created was easier than the C++ code, though not comparable to the readability of the Javascript-code. It took us roughly a week to implement the desired functionalites.

Finally, we decided to revisit our C# code and tried to implement it in Javascript. The same functionalites took us about 6 hours to implement, though we did by then have intermediate experience in Javascript and started our C# implementation with Unity-C#-Knowledge. In our perception, the async-capabilities of Node.js showed a tremendous advantage compared to the threading difficulties wie experienced with the .NET project.



Challenge	Implemented	How/Why Not?
Implement connection from Serial and TCP to Webserver	🟩	Implementation with Socket.Io
New riddles register automatically	🟨	If the riddle-device is configured accordingly, the registration works
Send messages from Web Interface to the corresponding riddles	🟩	Messages are build with the riddles information stored in the „riddledetail“ table
Translation	🟨	Integrated for the new riddle structure, for the older riddles only finish and reset codes are translated since their code structure is slightly different
Optional:		
Database integration	🟩	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	🟨	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

🟩 = yes 🟨 = medium 🟥 = no

FIGURE 4.3: Overview about our front-view challenges.

4.6 Front-End

For setting up our front-end, we used the Create-React-App which provides a front-end build pipeline with Babel and Webpack. React recommends to start there for single-page applications (*Create React App*).

It provides a package.json file where which version of a module one wants to use is defined. This prevents unwanted updates so the existing code won't risk becoming deprecated.

The npm installer (which is the standard installer for React and Node.js) automatically generates a package-lock.json file which saves the dependency tree in further detail. We discovered GIT-difficulties with the modules and were thankful that we could reinstall the needed packages without having to search which modules we needed.

For our file-structure, we used the recommended approach to group by filetype (*File Structure*) in combination with the Create-React-App-structure.

Starting out, it was planned to implement a node-editor to connect riddles in all thinkable ways. When we listed our wished functionalities (Changeable riddle-assignments with "Single", "AND" and "OR" connections to the Unity-Events) we decided that a drag-and-drop table would supply those functionalites (Changeable assignements, OR connections) without creating a difficult User-Interface. We used the React-dnd libary (*React DND*) to implement the drag-and-drop functionality in React. Currently, it works only on PC since we thought that would be a more 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.

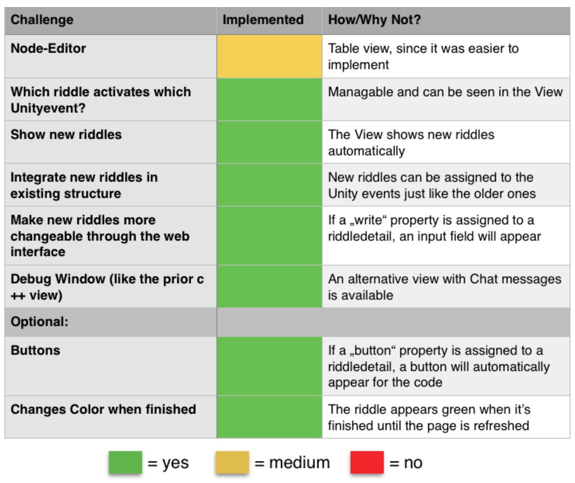
If a Riddle was newly registered, it would be named "NewRiddle" and appear in the "Unassigned Riddles"-List on the web interface. We designed an "Edit"-function which enabled changing the name of the riddle and deleting it in case it got corrupted (or deleted in real-life).

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 it's 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.



Challenge	Implemented	How/Why Not?
Node-Editor	🟨	Table view, since it was easier to implement
Which riddle activates which Unityevent?	🟩	Managable and can be seen in the View
Show new riddles	🟩	The View shows new riddles automatically
Integrate new riddles in existing structure	🟩	New riddles can be assigned to the Unity events just like the older ones
Make new riddles more changeable through the web interface	🟩	If a „write“ property is assigned to a riddledetail, an input field will appear
Debug Window (like the prior c ++ view)	🟩	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	🟩	The riddle appears green when it's finished until the page is refreshed

🟩 = yes 🟨 = medium 🟥 = no

FIGURE 4.4: Overview about our front-view tasks.

Chapter 5

Evaluation and Conclusion 10-15P

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 2.2 again, we analyzed the existing architecture of the escape room in respect to the changes we made.

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-libary via radio communication, however, the new web-server functionalities could expand the communication for the software-side of the room. If the Server-PC gets a steady W-Lan connecection. An overview from anywhere within the network could be achieved.

Information Layer

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. For newer Riddles, an advanced communication protocol was developed where a user 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,we expanded the given system of strings separated by "/" to trigger more actions.

Function Layer

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

The process layer received the most changes. We now have a web interface showing the existing riddles, enabling an immediate interaction with them, and providing a scalable overview about the room.

5.1.2 Workload Analysis

We analyzed the changed amounts of workload again. We didn't have the possibilty to test our findings, so the listed views are highly subjective and should only be interpeted as the authors impressions, backed by our research.

Cognitive Work

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

1. The user 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, which requires drivers and maybe another libary, but has thorough instructions through a documentation
    (b) doesn't need to understand the communication LowPowerLab libary if he uses the designed template
    (c) doesn't need to look up the other riddles NodeIDs as he can look them up in the WebInterface
2. Understand Arduino-coding
3. Understand working with the "Switch-Case"-scenario used for communication (with instructions)

Process Layer:

1. Understand Javascript to make changes in the Server (in a cleaned up, documented and component-based code with improved readability)
2. We were instructed 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 (*Socket.IO for Unity*; *socket.io-unity*; *socket.io-client.unity3d*) 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.

3. Understand Javascript and HTML to make changes to the React-front-end (also component-based)

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. 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 user has the possibility to save it's 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 user can take the shortest way to activate a functionality

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

Automatisation
We automated tasks like activating the feedback mode of the riddles and sending and starting the applications in the right order

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.

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 it's 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, we didn't have the time to evaluate the user-experience with the new system so we can only estimate the value our framework produces. Also, we weren't able to include a lot of features, listed in the "Big Picture"-section below. A more finished product would have been desirable.

5.1.4 Big Picture

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 riddledetails 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 it's 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 a 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 authentification 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 authentification system. Another aspect was that an escape room doesn't contain privacy sensitive data in our point of view.

Component Isolation

Though we 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 we didn't change the communication protocol, scalability issues might arise depending on the number of incoming messages at the gateway component, especially in combination with the serialport communication. The system 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 easily avoidable bottleneck, if replaced the serial communication with a direct W-LAN connection. A rasperry Pi could be set up as a server with a W-LAN module. There is an regulary updatet (last update is 6 months old in a branch) python wrapper (*Python RFM69 library for RaspberryPi*) with thorough documentation (*RPI RFM69 documentation*) available for connecting a raspberry Pi to the LowPowerLab RFM69 libary. The raspberry could host the websever in a local, protected W-LAN environment and supply the PC hosting Unity with the needed events within the closed network. This way, only authentificated users would have access to the escape rooms properties, the architecture would be separated and therefore clearer for developers, and easier to change. (*RPI RFM69 documentation*)

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

5.2 Conclusion

...

Bibliography

ada, lady. *Radio Range F.A.Q.*

"Prototype" (2015). In: *UXL Encyclopedia of Science*. Ed. by Amy Hackney Blackwell and Elizabeth Manar. Farmington Hills, MI: UXL. URL: http://link.galegroup.com/apps/doc/ENKDZQ347975681/SCIC?u=dclib_main&sid=SCIC&xid=0c8f739d.

Bonomi, F. et al. (2012). "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,*

Dam, Rikke and Teo Siang. *Design Thinking: A Quick Overview*.

Doorley, Scott et al. *The Design Thinking Bootleg*.

floatinghotpot. *socket.io-unity*.

Gartner. *Gartner Predicts 20.4bn Connected 'Things' by 2020*.

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

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

Khan, Rafiullah et al. (2012). "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., pp. 257–260. ISBN: 978-1-4673-4946-8. DOI: 10.1109/FIT.2012.53.

Mashal, Ibrahim et al. (2015). "Choices for interaction with things on Internet and underlying issues". In: *Ad Hoc Networks* 28, 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.

Panettieri, Fabio. *Socket.IO for Unity*.

Postgres.

React. *Create React App*.

— *File Structure*.

React DND.

RPI RFM69 documentation.

smallmiro. *socket.io-client.unity3d*.

Socket.io. *Other Client Implementations*.

StackOverflow. *Developer Survey Results 2018*.

stackshare. *Companies using React*.

Stanford University, Hasso Plattner Institute of Design at. *A Virtual Crash Course in Design Thinking*.

Stojmenovic, I. and S. Wen (2014). *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.

t, vitaly. *pg-promise*.

Trombly, Eric. *Python RFM69 library for RaspberryPi*.

Verizon. *2017 State of the Market: IoT Report*.