

HOCHSCHULE DÜSSELDORF

DOCTORAL 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 22, 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

HOCHSCHULE DÜSSELDORF

Abstract

Media
Media Technology

Bachelor of Engineering

Development of an Integration Platform for IoT Devices

by CaraWATERMANN

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

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

List of Abbreviations

LAH List Abbreviations **Here**
WSF What (it) Stands For

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 Introduction

The Influence of IoT-Devices in everyday life has been rising for decades. Connected devices are expected to number 20 billion (*Gartner Predicts 20.4bn Connected 'Things' by 2020*) by 2020 in nearly every industry. According to (*2017 State of the Market: IoT Report*) the use of media and entertainment on digital devices 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 or will be freed by a supervisor who watches the process to support and step in 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 make the first step to a smart escape room.

The faculty provided an escape room with micro-controllers as riddles. The room provided support for the existing riddles but provided little support for changes. 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 first chapter will provide an overview about our research on topics relevant for this project. The second chapter will analyze the escape rooms architecture concerning our research. The third chapter will explain in further detail how the project was implemented. Chapter four will evaluate the implementation and examine future possibilities for the project.

1.2 Research 5-10 pages

In the following, research concerning the development for our project is listed and explained.

1.3 Architecture (2-3p)

//IoT-system, architectures, layers, models

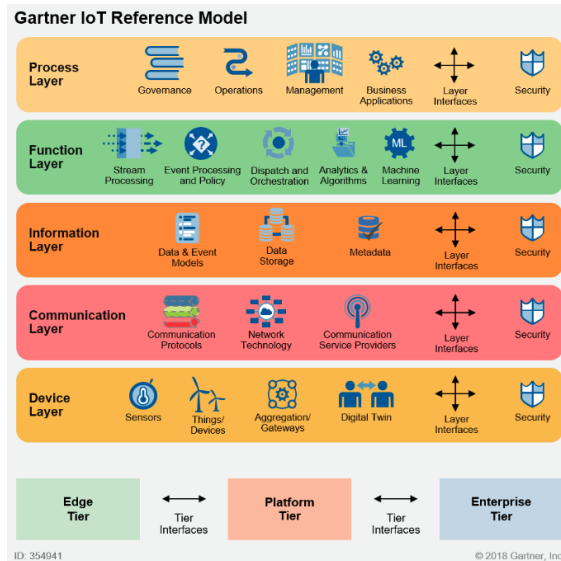


FIGURE 1.1: gartnerIoT1

1.4 Prototyping (1-2p)

//Proof of concept, MvP, Rapid Prototyping

1.5 Front-End

//Should I talk about other options too? (also in the following sections) For the Front-End for our implementation we used Reactjs which is an Open-Source Javascript-Library. After developing Reactjs in 2011, Facebook soon discovered that its' 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, library or Tool" (*Developer Survey Results 2018*). This year, over 100,000 developers participated in the survey. React is designed for building User-Interfaces. Other libraries use the same concepts but Reactjs is currently the most popular one. A lot of libraries are available for react. It's main concepts are:

Components

React motivates its' users to write encapsulated components with single responsibilities. Components combine the HTML-markup and Javascript-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 library like Redux.

Virtual Dom

A 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 negotiator to enable calculating only the parts that need calculating. That's also possible because of React's 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 ({}).

1.6 Communication

Socket.io is a Javascript-Library designed for realtime communication. The server-side is developed specifically for Node.js whereas for the client different implementations (e.g. .Net, Swift, C++)(*Other Client Implementations*) are available. It enables a bi-directional communication channel between a client and a server and offers a fallback mechanism to long polling when WebSockets are not available. Once a connection is established it's maintained and uses a diminishing small amount of 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.

1.7 Middleware

We used Node.js for our back-end. Node.js is an open-source, cross-platform, javascript-runtime-environment. With 49.6% it is this year's "Most Popular Framework, Library or Tool" on this year's Stackoverflow-survey (*Developer Survey Results 2018*). According to Google Trends, interest is rising since 2012(*Google Trend Node*). 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, which saves companies learning costs. Node.js is estimated to have a high learning curve (*node Learning curve*). 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 built on the Google V8 JavaScript engine which compiles Javascript to machine code instead of interpreting it in real time. That makes it faster than other engines. There are thousands of open-source libraries and web frameworks.

1.8 Device

Our prototype used, like the other riddles in the escape room, an Arduino Uno with a RFM69HCW module to communicate with the gateway. Arduino is a micro-controller-company from Italy which was founded in 2005. It's completely Open-Source and supplies everyone with its own IDE ([**arduinoIDEDownload**](#)). The IDE works with nearly all micro-processors on the market. The Arduino (and comparable micro-processors) are programmed in C.

1.9 Back-End

We decided to use PostgreSQL for our database-back-end. 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's easy to set-up and maintain.

Chapter 2

Project Overview5-10p

2.1 Analysis

2.1.1 Layer Analysis

Referring to 1.1, 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 libraries for the RFM69HCW, the LowPowerlab (**LowPowerlab**) and Radiohead (**Radiohead**) library. The escape room used the LowPowerlab library since the architect was familiar with the library. Now-a-days the Radiohead library is the recommended library 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 adresssing number. The other numbers would represent the current state of the Arduino with a "Switch-Case" Szenario 2.2.

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.

- (b) understand the communication within the LowPowerlab-library
 - (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 the answers in a project 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 our current state, the architecture demands a lot 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: you have 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, it wouldn't fit to have hardware with "do not touch" signs all over the place lying around.

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 form a developer were the most critical points where one might spend a lot of time or might decide not to join the project at all because of time planning reasons. 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 ways, 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 environment
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 [2.4](#).

| Work | Before | After |
|-----------|--------|--------|
| Cognitive | high | low |
| Visual | low | medium |
| Memory | high | medium |
| Physical | medium | low |

= low
 = medium
 = high

FIGURE 2.4: Workload of the different aspects color-coded

Chapter 3

Implementation 10

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

3.1 Architecture

We tried to integrate a SoA, so that our components would be self-contained and reusable. For example, the communication to Unity would still work if a connection to the webserver failed. The Tcp and Serial connection were 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 its own namespace for socket.io events so front-end relevant data would only be processed in its set namespace component.

3.2 Prototyping

Our research inspired us to try Rapid Prototyping in our project. We designed a Proof of Concept for each stage 3.1 of our implementation. Our final result is a MVP which provides the bare functionalities but lacks in design and more extensive features which we will explain in the evaluation.

3.3 Device

Since the room consisted of Microcontroller-riddles only at the time of this thesis, we decided to design a prototype and a template for integration of future riddles. 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 protocols for two reasons. First, we wanted keep them out of the way (as far as possible) for new developers. Second, our implementation should be compatible with the older riddles as well, therefore needed to follow the same general rules.

The existing communication protocol followed a "string-to-chararray-send" and "receive-chararray-to-string" structure. The template is divided into 3 parts: "Groundwork", "Riddlefunctionality" and "Remote Functionality". The "Groundwork" section should be filled with libraries, variables and definitions. The "Riddlefunctionality" section should contain the riddles functionalities separated from nearly any communication. The only communication that needs to be defined here is when the Microcontroller should send messages and which. That is executed by writing a

| 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 3.1: List of our PoC steps

single line command containing the desired string. If the string is defined in the "registerRiddle()" function in the "Remote Functionality" section, it will be translated in the web interface. Likewise should the "Remote Functionality" section contain any remote commands for interaction with the web interface and the server. The "Remote Functionality" section consists of two functions:

"registerRiddle()" 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 applicable for any variable.

The other function is named "remoteCommand()" and 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 arrive. The developer is advised to use a "Switch-Case" structure


```

//-----
//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 3.2: "registerRiddle" definition in the Arduino

to define his actions in order to keep the messages understandable. For any action concerning the defined variables, the case should match the index of the variable.

For our prototype, we used an Arduino Uno with a RFM69HCW module, a keypad, and an I2C-display. The riddle would be to guess a code and enter it. We wanted the code to be changeable to provide different difficulties. Moreover would the riddle possess static "won", "lost", and "reset"-values to track and control the riddle's state.

3.4 Back-End

We normalized our database with two tables, one for the riddledetails shown only when the riddle is clicked, and another for listing the general information. We used the 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 separation simplifies database changes, clarifying 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.

3.5 Middleware

3.5.1 Web Server

It quickly became apperant that our back-end web server would use Node.js due to the reasons mentioned above. 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" library (*pg-promise*) for our database integration with postgres. Depending on the event emitted by the web interface,

database-queries to select, update or delete 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 were recognized) or translate the incoming data .

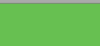





With Socket.io, the client would register whether a front-end or another client would register and forward the needed data from the database. By namespacing (creating different channels for different clients), we tried to avoid dispensable traffic. The gateway would receive the changeable ("write") values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

3.5.2 TCP/Serial/Socket.IO-Client

This part of the back-end changed during our developing process. Since it had already been implemented in C++, we wanted to recreate the functionalities however rewrite them in a more understandable way. Additionally, we wanted to filter the serial messages on their way to the Unity-application via TCP. Since the front-end allowed a reassignment of the messages that would trigger an Unityevent, a backend implementation was needed. We decided to filter the relevant "Finish"-serialmessages through a PostgreSQL-database, which is compatible with loads of languages and frameworks.

Because many developers of our target group were proficient in C# from Unity development, we first tried to implement the functionalities 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 functionalities.

At the end of the project, we decided to revisit our C# code and tried to implement it in Javascript. The same functionalities 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 we 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 3.3: Overview about our front-view challenges.

3.6 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 you define which version of a module you want to use. This prevents unwanted updates so the 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 save 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, we wanted to implement a node-editor to connect riddles in all thinkable ways. When we listed our wished functionalities (Changeable riddleassignments with "Single", "AND" and "OR" connections to the Unity-Events) we decided that a drag-and-drop table would supply those functionalites (Changeable assignments, OR connections) without creating a difficult User-Interface. We used the React-dnd library (*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. We wanted it 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. Additionally, an input field would enable changing the defined value and sending it to the Arduino automatically next time the Server would start. If the Arduino 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 3.4: Overview about our front-view challenges.

Chapter 4

Evaluation and Conclusion 10-15P

4.1 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.

4.1.1 Layer Analysis

Referring to 1.1 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-library 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 connection. An overview from anywhere within the network could be achieved.

Information Layer

A postgresSQL 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 tcpclient(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.

4.1.2 Workload criteria

We analyzed the changed amounts of workload again. We didn't have the possibility to test our findings, so the listed views are highly subjective and should only be interpreted 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 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 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 (**socketioUnity1**; **socketioUnity2**; **socketioUnity3**) That reportedly work (at least) up to the current version of Unity. Since the Unity version in the escape room won't be upgraded regularly, deprecation issues should not arise.
3. understand Javascript and HTML to make changes to the React-front-end (also component-based)

Visual Work

We didn't have the time to research web-design to an extent where we could state confidently that our 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 possibilities to interact with the environment, 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. A user can access the room remotely and doesn't need to start the applications manually.

4.1.3 Limitations

Even though we, to our judgement, achieved our set goals, we should mention 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 assignment is currently not possible, and riddles might be involuntary reset on activating a trigger in Unity.

The code of the project was built 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.

4.1.4 Big Picture

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

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:

- (a) Resetting the room could be implemented by selecting all riddle details with "Finish" as an info value when Unity sends a "resetRiddles" event.

- (b) 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, we didn't include security protection like password authentication and encryption. 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.

We could have used an external raspberry with an integrated gateway module

4.2 Conclusion

...

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

```
\hypersetup{urlcolor=red}, or  
\hypersetup{citecolor=green}, or  
\hypersetup{allcolor=blue}.
```

If you want to completely hide the links, you can use:

```
\hypersetup{allcolors=.}, or even better:  
\hypersetup{hidelinks}.
```

If you want to have obvious links in the PDF but not the printed text, use:

```
\hypersetup{colorlinks=false}.
```


Bibliography

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

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.

Postgres.

React. *Create React App.*

— *File Structure.*

React DND.

Socket.io. *Other Client Implementations.*

StackOverflow. *Developer Survey Results 2018.*

stackshare. *Companies using React.*

t, vitality. *pg-promise.*

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