Design and implementation of a computer aided ergotherapy framework

Georg Grab

Advisor: Dirk Reichardt, Prof. Dr.

STUDENT RESEARCH PROJECT

created as part of the
Bachelor study program

Applied Computer Science

in Stuttgart

June 2018

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Stuttgart, June 4, 2018

Georg Grab

Contents

List of Figures

Acronyms

AR Augmented Reality. 2

FPS Frames per Second. 8

NFR non-functional requirement. 3, 13, 14, 16

ReactiveX Reactive Extensions. v, 7, 8, 19–23

VR Virtual Reality. 2

Abstract

After carrying out hand surgeries, the patient often has to undergo a lengthy recovery period in order to get hand mobility back to the original, healthy state. This recovery phase is usually accompanied by a dedicated ergo therapist in various therapy sessions, requiring the physical presence of both the patient and the ergo therapist.

In a joint venture of the DHBW Stuttgart and the Katharinenhospital Stuttgart, the possibility of computer aided recovery is explored. The long term goal of the collaboration is for the patient to be able to complete some of the recovery exercises at home, saving time and resources for both the patient and the clinic.

This student research project is exploring one particular possibility of achieving this: combining low cost hand tracking devices with the modern web. Hand tracking devices are small hardware devices containing various sensors, capable of producing a virtualized representation of the hand.

The essence of the project is to gamify the recovery exercises: the patient should be able to play games through a web interface, controlled by Hand Gestures (for example, spreading the thumb to make a spaceship shoot). The Hand Gestures correspond roughly to recovery exercises that would normally have been done together with a therapist. The therapist should be able to configure gestures for a patient that he or she has to get better at in order to aid in recovery. These gestures must then be used by the patient in order to correctly navigate the game. The gameplay should finally be producing monitoring information for the therapist to review, and thus provide evidence for the recovery progress of the patient.

This work presents a possible Architecure and Minimal Viable Product (MVP) implementation for such a system. The core system components are identified and implemented. Furthermore, advise on extending the system and the recommended next steps are given. The work concludes with the assessment that the modern web is mature and performant enough to sucessfully allow the implementation of the project. Every subcomponent, from data preprocessing to playing games can be implemented using web technologies with adequate performance and robustness, without the need for any external dependencies. The Appendix of this work contains both an end-user and developer manual.

Chapter 1

Introduction

1.1 Problem description

In 2016, approximately 60.000 german residents have been hospitalized due to hand or wrist injuries [18]. The inpatient treatment of such injuries is often followed by a lengthy recovery phase in which ergotherapeutic treatment occurs in order to further aid recovery. As ergo therapy sessions are usually held in one-on-one sessions, this results in a significant time and resource both by the patient and the treating clinic. Additionally, the sessions themselves are often described by patients as boring and unmotivating, citing their repetitive nature.

Ergo therapists of the Katharinenhospital Stuttgart are currently researching alternative treatment methodologies that could constitute an improvement to all three of these fundamental problems. The aim of the research is to introduce various gamification aspects to the recovery sessions. The patients should be enabled to perform repetitive parts of the recovery exercises at home by means of successfully executing them while controlling video games. This methodology of executing prevention and rehabilitation measures is an active and well known field of research commonly referred to as Exergames [10]. In a more general sense, games that are designed with a second primary purpose (apart from entertainment) are referred to as serious games [14].

1.2 Project Scope

The scope of this student research project is to design and provide implementations for a software solution acting as an underlying framework on which the Exergames can be executed. The framework should be capable of meeting the determined requirements as outlined in section 1.3 either directly through the provided implementation, or, if some requirements cannot be fulfilled with the reference implementation, by way of easy extensibility. The framework should also contain a user facing component where relevant measurements and game configurations can be made and from which the games are executed. Additionally, the framework should contain various tools that will make it easier for future developers to develop and debug subcomponents.

1.3 Requirements Analysis

Outsourcing recovery exercises into a space where no direct therapeutic supervision is available generates a series of challenges that have to be identified and overcome before successfully integrating Exergames in the recovery sessions.

1.3.1 Functional Requirements

In a software development context, the challenges a system has to solve in order to become useful to the stakeholders are referred to as functional requirements [2]. The most notable functional requirements are outlined as follows.

1.3.1.1 Domain Virtualization

In order for Exergames to fundamentally function, they require an accurate, real-time virtualized representation of the problem domain. For example, in order to develop Exergames for treating hand injuries, a virtual representation of the hand must be available. For the domain of upper extremities, several hardware devices exist that are capable of providing the virtualized representations. Notable examples are the Leap Motion Device by Leap Motion, Inc.[1], and Microsofts Handpose technology for the Kinect Device[2]. Both devices, normally used in the context of Virtual Reality (VR) and Augmented Reality (AR), are capable of producing the virtual representation by employing a variety of hardware sensors.

1.3.1.2 Exercise Classification

The most important capability of an Exergame is to correctly classify whether a recovery exercise has been executed. In the domain of hand and wrist injury recovery, a recovery exercise may for example be the spreading of the thumb, where the remaining fingers of the hand remain closed. Other examples for recovery exercises have been outlined by [3]. The result of the exercise classification can then be used as a gameplay element in the Exergame. For example, if a thumb spread exercise as described above has been executed well enough, a *Space Invaders-like* Exergame could trigger the space ship to shoot.

1.3.1.3 Patient Adaptibility

One-on-one therapy sessions in ergo therapy are required because of the large variety of different hand injuries, each requiring a different set of recovery exercises. Additionally, the classification logic (see 1.3.1.2) for the recovery exercises themselves have to be adaptable to how far the patient has progressed so far in recovery. For example, if the patient is progressing well in recovery, the relevant exercise has to be increased in difficulty in order for the treatment to remain effective.

[1] https://www.leapmotion.com

[2] http://research.microsoft.com/en-us/projects/handpose/

1.3.1.4 Monitorability

The ergo therapist has to be able to view monitoring information related to the patients playing activity. Most fundamentally, the therapist should be able to view the number of times and total duration of Exergames played in order to verify if the agreed upon exercise volume has been completed. Additionally, specific information that aid the ergo therapist in assessing the recovery progress of the patient should be available. If the therapist determines that the current exercise has to be adapted in some way, or for exchanging other kinds of information with the patient, such as providing hints or agreeing on the next physical appointment date, this should be possible through an integrated messaging platform. Furthermore, the monitoring information should be able accessible through a web-based interface.

1.3.1.5 Gameplay and Frontend

Finally, the system should provide a frontend component, from which the actual games are executed and configured, and where display components relevant for resolving other software requirements can be found.

1.3.2 Non-Functional Requirements

In addition to the functional requirements, the following non-functional requirements (NFRs) have to be considered while designing and implementing the system. NFRs are global requirements that are not directly related to function, but refer to the development or operational costs of the system, such as performance, reliability, and maintainability [5].

1.3.2.1 Modularization

On a technical level, the program logic responsible for classifying if an exercise has been completed (see 1.3.1.2) should be separated from the actual Exergames logic (see 1.3.1.5). This would pose the advantage of introducing a modular aspect to the system, as both exercise classifiers and games could be exchanged, both keeping the patients engaged in the platform by allowing them to train their assigned exercise using a variety of games, and greatly simplifying the work of future developers, as they will be enabled to develop games for the platform without any prior knowledge of exercise classification, and vice versa.

1.3.2.2 Performance

Performance is a critical NFR for the system. All data coming from the device providing the relevant domain virtualization has to be ingested, preprocessed, and classified in real time. If this is not the case, the patient will experience a significant lag between the performed exercise and the feedback of the Exergames, quickly resulting in frustration. Additionally, the execution of the Exergames themselves should be performant enough so that the gameplay experience isn't negatively obstructed.

1.3.2.3 Availability

From the therapists point of view, it is critical that the platform is capable of running without an active network connection. This results in the technical restriction that all network connections made by the platform must be both optional and fault tolerant. This requirement originates from the assumption of the therapists that the system will not always be used in contexts where an internet connection is readily available.

1.3.2.4 Ease of deployment

The system should ultimately be primarily executed on a patient provided device. As such, deployment of the application should be easy, and robust with respect to a multitude of possible, previously unknown target environments.

1.3.2.5 Extensibility

As the system is acting primarily as an underlying framework on which other developers should build upon in the future, it should be written in a way that allows for easy extensibility. It should especially be written in a computer programming language that is well known to the potential target developer audience, so minimum prior knowledge is required before starting development with the project. Additionally, the framework should be future proof: it should be simple to exchange subcomponents with more modern equivalents in the future. For example, it should be simple to add support for more modern hardware devices providing domain virtualization, or more modern graphics libraries for developing the Exergames in the future.

1.4 Outline and structure

Section 1 gave a general overview of the project motivation, scope, and method. Additionally, the project requirements gathered in various in person meetings with the stakeholding therapists were outlined.

Section 2 presents prior relevant work on the subject of computer aided ergotherapy and puts this work into context. The following section will introduce the relevant technologies and theoretical foundations used in the implementation phase of the project.

The main part of this paper is constituted by section 4, where possible project architectures are presented, the project reference implementation is detailed and architecture and technology choices are justified. The work concludes with the recommended next steps in developing the system (section 5), and a critical conclusion (section 6).

Chapter 2

Related Work

The general feasibility of using devices such as the Leap Motion for different usecases than their primary intended purpose in the entertainment industry, specifically using them for clinical purposes, has already been shown by various research.

In 2014, researchers from the University of California have successfully developed a Exergame version of the popular smartphone game *Fruit Ninja*, utilizing the Leap Motion Device for domain virtualization [13]. The researches have shown that the Game could purposefully be used for stroke rehabilitation by showing that a strong correlation exists between the achieved score while playing the *Fruit Ninja* game, and standard clinical assessment scores.

The feasibility of using the Leap Motion device for developing Exergames specifically in the context of the Web browser have been shown by [7]. In the work, the authors showed the general feasibility of web based digital hand rehabilitation by implementing and evaluating Web based Exergames on a prototypical level. In an earlier work, the same researchers have shown that the Leap Motion device is capable of delivering a virtualized hand representation of sufficient accuracy for tracking the patients rehabilitation progress [6].

In a research project completed in 2017, students of the DHBW Stuttgart initially collaborated with ergo therapists of the Katharinenhostpital Stuttgart in order to determine requirements for a web based handtherapy system that would be mature enough to be deployed at hospitals at a bigger scale [3]. In the project, some preliminary system requirements have been identified, and recommendations for the architecture and implementation of the system have been given. In addition, some of the recovery exercises that the system should be able to assimilate have been documented.

This work follows up on the previous research by providing an architecture and reference implementation for a mature, holistic hand therapy system that is ready to be used by ergo therapists and their patients as an accompaniment for hand and wrist injury rehabilitation.

Chapter 3

Technologies and Methods

This chapter gives a quick introduction to the non-trivial technologies and methods used while developing the system. For purposes of brevity, technologies that are well known or only marginally relevant to the following chapters have been ommited[1]. The utilized technologies can roughly be categorized into runtime specific technologies, which are relevant during system execution, and development technologies, which are only relevant while developing, compiling, and testing the system.

3.1 Runtime specific Technologies

3.1.1 Vue.JS ecosystem

3.1.1.1 Vue Framework

Vue is a Javascript framework used for simplifying the development of reusable, responsive frontend components. The core framework was initially developed by Evan You in 2014. Since then, the framework quickly gained popularity in the open source community, currently being the 5th most popular open source project on the code sharing platform GitHub [29]. The framework serves a similar purpose as other popular Javascript frontend libraries such as React and Angular, though design, scope, and implementation differences can be found in various places [33].

3.1.1.2 VueX: State management and Flux Architecture

The Vue core library is intentionally restricted in scope to the applications view layer. However, a vast Ecosystem has recently emerged around the Vue library covering other required functionalities of modern Web Applications, such as state management, client-side routing, or network connectivity. VueX is a library providing a state management system to the Vue core library which is modelled after Facebooks Flux Architecture [25][28]. It serves as a centralized store where all application state is located. State can only be retrieved and modified in a predictable fashion. This results in easier maintainability and extensibility for large applications, as developers always know how state is

[1] Such as CSS3, HTML5, JS ES6, CSS Preprocessors, the vue-router extension, and the Git distributed version control system.

located, accessed, and modified.

3.1.1.3 Material Design Components

Material Design is an open source user interface design system developed and maintained by Google. It consists of design guidelines, components, and development tools. At the time of writing, it is being used in a large quantity of Google products. Furthermore, it is the officially recommended Design Language to use when developing Applications for the Android Platform [30]. The Design Components are thus familiar and intuitive to use for a large amount of potential end users. The Design Components are available for Vue.JS applications through the vue-material library [32].

3.1.2 Reactive Extensions and rx.js

ReactiveX is a language agnostic, open source API for asynchronous programming with observable streams. It can be seen as an extension of the Observer pattern [24], one of the twenty-three well-known "Gang of Four" design patterns [9].

The traditional Observer pattern is based on the fundamental concepts of the *Observable* and the *Observer*. The *Observable* has the ability to *emit* items over time. An *Observer* may declare a dependency on the *Observable* by means of *subscribing* to it. If a subscription has been established, the *Observer* will receive the items as they are emitted from the *Observable*.

The ReactiveX specification extends on this concept by providing a variety of *Observable operators* to change the emission behavior, transform the emissions, or combine multiple *Observables*.

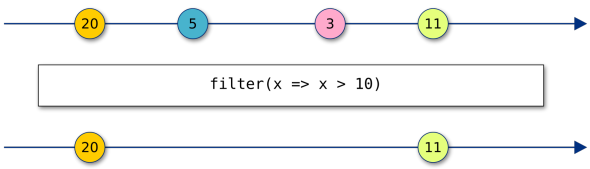


Figure 3.1: An example Observable transformation, represented using *Rx Marble Diagrams*.

Figure 3.1 gives a visual presentation of an Observable transformation. The top and bottom arrows both represent Observables. The arrow-direction represents time, the colored circles on the arrows represent items emitted by the Observable. The box in the center represents the application of an Observable operator, in this case the *filter* operator, which filters Observable emissions by applying the given predicate, in this example $x > 10$. The result of the transformation is a new Observable that only includes the items that match the given predicate.

Apart from the Observable operators, ReactiveX also includes specialized versions of the traditional Observable, such as the *Subject*, which has the property of being able to act as Observable and Observer simultaneously, or the *Single*, which represents a singular value emission in the future (similar to the Javascript ES5 Promise).

As the ReactiveX project only outlines the specification and API of the programming

concept, concrete implementations have to be provided for the target programming environment. For Javascript, the official implementation of ReactiveX is rx.js.

3.1.3 Leap Motion Device

3.1.3.1 Virtualized Domain Model

The Leap Motion Device is capable of producing a virtualized model of one or more hands placed inside the devices field of view. The Device Data consists of a stream of JSON-formatted device frames. Each device frame contains the current position of all detected hands in that point of time. The data is transmitted in a certain amount of Frames per Second (FPS) (depending on device version, protocol version, and device driver settings), usually in the vicinity of $30 - 100$ FPS.

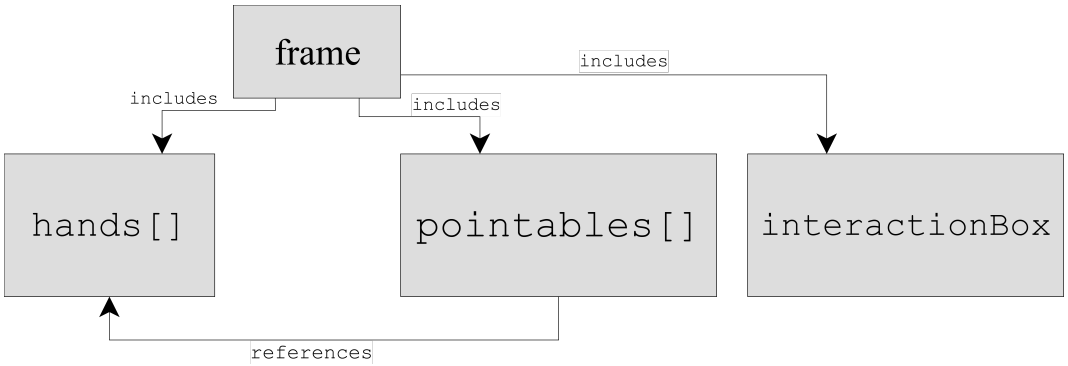


Figure 3.2: High Level view of the Leap Motion Device Frame Data

Figure 3.2 gives an overview of the relevant device frame data. The frame data consists of multiple subcomponents. The subcomponents relevant for this paper are the *hands* and *pointable* arrays, as well as the *interactionBox* object. The hands array contains representations of all hands that have been detected in the frame. The hand representations contain positional information, such as estimated position, width, and velocity, as well as meta-information, such as a frame-unique ID.

The pointables array essentially contains all detected fingers, though recent Leap Motion devices are capable of detecting other types of pointable objects, such as pencils as well. Like for tracked hands, positional information such as estimated position, width, velocity, and finger type (thumb, index..) is encoded in the representation. Additionally, recent Leap Motion Devices are capable of tracking skeletal information, such as the position of the hands metacarpals, proximal phalanges, intermediate phalanges, and distal phalanges [26]. Finally, the pointable is associated to a hand object of the same frame by referencing its ID.

The interactionBox object denotes the position of a virtual three-dimensional rectangle positioned above the device sensor. The box describes the area in which the device is able to detect hand objects. All positional data coming from the device is described in relation to the interactionBox.

3.1.3.2 Hardware Device Driver

In order to interface with the Leap Motion Controller, its Hardware Device Driver has to be installed on the target device. The Hardware Device Driver is constituted by an operating system service named *leapd* which interprets data coming from the Leap Motion controller and makes it available through a locally running Web Socket Server (see section 3.1.5.4).

3.1.4 Graphics

3.1.4.1 THREE.js

Recent Web Specifications such as WebGL make it possible to run graphically and computationally expensive programs directly in the browser, by allowing Web Applications to directly access the devices Graphical Processing Unit [12, sec. 1]. Several Javascript libraries have since been created to simplify working with the WebGL API. A popular library is THREE.js, which is specializing simplifying the creation of 3D Applications.

3.1.4.2 p5.js: Game Development

p5.js is a simplistic Graphics Library similar in functionality to THREE.js. However, it primarily focused on 2D drawing, and has been simplified greatly in other aspects as well, allowing for rapid graphical prototyping and game development [31].

3.1.4.3 d3.js: Dynamic Documents and Visualizations

The final relevant graphics library is d3.js, which greatly differs in purpose to the two previously introduced libraries. Its primary purpose is simplifying the creation of interactive, data-driven documents and data visualizations, such as diagrams and plots [20, preface].

3.1.5 Web Specifications

3.1.5.1 IndexedDB: Persistent Storage

IndexedDB is a Browser API allowing the Browser Application to persist arbitrary data. It differs from other client side persistence APIs such as Cookies and Localstorage in various aspects, most notably the amount of data it can hold. Unlike Cookies and Localstorage, which both are constrained to a relatively small maximum size, the maximum size of an IndexedDB Database is usually much larger, sometimes restricted solely to the free space on an end devices hard drive[2] [21]. Additionally, IndexedDB allows for highly efficient storage access by providing the possibility to locate entries using incides. Though the exact implementation of the IndexedDB specification is dependent on the Browser vendor, it is usually implemented using a persistent B-tree data structure [1, sec. 1].

[2]The exact size limitation depends on the implementation of the Web browser, as no standardization exists thus far.

3.1.5.2 Web Workers: Javascript threading

The Web Workers specification defines an API for executing Javascript code segregated from the main thread, often also called the window context. The Web Workers can thus be executed on a different processing core, resulting in no noticable performance impact while navigating the Web Application, even if computationally expensive tasks are performed [11, sec. 1.2.1]. A Web Worker is able to communicate with the window context through a bidirectional, event based API [11, sec. 4.6.1].

3.1.5.3 Service Workers: Offline capability

Service Workers are a special form of Web Workers that additionally allow the Web Application to work in an offline mode by means of intercepting and caching its asynchronous HTTP requests [17, sec 4.5, 5].

3.1.5.4 Web Sockets: bidirectional message streaming

The WebSocket protocol is a protocol allowing for low overhead, full-duplex communication with a WebSocket compliant Server over a single TCP connection [8, sec. 1.1]. The protocol is able to be used from a Web Application context.

3.2 Development specific Technologies

3.2.1 Webpack: module bundling

Webpack is described as a module bundler for Javascript, essentially allowing to express dependencies between modules [23]. In the terminology used by Webpack, modules are defined as anything that is required for the compilation of the Web Application, such as Javascript source code, but also images, style information, and HTML source code. Webpack transforms modules with dependencies into multiple static assets that are ready to be executed by the Web Browser. The transformation step may include various preprocessing and postprocessing steps, allowing the optimization of the Web Applications build pipeline regarding development comfort, runtime performance, and final asset size.

3.2.2 Typescript: static typing

Typescript is a typed superset of Javascript. Its primary purpose is to augment the language with a static type system [4]. As Javascript is an interpreted and dynamically typed language, Typescript is capable of introducing some compile time type safety to the language. Typescript is not able to be executed directly in the Web Browser, but has to be compiled back to Javascript prior to execution. This can for example be achieved by preprocessing the Typescript source using a module bundler such as Webpack (see section 3.2.1). During the compilation phase, the Typescript Compiler is able to perform static type checks and checks for nullity, usually resulting in less runtime errors. Additionally, as Typescript introduces constructs well known from statically typed programming languages, scaled development is claimed to be become simpler, as multiple developers are able to define interfaces and contracts between submodules.

3.2.3 Inversify: Inversion of Control

Inversify is an Inversion of Control framework, providing support for dependency injection to the Javascript language.

3.2.4 jest and sinon.js: Unit Testing and Mocking

Jest is an open source unit testing framework for Web Applications, primarily developed by Facebook Inc. Unit tests require all external dependencies of the tested module to be mocked, so that only the relevant code is tested. sinon.js is a library providing this mocking functionality to Javascript tests.

Chapter 4

Framework Implementation

4.1 Architecture Decision

4.1.1 Available Alternatives

Based on the requirements outlined in section 1.3, which have been gathered and derived in various in-person meetings with the stakeholding therapists, multiple technologies for implementing the system seem feasible. The alternatives considered at the beginning of this project are outlined as follows.

4.1.1.1 Fully featured Web Application

One technological possibility would be to implement the core system framework as a fully featured Web Application. All requirements would be implemented using web technologies. Most notably, data ingestion, preprocessing, classification, and monitoring would have to be accomplished entirely in the context of a web browser. The system would be self-sufficient in this configuration, without reliance on any external systems, though an external server component for sending the monitoring information is conceivable (see requirement 1.3.1.4).

This configuration is very favorable in terms of deployment and extensibility, as every component is consistently written in Javascript, the standardized programming language of the web. Likewise, as functionality is not distributed over multiple systems, the deployment of the systems only requires a web browser[1].

However, for this configuration to be feasible, it is required that the domain virtualization device contains an API to the Web Browser. Also, the Javascript programming language is sometimes critizied for its relatively poor performance when compared to programming languages with compilers capable of producing native code. This is owed in large parts to the dynamically typed and interpreted nature of the language. Some benchmarks show that Javascript is up to ten times slower in terms of runtime performance compared to C++ when running computationally expensive tasks [27]. The question of whether Javascript is performant enough to tackle the task at hand has to

[1]The installation of relevant hardware device drivers on the target device is also required for the system to function, but as the same is true for all other alternatives, this is not considered under the ease of deployment aspect.

be clarified before this alternative can be considered feasible.

4.1.1.2 Web Application with local server component

The two main disadvantages of the fully featured Web Application, the need for a Web API of the virtualization device and the performance considerations, could be mitigated by moving the computationally expensive logic in a locally running server component, and interfacing the two components by using an asynchronous communication specifications such as WebSockets or XmlHttpRequest. As the code running in the server component is running with Operating System permissions, it could directly interface with the hardware device. Additionally, all logic concerned with working with the virtualization device data could be implemented in native code, resulting in high performance.

This approach in turn poses the disadvantage that a lot of additional complexity is introduced into the system. The system would no longer be implemented in a single programming language. Additionally, the system would no longer be easy to deploy, as compiled binary packages would have to be provided and thoroughly tested for each desired target Operating System.

4.1.1.3 Web Application with remote server component (backend)

Following up on the design outlined in section 4.1.1.2, the system could also be designed with a remote server component instead of a locally running server. This would have the advantage of all device data being available at one centralized location, where very elaborate analytics could be performed. As all alternatives outlined in this section will eventually require a backend component for accumulating monitoring information for the therapist to view and inspect, this approach initially seems to reduce complexity.

However, the virtualized domain data would again have to be ingested by the Web Application, in a similar fashion to the alternative outlined in 4.1.1.1, as no local application is available to handle the connection to the hardware device. Furthermore, all virtualization device data would have to be sent over the network connection of the end device, potentially resulting in high latency, and possibly full system outage if the end device fails to establish an internet connection.

4.1.1.4 Desktop Application

Finally, the system could be designed without relying on Web technologies altogether, and be instead implemented as a traditional Desktop Application.

4.1.2 Elected Alternative

Table 4.1 gives an overview over the likely requirement fulfillment of the discussed implementation alternatives.
The Desktop Application is the only proposed architecture incapable of intrinsically meeting the functional requirements: as the therapists plan on viewing and evaluating the platform monitoring data from a Web based interface, a separate application would need to be developed for that sole purpose. Regarding the NFRs, Performance and Modularization can be fulfilled easily, as statically typed, modern programming languages

Table 4.1: Comparison of implementation alternatives based on estimated requirement fulfillment

	Functional Reqs.	Modularization	Performance	Availability	Deployment	Extensibility
Desktop App (4.1.1.4)		X	X	X		
Web thin client (4.1.1.2)	X	X			X	X
Web local client (4.1.1.3)	X	X	X	X		
Web fat client (4.1.1.1)	X	X	?	X	X	X

that compile to platform-native code can be employed. In addition, the system could be designed in such a way that meets the Availability requirement. However, as the architecture cannot fulfill all Functional Requirements, and additionally lacks the NFRs of Deployment (platform dependant binary must be provided) and Extensibility (future developers will likely need to learn a new programming language), this alternative is ruled out as a potential platform architecture.

While the Web based thin client outlined in section 4.1.1.3 is capable of implementing all functional requirements, the architecture is ruled out as it is relying on a remote server for basic functionality, resulting in a failure to meet the Availability NFR. Additionally, even if a network connection is available, there are serious considerations to be made regarding network latency. The latency aspect imposes a performance dependency on the quality of the users internet connection, resulting in at least unreliable performance.

The Web based client backed up by a local webserver as outlined in section 4.1.1.2 solves the problems of the Web based thin client regarding Availability and Performance, however, the ease of deployment aspect would be lost by the fact that a native platform binary is required for executing the application. Also extensibility is jeopardized, as a large amount of complexity is introduced into the system, most notably the fact that two separate programming languages or frameworks would be required for developing the system. While these tradeoffs would certainly not be critical, this choice of architecture is considered unfavorable for the time being.

The fully featured Web Application has the theoretical capability to meet all functional requirements. Additionally, most non-fuctional requirements can be met by the architecture: the modern Javascript ecosystem allows for modularization and code splitting capabilities through the use of ES6 Modules. Furthermore, supersets of the language exist that provide support for static typing, such as Typescript. A static type system has the advantage that programming interfaces may be explicitly typed, so the program becomes easier to extend by developers not fully familiar with the program as a whole [4]. It can be assumed that development on the Web platform is well known to the target developer audience, as it is a well established topic that is teached in most computer science related university courses. In addition, the architecture is easy deploy on the

target end devices: in essence, all that is required from the end users is to navigate to a Website using a relatively recent Internet Browser. While this action initially seems to break the Availability requirement, modern Web Specifications, most notably the Service Worker Specification, allow for the application to still be available if the network connection is lost [17]. The only non-functional requirement of the application that is in need for clarification is whether the architecture supports adequate performance for resolving the task at hand. However, several Web technologies are currently developed to mitigate this exact problem. Most notably, the widely adoped Web Workers specification essentially allows for developing multi threaded applications on the Web [11]. Additionally, the WebAssembly specification has recently reached a mature state and is available in all major browsers. WebAssembly is a binary instruction format designed to be deployed on the web, allowing for web developers to develop code executing at near native speed while maintaining cross platform compatibility [16].

Based on the considerations employed in this section, the fully featured Web Application is chosen as the system architecture, as it seems to support implementation of all imposed requirements.

4.2 System Architecture

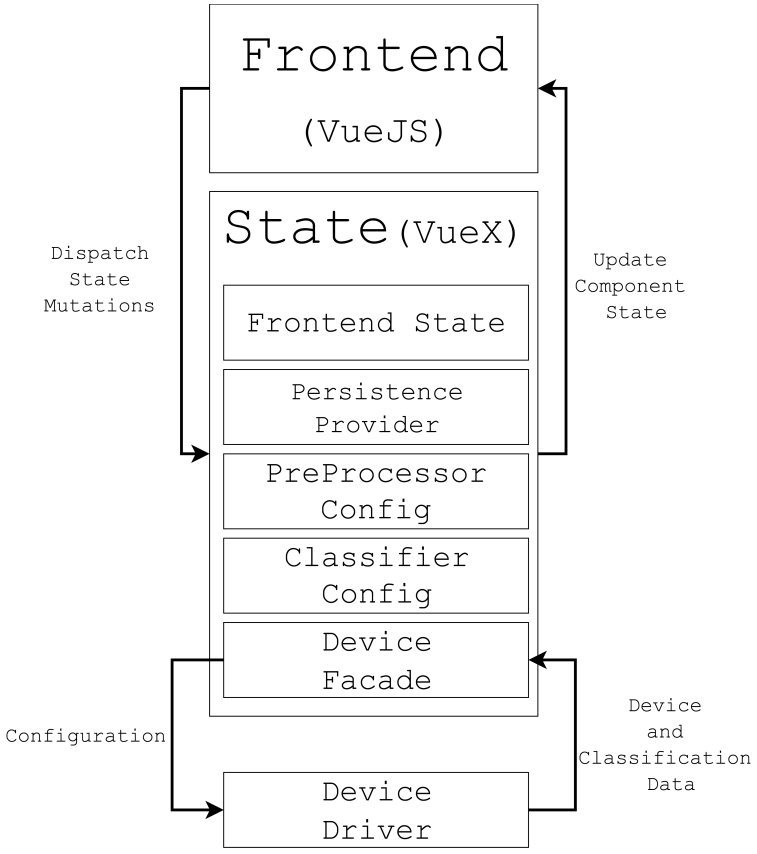


Figure 4.1: High Level Architecture Overview

A high level overview of the resulting architecture is outlined in Figure 4.1. High level architecture components are prepresented as rectangles, relations between them are displayed as arrows. An *includes* relationship is illustrated by nested rectangles. The architecture closely follows the state management pattern recommended by Vuex.

The Frontend, illustrated by a single rectangle in this case, communicates with the Application State by signifying desired state changes (for example, the ticking of a checkbox) by commiting mutations. At the same time, the Frontend components declare a dependency on relevant parts of the state. If these parts of the state update, Vuex automatically rerenders the dependant Frontend components in order to reflect the changes.

The Application State itself contains, apart from the state of the Frontend Components, references to the other subcomponents of the application, namely references to a component facilitating persistence, configuration information for data preprocessors and classifiers, and finally a reference to the device facade component, which facilitates access to the device driver implementation. These components are further elaborated in the following sections.

4.2.1 Frontend

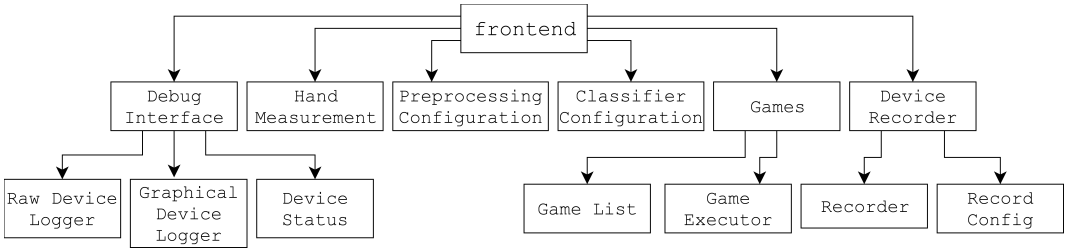


Figure 4.2: Frontend: Detailed Subcomponent Overview

As Figure 4.2 illustrates, the frontend is divided in a number of subcomponents, each responsible for fulfilling a specific purpose.

4.2.1.1 Debug Interface

The Debug Interface is responsible for diagnosing problems if the connection to the motion tracking device cannot be established, and for providing development tooling. It is thus mainly responsible for contributing towards the Extensibility NFR (section 1.3.2.5). The Interface consists of three subcomponents.

Raw Device Logger The Raw Device Logger is a Frontend Component capable of displaying the raw device frames as they arrive. The component receives the incoming data stream, and formats it in order to be human readable. As new data arrives, the Component updates itself automatically, making it possible to inspect the incoming device data over time. The Component acts as a developer tool, allowing the developer to view live device data, and thus gain intuition over how the device data is structure and how it is changing over time.

Graphical Device Logger Similarly, the Graphical Device Logger is acting on the incoming device data stream, but instead of formatting the raw data in a textual manner, it is interpreted graphically. Using the Three.JS library (3.1.4.1), graphical hand and finger objects are created and displayed inside a 3-dimensional coordinate system.

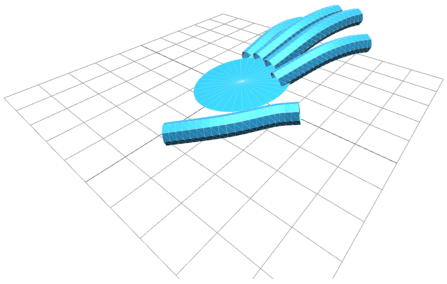


Figure 4.3: The Graphical Hand Logger.

Figure 4.3 illustrates the graphical representation the component is able to produce for Leap Motion Device Data. The Component is able to display other device data as well, though only illustrations based on Leap Motion Device Data are implemented as part of this work. The Leap-Motion specific illustration is constructed by first projecting the interactionBox-relative positional device data (see section 3.1.3.1) in the THREE.js coordinate system by applying the formula

$$x_{proj} = proj_{min} + (proj_{max} - proj_{min}) * \frac{x_{leap} - iBox_{min}}{iBox_{max} - iBox_{min}} \tag{4.1}$$

to each dimension of relevant positional data, where x_{proj} is the target position, $proj$ describes the bounds of the target projection, x_{leap} is the positional data of the Leap Motion Device, and $iBox$ describes the bounds of the interactionBox.

Afterwards, the palm is drawn by constructing a 2-dimensional disc, and sizing, positioning, and rotating it according to the transformed hand data. The fingers are drawn by constructing a 3-dimensional tube from each pointable object of the frame data. The tube is constructed from the skeletal tracking information of the pointable and interpolated using a Centripetal Catmull-Rom spline. This type of interpolating curve is widely used in the field of computer graphics because of several useful mathematical properties [19].

Device Status Log The Device Status Log is the final debugging component of the framework. The Status Log portrays the state of the three main failure points of the application. The main failure points include the connection of the Javascript Device Driver to the Native Device Driver (see 3.1.3.2), the availability of the Native Device Driver, and the Physical availability of the hardware tracking device. The Framework is able to detect the state of all three of these failure conditions, and displays instructions to the user on how to mitigate these problems in case one or more failures occur.

4.2.1.2 Hand Measurement

The ergo therapist is required to measure angles between certain bones of the hand in order to determine how the patient is progressing in recovery [3, sec. 4.1.6]. The digitized ergo therapy system should be able to perform these measurements as well in order to reduce the manual workload of the therapist.

The reference implementation of the system includes a frontend component with the ability to take measurements of the angles between neighboring fingers. Other types of measurements are able to be integrated in the framework easily by using existing code structure and utility functions. The measurements are able to be averaged across a configurable timeframe in order to achieve more accurate results. The angle calculation itself is performed by applying the formula

$$angle = \arccos(\frac{\vec{p_1} * \vec{p_2}}{\|\vec{p_1}\| * \|\vec{p_2}\|}) * \frac{180}{\pi} \tag{4.2}$$

to the directional vectors of neighboring fingers.

4.2.1.3 Preprocessing and Classification Configuration

The stream pre-processing and classification frameworks, outlined in further detail in sections 4.2.5 and 4.2.6 respectively, require user configuration from the frontend. The available subcomponents that are available for each framework can be toggled on or off, and supplied with configuration that the subcomponent requires. Every change made in the Frontend configuration will automatically be propagated through the system, and will be available immediately.

4.2.1.4 Game Listing

The most integral part of the system is for the patient to be able to play games by means of performing the relevant recovery exercises as dictated by the therapist (see section 1.3.1). In order to implement this functionality in the system, the frontend has a component where the available Games are listed. For each available Game, a short title and description is shown. Furthermore, this component validates if the system is ready to execute the game (i.e., the motion tracking device is plugged in and functioning normally, and a classifier has been activated, as outlined in 4.2.6). After the component has validated that the system is ready, a Play Button becomes available to the user. After the Play Button has been pressed, the component hands control to the Game Execution Framework (see section 4.2.7).

4.2.1.5 Hand Recorder

The Hand Recorder is a Frontend Component with the primary purpose to alleviate framework developers from having to physically connect a hand tracking device in order to develop preprocessors, classifiers, or games for the platform. The component allows to record series of device frames. Optionally, the recordings may be titled, and persisted in the Persistence Provider implementation (see section 4.2.4). If a previously saved recording is activated using this component, the contained frames will be propagated in an endless loop through the framework as if they were coming from a physical device.

```
export interface DeviceFacade {
    getHandTrackingData: (
      store: Store<RootState>
    ) => Observable<GenericHandTrackingData> | undefined;
    getClassificationStream: () => Observable<ClassificationData> | undefined;
    getDeviceDriver: () => DeviceDriver;
    updatePreProcessors: (x: PreProcessorConfig[]) => void;
    updateClassifier: (x: ClassifierConfig) => void;
    ...
}
```

Figure 4.4: Device Facade Interface Definition (excerpt)

Only one saved recording may be active at any given time. If a hand tracking device is already connected and functioning as expected, the data coming from the recording will be preferred. Furthermore, the user is able to download saved recordings, for example in order to reproduce bugs, or using them for illustration purposes in other parts of the system.

4.2.2 Device Facade Interface

The Device Facade Interface is the primary access point for frontend components that wish to work with motion tracking device data. The Interface, named after the well-known Facade pattern in Software Development [9], decouples the Frontend Components both from the actual implementation of the Device Driver (see section 4.2.3), as well as the actual data source (i.e., Recording, Physical, or potential future data sources).

Figure 4.4 shows an excerpt of the Interface. References to ReactiveX streams representing Hand Tracking Data, Classification Data (see 4.2.6), and the DeviceDriver implementation itself may be optained. Additionally, the Facade may receive Configuration updates regarding the Data Preprocessing and Classification Frameworks from Frontend Components, which will subsequently be relayed into the Device Driver for further processing. The Interface is bound to the desired Implementation at compile time through dependency injection, and registered to the global application state on page load.

The System contains a reference implementation of the Device Facade Interface, named *AllPurposeFacade*, though different implementations, for example a mock implementation for utilization in Unit Tests, are conceivable.

4.2.3 Device Driver Interface

Components implementing the Device Driver Interface are primarily responsible for handling the connection to the physical device from the Web Context. In case of the Leap Motion Platform, such a connection is obtained by connecting to a local Websocket Server provided by the Leap Motion Hardware Device Driver *leapd* (see section 3.1.3.2). Other potential Motion Tracking Devices, such as the Microsoft Kinect Plat-

```
export interface DeviceDriver {
  deviceName: string;
  establishConnection: () => Observable<DeviceConnectionState>;
  getTrackingData: () => Observable<GenericHandTrackingData>;
  getClassificationData: () => Observable<ClassificationData> | undefined;
  enableClassification: (classifiers: string[]) => void;
  updatePreProcessors: (configs: PreProcessorConfig[]) => boolean;
  updateClassifier: (config: ClassifierConfig) => boolean;
  digest: (data: GenericHandTrackingData) => void;
  ...
}
```

Figure 4.5: Device Driver Interface Definition (excerpt)

form, provide similar methods of accessing the Hardware Device from a Web Context [15]. New Device support can thus be added to the platform by means of adding a Device Driver Implementation, and optionally providing the parts of the Platform that depend on specific Device Data with the relevant device-specific implementations, for example providing rendering methods for the new Device to the Graphical Device Logger (see 4.2.1, par. *Graphical Device Logger*).

Figure 4.5 illustrates the methods and properties that must be implemented by Device Drivers. The `deviceName` property must be defined on the implementing class. This property acts as a unique device identifier across the whole system. If device specific implementations are required for some parts of the system, they are resolved using this identifier. The method `establishConnection` is used to signal to the DeviceDriver that a connection should be established. This method is called on system start. The method must return a ReactiveX Observable representing the connection state to the device driver over time. After `establishConnection` has been called, the DeviceDriver should automatically attempt to reconnect to the native device driver if the connection has been lost or could not be established initially, and update the Observable on each registered change of connectivity.

The `getTrackingData` method must return a ReactiveX Observable that should emit the tracking data each time a new frame is received by the driver. The generic type of a device frame is named `GenericHandTrackingData`. The concrete implementations of the Interface may extend this type in order to explicitly type the data coming from the device. The methods `enableClassification`, `updatePreProcessors`, and `updateClassifier` are used to signal to the device driver that its internal Data Preprocessing and Classification pipelines should update. These subcomponents of the Device Driver are further explained in sections 4.2.5 and 4.2.6.

Finally, it is possible to inject arbitrary data into the Device Driver by means of calling the `digest` method. The Device Driver is responsible for processing the data coming through this method in the same way that real device data is processed. This enables testing of the Device Driver without requiring a physical hardware device, and furthermore provides the technical foundation for tooling such as the Hand Recorder (section 4.2.1.5)

As part of this work, two implementations of the Device Driver interface are provided: `LeapMotionDeviceDriver` and `ThreadedLeap2Driver`, both providing support for the Leap Motion Device. While the `LeapMotionDeviceDriver` only provides minimal support for the most basic functionality, the `ThreadedLeap2Driver` provides a full implementation of all required functionality. Additionally, this implementation is spawning a Web Worker (see section 3.1.5.2) internally, so all data ingestion and preprocessing is performed in a separate thread, ensuring performant operation for the rest of the system.

4.2.4 Persistence Framework

It can be argued that the User Experience of a complex software product improves significantly if it has the capability of persisting the User Configuration over subsequent visits. If this were not the case, the User would have to reenter any settings that have been made previously each time the application is restarted. In the context of Web Applications, various possibilities exist for persisting data, such as storing and loading the data from a Backend, saving the data using Browser Cookies, or using more recent Web Specifications such as Local Storage or IndexedDB.

All of these technologies have several advantages and disadvantages. In addition, it does not actually matter for the end user *how* the data is stored in most scenarios, as long as it is stored in the first place. The system thus provides a `PersistenceProvider` Interface, which contains methods representing create, read, update, and delete functionality for persisted records. The System is shipped with one ready to use implementation of the `PersistenceProvider` interface, which is named `IndexedDBPersistenceProvider`, using IndexedDB as the concrete persistence technology. The `PersistenceProvider` is available to the whole application, as it is registered in the global application store (see figure 4.1).

4.2.5 Stream pre-processing framework

As mentioned briefly in section 4.2.1, the user has the option to configure various data pre-processors. The goal of pre-processors may be to provide more accurate data to subsequent system components, or reduce workload by stopping unnecessary data from further propagating through the system. The configuration, which pre-processors have been activated by the user and how they are configured, is supplied to the device facade, and subsequently, the device driver implementation, on each change. In the case of pre-processors, the configuration consists of an array of configuration objects, in turn consisting of a unique pre-processor identifier, and an array of pre-processor specific configuration options. The Device Driver implementation should now call the provided entrypoint to pre-processing framework, named `PreProcessingResolver`, with the received configuration. The `PreProcessingResolver` internally looks up the preprocessing identifier in the `PreProcessingRegistry`, where the identifier is resolved to a class reference implementing the desired preprocessor. The class reference is then instantiated by calling its constructor with the supplied configuration options, and returned from the resolver.

The Preprocessors themselves must implement the standard ReactiveX Operator interface. The Operator interface is primarily used internally in the rx.js library in order

to implement the ReactiveX Operators (such as `map` or `filter`), but implementing the interface in order to develop custom Operators is possible and the official recommendation for creating reusable Observable transformations [22]. This architecture allows multiple Preprocessors to be chained, allowing for complex data cleansing and refinement pipelines to be implemented.

Preprocessors may be device specific or generic. Device specific Preprocessors are allowed to access the data supplied by the device, and thus unusable for other devices which likely send a different data format,

As part of this work, a number of example Preprocessors have been defined.

4.2.5.1 Destroy Useless Frames

The `DestroyUselessFramesPreProcessor` is a configuration-less Leap Motion device-specific preprocessor which stops frame propagation when the `hands[]` array of the frame is empty, and thus likely unsalvagable by the rest of the System. The `hands[]` array is empty if and only if no hand has been detected by the Leap Motion Device at that point in time. The PreProcessor reduces processing load and provides more accurate data to the rest of the system in most cases[2].

4.2.5.2 Drop N Frames

The `DropNFramesPreProcessor` is a generic preprocessors which drops every n-th frame. The variable `n` is supplied through configuration. Its primary purpose is to reduce load for slow devices at the cost of some accuracy.

4.2.5.3 Limit FPS

The `LimitFPSPreProcessor` is a generic preprocessor which throttles the device framerate to a specific number per second, as supplied through configuration.

4.2.6 Classification framework

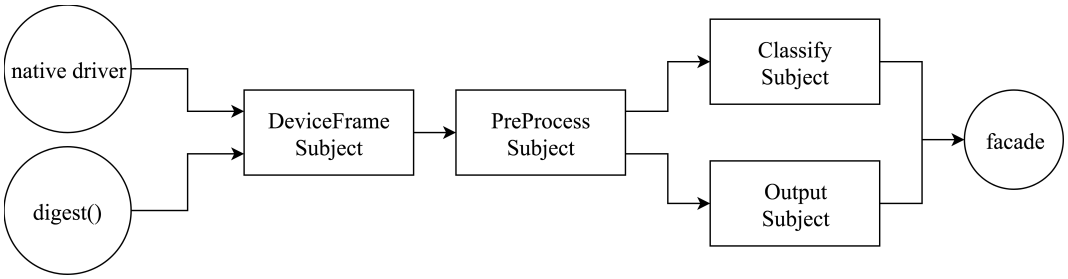


Figure 4.6: Recommended Data flow pipeline for Device Driver implementations.

Figure 4.6 gives an overview over the recommended reactive data flow pipeline for Device Driver Implementations. Data is first ingested into a ReactiveX Subject named

[2]The Leap Motion Device sometimes reports empty frames even if a hand is positioned above the sensor. As all empty frames are omitted by this preprocessor, the overall accuracy arguably increases

```
bufferTime(w, f)
    >>= map(hof_thumb)
    >>= filter(pred)
    >>= map(hof_deriv)
    >>= map(classify)
```

Figure 4.7: Composition of ReactiveX Operators suitable for detecting Thumb Abduction

`DeviceFrameSubject` from either the physical data source or the Device Drivers `digest` method. Afterwards, the data is preprocessed according to any configured preprocessing operators and afterwards fed into the `PreProcessSubject`. After Preprocessing, the data is directly relayed back to the facade, and, subsequently, the frontend, in order to provide it to relevant frontend components (i.e. Hand Loggers and Games). Additionally, the Data is fed into the `ClassifySubject`, where the Hand Exercise Classification occurs.

The Classification is functioning in a similar fashion to the Preprocessing. The Device Facade receives data from the Frontend containing an Exercise ID, and Exercise specific configuration data, which is relayed to the Device Driver, where the ID is resolved to the concrete Classification Operator. However, while the Preprocessing Operators both input and output Hand Tracking Data, the Classification Operators only receive the Hand Tracking Data as Input and produce data of type `ClassificationData` as output. The `ClassificationData` type may contain information regarding what kind of Exercise has been detected, how precisely the Exercise has been executed, how likely the result was a result of cheating, and other potentially relevant, Exercise specific monitoring information.

As part of this work, an example implementation for one Exercise Classifier has been provided.

4.2.6.1 Thumb Abduction Exercise

A common exercise required in ergo therapy is to spread the thumb from the otherwise closed hand as far as possible, an exercise known as Thumb Abduction[3, sec. 4.1.5, par. Abduction]. By composing several ReactiveX Operators, as illustrated in Figure 4.7, the classification algorithm for this exercise can be provided to the framework.

Using the ReactiveX `bufferTime(`w`, `f`)` Operator, the incoming preprocessed device frames, represented by the Observable O, can be accumulated over a window w in an array representing that timeframe. This accumulation is executed every f milliseconds. The application of the Operator returns a new Observable, O', emitting arrays of device frames that were emitted by O in the specified time period. For example, applying `bufferTime(200, 50)` to O returns a new Observable O', emitting arrays containing the emissions of O in the timeframe $t_1 = [0; 200]ms$, $t_2 = [50; 250]ms$, $t_3 = [100; 300]ms$, etc.

By applying the `map(`hof`)` Operator to O', the emitted arrays of device frames can now by transformed into arrays of specific tracking information required for detecting

the Thumb Abduction. The higher order function *hof* required as an argument to the Operator must be implemented for each Device that should support the Abduction Exercise. In the provided implementation, the *hof* transforms the Device Frames into the tip position of the first detected hand of the frame[3]. The result of the `map(`*hof*`)` Operator is now a new Observable O'' emitting arrays of thumb tip positions, representing the timeframes as configured by the previous `bufferTime` transformation.

By applying the `filter(`*pred*`)` Operator to O'', a new Observable $O^{(3)}$ is returned containing only emissions of O'' that satisfy the predicate function *pred*. In this algorithm, the predicate function compares the difference between the maximum and minimum values of the emitted arrays to a variable *threshhold*, denoting the minimum distance in millimeters the thumb has to be spread in order to register as a successful exercise execution.

The `map` Operator is now applied to Observable $O^{(3)}$, taking the first derivative of the emitted thumb positions. The resulting Observable, $O^{(4)}$, now emits arrays representing the change of thumb positions in the configured timeframe (thumb tip velocity).

In a final application of the `map` Operator, it is first determined if a zero intersection can be found in the emissions of $O^{(4)}$. This would mean that the timeframe contains data indicating that a thumb abduction, followed by an adduction (return of the extended thumb to the closed hand) has been performed. Finally, if the zero intersection has happened near the center of the emission, the classifier reports that the Exercise has been performed[4].

The tolerance window used to determine if the zero intersection is in the center of the thumb velocity data, as well as all other variables mentioned in this section, are initialized with values giving good empirical classification results, and fully configurable from the frontend in order to adapt to a variety of patients and exercise variations.

4.2.7 Game Execution Framework

As mentioned briefly in section 4.2.1, after the Game List Frontend Component has determined that the system is ready to execute a game, and the User has clicked the play button, the Component hands control to the Game Execution Framework. The Framework is responsible for dynamically loading all additional libraries required by the Game Implementation, initializing the Game, as well as providing lifecycle hooks, classification data, and device tracking data to the Game.

The Interface that all Game Implementations must implement is shown in Figure 4.8.

Once the Game Execution Framework has finished initializing the Game and loaded its third-party dependencies, the `onStart` method is called. Two parameters are supplied to the Game Implementations at that stage. The `config` Parameter currently contains only a reference to the HTML Element where the Game should reside in. The `notifyGameOver` parameter is a function reference that the Game should call once the

[3]This results in adequate detection accuracy, though transforming the device frames into angles between the thumb and index finger would obviously be the better choice when implementing a serious Thumb Abduction classifier.

[4]For reasons of brevity, some transformations responsible for omitting duplicate classification emissions have been left out in the textual explanation of the algorithm.

```
export interface Game {
  onStart: (
    config: GameConfiguration,
    notifyGameOver: () => void
  ) => Promise<void>;
  onPause: () => Promise<void>;
  onResume: () => Promise<void>;
  onStop: (vm: Vue) => Promise<void>;

  onClassificationReceived: (c: ClassificationData) => void;
  onMotionTrackingDataReceived: (m: GenericHandTrackingData) => void;
}
```

Figure 4.8: Interface Definition for Platform Games

Game is over, in order to notify the framework of this event, and appropriate action can be taken. In the `onStart` method, the Game Developer should initialize the Game inside the supplied HTML Element Reference, start Game Execution, and optionally save the `notifyGameOver` function reference for later use.

The Game Execution Framework implements a Pause functionality, which is triggered by pressing the Space Keyboard Button while playing the Game. The Game Execution Framework subsequently delegates the Information that the Player wishes to pause the Game to the Game Implementation, by calling its `onPause` or `onResume` methods. In these methods, the appropriate actions necessary to implement Pause or Resume functionality should be taken by the Game Developer.

The `onStop` method will be called by the Game Execution Framework if one if these conditions is true:

1. After the Game signals to the Framework that the Game is over, by calling the `notifyGameOver` method
2. After the Player presses the `Esc` Keyboard Button
3. After the Player navigates to a different Frontend Component

In the `onStop` method, the Game Developer is responsible for stopping the Game Execution, and releasing any global resources that were required for Game Execution (i.e., the `window.requestAnimationFrame()` handler, Timeouts set using `window.setTimeout()`, etc.). Additionally, the Game Developer may want to display a Game Over screen to the Player, perhaps also containing statistics, such as the final game score, or even a detailed analysis of hand movements and performance throughout the game. In order to make this possible, a reference `vm` to the Game Execution Frameworks Vue Instance is provided as a parameter to the `onStop` method. Using the capabilities of the parent Vue Instance, for example its programmatic routing facilities, accessed through the `vm.$router` property, the Game Developer may redirect the User to any different part of the Application Frontend, where such final reporting may occur.

The `onClassificationReceived` and `onMotionTrackingDataReceived` are called each time the Classification Frameworks currently active Exercise classifier reported a

Figure 4.9: Screenshot of the Browser Window while playing the provided Space Shooter Game.

successful execution, and new motion tracking data was received from the Device Facade, respectively. The Game Developer can use the Data contained in `onMotionTrackingDataReceived` in any way that is useful, for example, to use the Palm Position for moving the Figure the Player controls. Every time data is received through the `onClassificationReceived` method, the Players Game Figure should execute a meaningful action, for example shooting a bullet in the context of an action game.

As part of this paper, an example implementation of a Game has been provided.

4.2.7.1 Example Implementation: Space Shooter

Figure 4.9 shows a Screenshot taken while playing the Game Space Shooter. The Game has been conceived and implemented as part of this research, using the Javascript Graphics Framework p5.js (see section 3.1.4.2). In the Game, the player is controlling a small, triangular spaceship, and is evading polygons, which are randomly flying towards the player from the top of the screen. The game objective is to destroy as many polygons as possible, maximizing the score displayed in the top left corner. The player can do so by executing the configured exercise successfully, which triggers the spaceship to shoot a bullet. For every polygon destroyed by the Player, 20 points are awarded. If a polygon hits the spaceship, it is destroyed and the game is over. The player can move the spaceship horizontally and vertically by moving the palm.

Chapter 5

Recommended Future Works

As described in section 5, the scope of this work was to design and provide implementations for a web based computer aided ergotherapy framework, which is architecurally capable of meeting the requirements outlined in section 1.3. While this work provides a working implementation of the framework, the focus has not been to provide actual implementations for the frameworks subcomponents, such as the preprocessing, exercise classification, and game components. While example implementations have been provided for each of these subcomponents, their function has yet to be verified by the stakeholding therapists. Additionally, more games and exercise classifiers have to be developed in order for the system to be useful to a broad spectrum of patients and ergo therapists.

While the System Architecture outlined in this work is theoretically capable of meeting all functional and non-functional requirements, as elaborated in section impl:reqanalysis, the provided implementation does not include the implementations required in order to fulfill the Monitorability requirement (see section 1.3.1.4). The additional components required in order to fulfill this requirement are outlined in the following section.

5.1 Monitorability: Additional Components

5.1.1 Data Postprocessing Framework

In a similar manner to the preprocessing framework, which is included in the System implementation, an additional framework responsible for handling data after they are fully processed is conceivable. The implementations for this framework would be injected into the existing Game Execution Component. After the Game Execution Component has provided the Device Tracking and Classification data to the Game Implementation, it would also provide the Data to the Data Postprocessing Implementation. Furthermore, the Game Interfaces `onStart` method could be supplied an additional parameter `supplyPostProcessingData(data: PostProcessingData)`, a function reference that the Game Implementation may call if it wishes to supply additional data to the Postprocessing Implementation.

A conceivable Interface Definition for such a Postprocessing Framework is illustrated in Figure 5.1. The Postprocessor would have to implement a methods for opening and closing Sessions. In this context, Sessions represent a period where the patient is training

```
export interface PostProcessingData {
    type: string;
    data: any;
}

export interface PostProcessor {
    openSession(authenticationData: any): Session;
    closeSession(session: Session);
    onDataReceived(session: Session, data: PostProcessingData);
}
```

Figure 5.1: A conceivable Interface Definition for Postprocessors.

using the System. In the simplest case, every Game Execution can be modelled as one Session. For opening the Session, data used to authenticate the patient may be required, and is supplied to the Postprocessor as an argument of the `openSession` method. Once the Postprocessor has opened a session, it returns a reference of it to the caller (in the simplest case, this would be a Session ID). The caller, that being either the Game Execution Framework or the Game itself, may now report to the Postprocessor that relevant monitoring data has accrued by calling its `onDataReceived` method. Arguments to the method may include the Session reference previously obtained by calling the `openSession` method, and arbitrary `PostProcessingData`. It may be useful to group the Post Processing Data using type identifiers, which could be used to separate Device Tracking Data, Classification Data, and other arbitrary Monitoring Data reported by the game. The Postprocessor may now arbitrarily process the data. At the end of a Session, it should be closed by the caller by using the `closeSession` method.

Similarly to the existing Preprocessor Component, is may be useful to support using multiple Postprocessors at the same time. Several implementations of Postprocessors are conceivable.

5.1.1.1 Backend Reporter

One possible implementation of the Postprocessor Interface might be a component `BackendReporter` that logs the accrued Postprocessing data to a remote Backend Server. The Backend could persist the data in a database, perform analytics and evaluation on it, and make the analysis results available again through an API. These insights could now be queried again by the System for purposes of visualization and progress analysis, as outlined in section 5.1.3.

5.1.1.2 Local Reporter

It is also possible to implement the same functionality without a remote backend component. The data could be supplied to another locally running component of the system, perhaps running on a separate Thread by utilizing Web Workers, where the same tasks would be performed.

5.1.2 Backend and Authentication Components

In conjunction with the previously conceived Backend Reporter, an actual remote Backend System performing the evaluation must be implemented. In addition to performing the analytics and making it available in an API, it may be useful to also implement Authentication and User Roles at this stage, so the therapists can remotely view patients progress, and multiple patients are restricted to seeing their own progress. Authentication and Authorization could be provided with minimal development effort by utilizing existing implementations such as the Open Source Software Keycloak by RedHat[1].

Once a choice has been made on how Authentication and Authorization should be implemented, standard Login and Register Components can be trivially added to the Framework, by following the Implementation Choices best practice.

5.1.3 Progress Analysis Dashboard

The Analytics and Evaluation generated by the Postprocessors outlined in section 5.1.1 could be displayed in a Frontend Dashboard Component in order to inform the therapist and patient of the recovery progress. In order to implement this component, a `DataSource` Interface should be implemented for every relevant Postprocessor, so the user remains flexible in how to use the System.

5.1.4 Messaging Platform

After a Backend Component is ready, and a method for Authentication is in place, a Messaging Platform as required as part of the Monitorability Requirement can be implemented trivially, for example by integrating existing Messaging Frameworks such as Socket.IO[2].

5.2 Proposed Enhancements to existing Components

Apart from adding new Features to the System, a few of which have been elaborated on in the previous section, many improvements to existing Component Implementations are conceivable.

5.2.1 Localization

Up until now, all Text Elements in the System are written in English and targeted at end users in Western Countries. Localization could be implemented to every existing textual Component in order to make the system usable to a broader end user spectrum. From a technical standpoint, this feature would be easiest to implement by using an existing Internationalization Plugin for Vue, such as `vue-i18n`[3].

[1] https://www.keycloak.org/
[2] https://socket.io
[3] https://github.com/dkfbasel/vuex-i18n

5.2.2 Local and Remote Persistence

As part of the provided System implementation, Persistence is only implemented for the Device Recorder Frontend Component. Persistence could be trivially provided to the remaining Frontend Components by using the existing Persistence Provider Interfaces.

5.2.3 Classification Metadata

So far, the `ClassificationData` abstract data type does not contain any useful information, apart from the fact that a Exercise has been performed. A lot of other Information could be encoded in this type in the future, such as a `performance` property indicating how well the exercise has been performed, or a `cheatFactor` property indicating how likely the exercise execution was a result of cheating.

Chapter 6

Conclusion

The work presented in this paper confirms that the modern web is mature and performant enough in order to fulfill all imposed requirements on the computer aided ergo therapy system. The provided implementation is complete to the degree that the system is ready to be integrated into recovery sessions. In addition, a pilot phase with motivated patients could be performed in order to collect initial end user feedback. However, in order for the full vision of the patient doing a large portion of the recovery exercises at home, and the therapist being able to closely monitor progress to become a reality, a lot more work is yet to be done. The next logical implementation steps have been detailed in chapter 5. The development time required until the System can be considered complete from an architectural perspective will probably fill the pages of at least one additional Student Research Project, not even considering the design and implementation of Preprocessors, Exercise Classifiers, and Games.

I hope that others might find this work useful for their purposes, and maybe even advance its development in order for it to potentially be considered an important component in the recovery period after hand injuries in the future. In order for this to be possible, I am releasing all materials generated while working on this project under an Open Source License, thus making it accessible to everyone.

Appendix A

Source Code & Deployment

The Source Code Repository, including the System implementation and this documentation is released on GitHub, and accessible to the general public through the URL https://github.com/talkdirty/theraleap.

Furthermore, the System is deployed and accessible to the general public on the deployment platform Github Pages under the URL https://talkdirty.github.io/theraleap/. The Web page will install a Service Worker upon first visit on supporting (recent) Browsers. Thus, once all pages of the Frontend have been retrieved once, the System is able to operate even when offline.

Appendix B

User Manual

Welcome to the User Manual of `theraleap`, a proof of concept implementation of a computer aided ergotherapy platform. With this Software, you will be able to play Games by performing Gestures with your hand. This Manual will help you to get started!

B.1 Getting started

Please complete these steps in order to get started.

B.1.1 Get Chrome

You will need a recent Version of Chrome. Firefox will not work unfortunately because it will not allow us to talk with the Hand Tracking Device due to Security Restrictions. To get Chrome, visit https://www.google.com/chrome/

B.1.2 Install the Leap Motion Driver

Currently, we support only the Leap Motion Device for tracking your hand, though that may change in the Future! In order for us to be able to talk to the Leap Motion Device, you'll have to install the Driver. To get the Driver, visit https://developer.leapmotion.com/sdk/v2 and follow the instructions for your Operating System.

B.1.3 Try it out

Now, after everything is installed, and the Device is plugged in, you're ready to test if everything works! Open up Chrome and visit https://talkdirty.github.io/theraleap/#/debug/status. If all boxes are green, you're ready! If not, follow the instructions you see on screen (it may also take a while for the Leap Motion Device to get ready, so be patient!).

B.2 A brief Overview

You can press the Button on the top left corner to get the Navigation Bar as shown in Figure B.1. From here, you can navigate to all available Subcomponents of the Software.

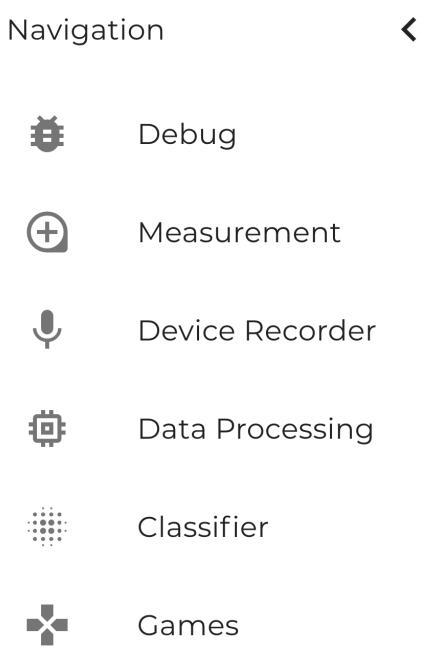


Figure B.1: All Navigation Options of the Software

B.2.1 Debug

This is a section containing Information useful to Developers. You can look at the Data coming from the Device, or a graphical representation of your hand here. You probably won't need to spend too much time here though.

B.2.2 Measurement

Here, you can take the angular measurements between your Fingers, a piece of Information required by the Therapist. In order for the measurement to become more accurate, you can choose to average it over several milliseconds, by putting your desired value in the Input Box. On the left hand side, you will see your hand as the Computer sees it. Make sure it is fully visible and is displaying everything correctly. If something is wrong, and the measurement can't be taken, you will be notified of what you need to do in order to fix the problem on the right side of the screen.

B.2.3 Device Recorder

With this Component, you can record your Hand, and play that Data back to the Framework as if it was coming from the real device. As a User of the System, you probably won't ever need this, but this Component is useful for Developers. To start a new Recording, click the + Button. Now you can title your Recording. You also see a miniature Hand on the right side, if you have your hand above the sensor. Once

you're ready, press the `Record` Button. A progress bar will begin to fill, indicating how much space is left for your recording, once it is full, or you press the `Save` Button, your recording is done. Now you can flip the Switch to activate it. It will then be propagated through the whole System in an endless loop.

If you want to save the Recordings for a later time, make sure the `Persist Recordings` Checkbox in the Settings Tab is checked.

B.2.4 Data Processing

If your Computer experiences performance issues while working with the System, you can attempt to alleviate this a bit here. Each of the Box shown here is representing a specific preprocessing step. For example, you can limit the framerate of the Device to a lower value such as `28 FPS` by turning on the `Limit FPS` Preprocessor. Your system has to do less that way, which will result in a better performance.

B.2.5 Classifier

Here, the Gesture you need to perform in order to control the Games is configured. Each Box represents one specific Gestures. Only one Box can be turned on at a time. Inside the Box, you will find a description of what to do in order to trigger the Gesture, and also some Input Boxes that allow the therapist to configure the Exercise to your needs. He or she might tell you which numbers you should put in here if you train from home.

B.2.6 Games

Finally, here you can choose and play any Game you like. If the Leap Motion Device is attached correctly to your computer, **and** you turned on a Gesture Classifier in the `Classifier` Tab, the Button `Play with Motion Tracking` will be green. Click on it to start the Game. In the game, you can hit `Space` to Pause or `Esc` to Quit if you feel tired. Good luck!

Appendix C

Developer Manual

Developers are advised to read through the main part of the paper in order to gain a general understanding on how the framework functions. Additionally, instructions on how to compile the project, and run the project in a development mode are given in the `README.me`, in the Source Code Repository (see Appendix A). This Manual briefly gives practical advice on how to implement the System Interfaces, namely the `Preprocessing`, `Gesture Classification`, and `Game` Interfaces.

C.1 Adding a Preprocessor

As an example, let's add a Preprocessor for the Leap Motion Device that filters out frames where more than one hand is detected.

C.1.1 Implement the Preprocessor

This Preprocessor is specific to Leap Motion Device data, so be sure to put it into the `src/processing/leap` folder.

```
// src/processing/leap/droptwohands.ts
export const DropTwoHandsId = "DropTwoHands";

export class DropTwoHandsPreProcessor
  implements Operator<GenericHandTrackingData, GenericHandTrackingData> {
  constructor() {}

  public call(
    subscriber: Subscriber<GenericHandTrackingData>,
    source: Observable<GenericHandTrackingData>
  ) {
    return source
      .pipe(
        // filter every frame where the hand
        // array is longer than 1.
        filter((value: LeapHandTrackingData) => value.data.hands.length > 1)
```

```
      )
      .subscribe(subscriber);
  }
}
```

C.1.2 Register the Preprocessor to the Framework

in `src/processing/resolver.ts`, register your Preprocessor.

```
// src/processing/resolver.ts
export const ResolverRegistry: {
  [_: string]: { new (...args: any[]): Operator<any, any> };
} = {
  [DropNFramesPreProcessorId]: DropNFramesOperator,
  [DestroyUselessFramesId]: DestroyUselessFramesOperator,
  [FPSThrottlerId]: FPSThrottler,
  // our new preprocessor
  [DropTwoHandsId]; DropTwoHandsPreProcessor
};
```

C.1.3 Add the new Preprocessor to the Vuex State

To the `preprocessors.ts` state submodule, add a the state of the preprocessor, and implement the `constructConfig` method.

```
// src/state/modules/preprocessors.ts
state: {
    preprocesors: {
        ...
        dropTwoHandsPreProcessor: {
            enabled: false,
            constructConfig: () => {
                identifier: DropTwoHandsId,
                // We don't need args for this
                args: []
            }
        }
        ...
    }
}
```

C.1.4 Add a Description to the Frontend

Create a new Vue Component, containing an `md-card`, describing your Preprocessor. Import and integrate it in the `PreProcessing.vue` Frontend Component. If the switch is flipped by the User, call `modifyPreProcessor`, and commit a state mutation to set the `enabled` property of your preprocessor to `true`. Finally, call `preprocessorSelectionUpdated`. This will tell the Device Driver that the Preprocessors have changed, call the state

function constructConfig internally, and construct your Preprocessor, supplying any arguments to the constructor in the order you specified in the `args` array.

C.2 Adding a Classifier

Let's add a classifier that randomly classifies every nth frame as a Gesture.

C.2.1 Implement the Classifier

```
// src/classify/classifiers/randomtestclassifier.ts
export const RandomTestClassifierId = "RandomTestClassifier";
export class RandomTestClassifier
  implements Operator<LeapHandTrackingData, ClassificationData> {
  constructor(
    // With this parameter we'll configure on which
    // frame we'll emit the Classification
    private n: number,
  ) {}

  public call(
    subscriber: Subscriber<ClassificationData>,
    source: Observable<LeapHandTrackingData>
  ) {
    return source
      .pipe(
          // Buffer n Frames
          bufferCount(this.n),
          // Transform the result into ClassificationData
          map((_ : LeapHandTrackingData[]) => {
              return {
                actionName: "ONE_SHOT",
                metrics: {
                    cheatFactor: 0,
                    quality: 0
                }
              }
          })
      )
      .subscribe(subscriber);
  }
}
```

C.2.2 Register the Classifier to the Framework

In `src/classify/resolver.ts`, register the Classifier so the Resolver knows how to construct an instance of it.

```
export const ClassifierRegistry: {
  [_: string]: { new (...args: any[]): Operator<any, any> };
} = {
  [ThumbSpreadClassifierId]: ThumbSpreadClassifier,
  // our shiny new classifier
  [RandomTestClassifierId]: RandomTestClassifier
};
```

C.2.3 Add the Classifier to the VueX state

Next, let's model the State of the Classifier Frontend Component. In `classifiers.ts`:

```
// src/state/modules/classifiers.ts
state: {
classifiers: {
    ...
    RandomTestClassifier: {
        enabled: false,
        n: 150,
        constructConfig: () => {
            return {
                identifier: RandomTestClassifierId,
                args: [
                    classifier.state.classifiers.RandomTestClassifier.n
                ]
            }
        }
    },
    ...
}
```

C.2.4 Add a Description to the Frontend

Create a new Vue Component, containing an `md-card`, describing your Classifier, how it works, and how to configure it. Import and integrate it in the `Classifiers.vue` Frontend Component. If the switch is flipped by the User, or any Configuration property has changed call `classifierSelectionUpdated`.

C.3 Adding a Game

C.3.1 Implement the Game

If your Game needs any third party assets, such as images, put them into the `src/dist` directory, or alternatively import them using Webpack.

To add a Game, add a class implementing the `Game` interface, and make it the default export. In the class, you can import any third party libraries you may need. All of these

will be lazy loaded by the Game Execution Framework, so you don't need to worry about making a negative impact on bundle size.

C.3.2 Register the Game to the Framework

Next, tell the Framework how to construct the game. The `import()` function should receive as an argument the path to the class implementing the `Game` Interface (or alternatively, the directory name. This works if there is an `index.ts` file in the directory.).

```
// src/games/resolver.ts
export const GameResolveMapping: { [key: string]: () => Promise<any> } = {
  "space-shooter": () =>
    import("@/games/space-shooter").then((imp: any) => {
      return imp.default;
    }),
  // our new game
  "new-game"; () =>
    import("@/games/new-game").then((imp: any) => {
      return imp.default;
    }),
};
```

C.3.3 Add the Game to the GameList

Finally, add the Game to the GameList component. Add a new card containing a brief description and title. Use the existing Button Components for rendering the Play Buttons. On click, call `play('new-game')` (the key you put into `GameResolveMapping`).

References

Literature

[1] Ali Alabbas and Joshua Bell. *Indexed Database API 2.0*. Tech. rep. W3C, 2018. URL: https://www.w3.org/TR/IndexedDB-2/ (cit. on p. 9).

[2] United States Government Army. *Systems Engineering Fundamentals*. Defense Acquisition University Press, 2001, p. 36 (cit. on p. 2).

[3] Moritz Jakob Baumotte and David Volz. "Konzeption und Entwicklung einer Benutzerschnittstelle für computergestützte Ergotherapie" (2017). unpublished (cit. on pp. 2, 5, 18, 23).

[4] Gavin Bierman, Martín Abadi, and Mads Torgersen. "Understanding TypeScript". In: *ECOOP 2014 – Object-Oriented Programming*. Ed. by Richard Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 257–281 (cit. on pp. 10, 14).

[5] Lawrence Chung et al. *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media, 2012, p. 1 (cit. on p. 3).

[6] Ahmed Ernaggar and Dirk Reichardt. "Analyzing Hand Therapy Success in a Web-Based Therapy System". In: *Proceedings of the ABIS 2016*. Aachen, 2016 (cit. on p. 5).

[7] Ahmed Ernaggar and Dirk Reichardt. "Digitizing The Hand Rehabilitation Using the Serious Games Methodology With a User-Centered Design Approach". In: *Proceedings of the 2016 International Conference on Computational Science and Computational Intelligence (CSCI'16)*. Las Vegas, 2016 (cit. on p. 5).

[8] I. Fette and A. Melnikov. *The WebSocket Protocol*. Tech. rep. IETF, 2011. URL: https://tools.ietf.org/html/rfc6455 (cit. on p. 10).

[9] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995 (cit. on pp. 7, 19).

[10] Erik Guttman. "Exergames: More fun with rehabilitation and exercise through games". *Rehacare International Magazine* (2018) (cit. on p. 1).

[11] Ian Hickson. *Web Workers Working Draft*. Tech. rep. W3C, 2015. URL: https://www.w3.org/TR/workers/ (cit. on pp. 10, 15).

[12] Dean Jackson and Jeff Gilbert. *WebGL 2.0 Specification*. Tech. rep. Khronos Working Group, 2018. URL: https://www.khronos.org/registry/webgl/specs/latest/2.0/ (cit. on p. 9).

[13] Maryam Khademi et al. “Free-hand interaction with leap motion controller for stroke rehabilitation”. In: *CHI’14 Extended Abstracts on Human Factors in Computing Systems*. ACM. 2014, pp. 1663–1668 (cit. on p. 5).

[14] David R. Michael and Sandra L. Chen. *Serious Games: Games That Educate, Train, and Inform*. Muska & Lipman/Premier-Trade, 2005 (cit. on p. 1).

[15] Microsoft. *Kinect for Windows SDK Developer Manual*. Tech. rep. MSDN, 2018. URL: https://msdn.microsoft.com/en-us/library/dn435664.aspx (cit. on p. 20).

[16] Andreas Rossberg. *WebAssembly Core Specification*. Tech. rep. W3C, 2018. URL: https://www.w3.org/TR/wasm-core-1/ (cit. on p. 15).

[17] Alex Russel et al. *Service Workers 1*. Tech. rep. W3C, 2017. URL: https://www.w3.org/TR/service-workers-1/ (cit. on pp. 10, 15).

[18] Statistisches Bundesamt, Robert Koch Institut. *Gesundheitsberichterstattung des Bundes: Diagnosedaten der Krankenhäuser ab 2000 (Eckdaten der vollstationären Patienten und Patientinnen). S60-S69 Verletzungen des Handgelenkes und der Hand*. URL: http://www.gbe-bund.de (cit. on p. 1).

[19] Cem Yuksel, Scott Schaefer, and John Keyser. “Parameterization and Applications of Catmull-Rom Curves”. *Computer Aided Design* 43.7 (2011), pp. 747–755 (cit. on p. 17).

[20] Nick Qi Zhu. *Data visualization with D3. js cookbook*. Packt Publishing Ltd, 2013 (cit. on p. 9).

Online sources

[21] Mozilla Contributors. *Mozilla Developer Network - IndexedDB: Browser storage limits and eviction criteria*. 2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria (cit. on p. 9).

[22] rx.js Contributors. *rx.js manual: Operator Creation*. 2017. URL: https://github.com/ReactiveX/rxjs/blob/master/doc/operator-creation.md (cit. on p. 22).

[23] Webpack Contributors. *Webpack Concepts*. 2018. URL: https://webpack.js.org/concepts/ (cit. on p. 10).

[24] The ReactiveX Developers. *ReactiveX: Intro*. 2018. URL: http://reactivex.io/intro.html (cit. on p. 7).

[25] The VueX Developers. *What is Vuex?* 2018. URL: https://vuex.vuejs.org/ (cit. on p. 6).

[26] Peter Ehrlich. *Leap Motion: Protocol*. 2014. URL: https://github.com/leapmotion/leapjs/wiki/Protocol (cit. on p. 8).

[27] Isaac Gouy. *The Computer Language Benchmarks Game. Web*. 2018. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/ (cit. on p. 12).

[28] Facebook Inc. *Flux: Application Architecture for Building User Interfaces*. 2014-2015. URL: https://facebook.github.io/flux/ (cit. on p. 6).

[29] GitHub Inc. *Most popular Repositories, sorted by most Stars*. 2018. URL: https://github.com/search?q=stars:%3E1&s=stars&type=Repositories (cit. on p. 6).

[30] Google Inc. *Core app quality*. 2018. URL: https://developer.android.com/docs/quality-guidelines/core-app-quality (cit. on p. 7).

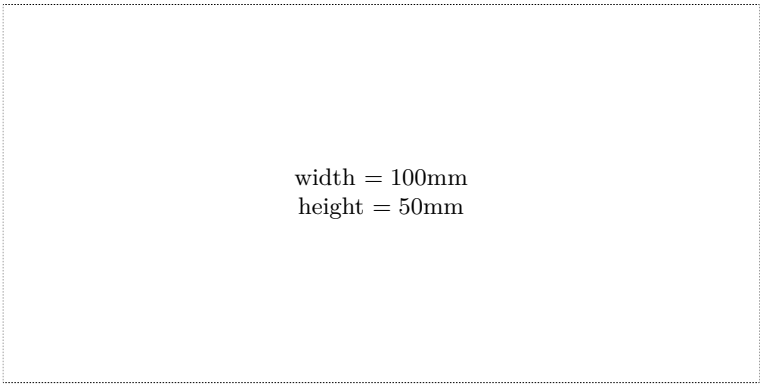
[31] Lauren McCarthy. *p5.js overview*. 2018. URL: https://github.com/processing/p5.js/wiki/p5.js-overview (cit. on p. 9).

[32] Marcos Moura. *Material Design for Vue.js*. 2018. URL: https://vuematerial.io/ (cit. on p. 7).

[33] Evan You. *Comparison with Other Frameworks*. 2018. URL: https://vuejs.org/v2/guide/comparison.html (cit. on p. 6).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —