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

Acronyms

AR Augmented Reality. 2

FPS Frames per Second. 11

NFR non-functional requirement. 3, 6

ReactiveX Reactive Extensions. 10

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 [17]. 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 Solution Design

1.4.1 Available Alternatives

Based on the requirements, which have been gathered and derived on 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.

1.4.1.1 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[3].

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 [23]. The question of whether Javascript is performant enough to tackle the task at hand has to be clarified before this alternative can be considered feasible.

1.4.1.2 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.

1.4.1.3 Web Application with remote server component (backend)

Following up on the design outlined in section 1.4.1.2, 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 1.4.1.1, 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.

[3]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.

1.4.1.4 Desktop Application

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

Table 1.1: Comparison of implementation alternatives based on estimated requirement fulfillment

	Functional Reqs.	Modularization	Performance	Availability	Deployment	Extensibility
Desktop App (1.4.1.4)		X	X	X		
Web thin client (1.4.1.2)	X	X			X	X
Web local client (1.4.1.3)	X	X	X	X		
Web fat client (1.4.1.1)	X	X	?	X	X	X

1.4.2 Elected Alternative

Table 1.1 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 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 1.4.1.3 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 1.4.1.2 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 [16]. 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 [15].

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.

1.5 Outline and structure

Section 1 gave a general overview of the project motivation, scope, and method. Additionally, the project requirements were outlined, and possible project architectures were presented and evaluated. The section concludes with the design decision that a fully featured Web Application is the most desirable target architecture given the requirements.

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 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 popular code sharing platform GitHub [25]. 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 [29].

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 [21][24]. 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 [26]. 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 [28].

3.1.2 Reactive Extensions and rx.js

Reactive Extensions (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 [20], 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*.

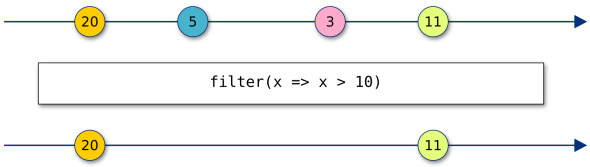


Figure 3.1: An example Observable transformation, represented using *Rx Marble Diagrams*.

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).

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.

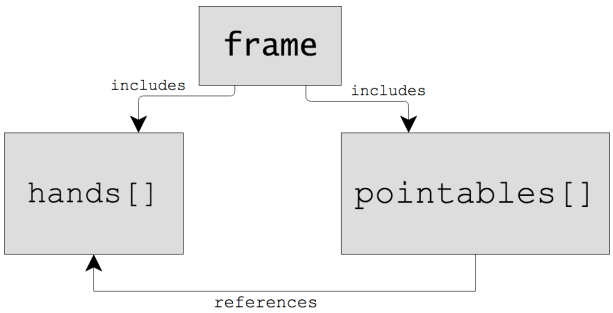


Figure 3.2: High Level view of the Leap Motion Device Frame Data

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. 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 metainformation, 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 pointed objects 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 [22]. Finally, the pointable is associated to a hand by referencing its ID.

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 [27].

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 [18, 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, with 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] [19]. 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].

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 inpact 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].

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

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 [16, 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

3.2.2 Typescript: static typing

3.2.3 Inversify: Inversion of Control

3.2.4 jest and sinon.js: Unit Testing and Mocking

Chapter 4

Framework Implementation

4.1 System Architecture

4.2 Development Pipeline

Webpack 4
- VueJS 2, Vue Router, VueX, inversify Dependency Injection
- Karma Unit Tests
- THREE.js

4.3 Implemented Subsystems

4.3.1 Device Driver Interface

Describe Generalized Device Driver Interface. Point out that adding different devices is possible with this architecture.

4.3.2 Device Facade

Justify for a need of a Facade in Front of the raw Device Driver: Seperation of Concerns and Recording (Mock data) Functionality. Ease of Testing.

4.3.3 Device Debug Interface

Describe Component: Raw Device Logger, Device Status Log, Device Graphical Log

4.3.4 Graphical Hand Logger

THREE.JS, OrbitControls, Smoothing, Prop Configuration

4.3.5 Device Recorder

Record Segments of hand movements in order to aid in development, make classification errors reproducible

4.3.6 Persistence Provider

Describe Abstract Persistence Provider Interface
    Describe Concrete Persistence Provider Interface Implementation: IndexedDB

4.3.7 Preprocessing Framework

Justify need for Preprocessing: Lots of useless data coming from the device. Preliminary clean up of data may be relevant for all classifiers

4.3.8 Classification Framework

Describe how Classifiers receive the preprocessed data frame stream, and transform the stream in order to emit another stream of classifications, along with relevant metadata

4.3.9 Game Execution Engine

Describe how Games are receiving the Classification Stream and using that in order to drive the gameplay.

Chapter 5

Recommended Future Works

5.1 Proposed Subsystems

5.1.1 Data Postprocessing Framework

Generic interface that does something with classification data / game data. For example logging it to a remote location, like a backend

5.1.2 Backend Component

auth -> classification data -> backend

5.1.3 Progress Analysis Dashboard

Component that gets patients postprocessed classification / game results, and visualizes progress

auth -> backend -> select patient -> patient view

5.1.4 Messaging Platform

Therapist Requirement. Ability to send messages to / from patients

5.2 Proposed Enhancements to existing Components

5.2.1 Classification Metadata

Anti Cheat (Therapist requirement)

Log Classification Specific relevant Metadata for Therapist Analysis

Chapter 6

Conclusion

Possibility of Project Confirmed, Modern Web is evolved enough to tackle this task. But lots of extensions should be made in order to make the project actually useful

6.1 Ciritizism

Security leapd

Appendix A

User Manual

A.1 Installing the Hardware Device Driver

A.2 Verifying the Installation Success

A.3 Configuration

A.3.1 Preprocessors

A.3.2 Classificators

A.4 Gameplay

Appendix B

Developer Manual

B.1 Building and Running the Project in Development Mode

B.2 Executing Unit Tests

B.3 Extending the Framework

B.3.1 Adding a Preprocessor

B.3.2 Adding a Classifier

B.3.3 Adding a Game

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. 12).

[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, 8).

[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 p. 7).

[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. 8).

[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. 8).

[8] I. Fette and A. Melnikov. *The WebSocket Protocol*. Tech. rep. IETF, 2011. URL: https://tools.ietf.org/html/rfc6455 (cit. on p. 13).

[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 p. 10).

[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. 7, 12).

[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. 12).

[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. 8).

[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] Andreas Rossberg. *WebAssembly Core Specification*. Tech. rep. W3C, 2018. URL: https://www.w3.org/TR/wasm-core-1/ (cit. on p. 7).

[16] Alex Russel et al. *Service Workers 1*. Tech. rep. W3C, 2017. URL: https://www.w3.org/TR/service-workers-1/ (cit. on pp. 7, 13).

[17] 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).

[18] Nick Qi Zhu. *Data visualization with D3. js cookbook*. Packt Publishing Ltd, 2013 (cit. on p. 12).

Online sources

[19] 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. 12).

[20] The ReactiveX Developers. *ReactiveX: Intro*. 2018. URL: http://reactivex.io/intro.html (cit. on p. 10).

[21] The VueX Developers. *What is Vuex?* 2018. URL: https://vuex.vuejs.org/ (cit. on p. 9).

[22] Peter Ehrlich. *Leap Motion: Protocol*. 2014. URL: https://github.com/leapmotion/leapjs/wiki/Protocol (cit. on p. 11).

[23] Isaac Gouy. *The Computer Language Benchmarks Game. Web.* 2018. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/ (cit. on p. 5).

[24] Facebook Inc. *Flux: Application Architecture for Building User Interfaces*. 2014-2015. URL: https://facebook.github.io/flux/ (cit. on p. 9).

[25] 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. 9).

[26] Google Inc. *Core app quality*. 2018. URL: https://developer.android.com/docs/quality-guidelines/core-app-quality (cit. on p. 10).

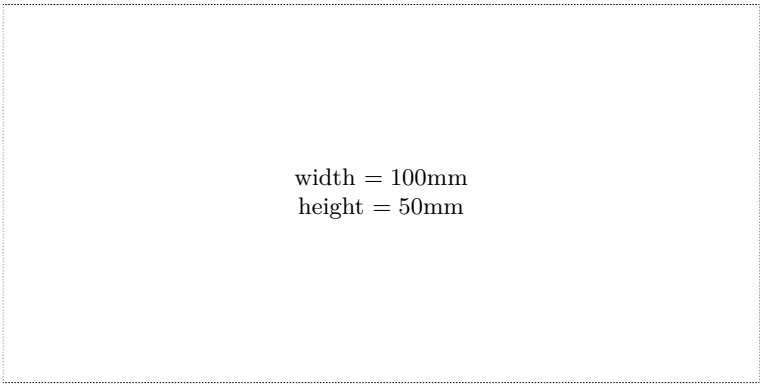
[27] Lauren McCarthy. *p5.js overview*. 2018. URL: https://github.com/processing/p5.js/wiki/p5.js-overview (cit. on p. 12).

[28] Marcos Moura. *Material Design for Vue.js*. 2018. URL: https://vuematerial.io/ (cit. on p. 10).

[29] Evan You. *Comparison with Other Frameworks*. 2018. URL: https://vuejs.org/v2/guide/comparison.html (cit. on p. 9).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —