DM

Task Generator 2.0
Leveraging personalized paper and pencil cognitive training to a tablet approach

MASTER DISSERTATION

Tomás Francisco Andrade Marcos
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

December | 2023

Task Generator 2.0
Leveraging personalized paper and pencil cognitive training to a tablet approach

MASTER DISSERTATION

Tomás Francisco Andrade Marcos
MASTER IN INFORMATICS ENGINEERING

SUPERVISION
Filipe Magno Gouveia Quintal
Sergi Bermúdez i Badia
Ana Lúcia dos Santos Faria

Faculdade de Ciências Exatas e da Engenharia

Mestrado em Engenharia Informática

Task Generator 2.0: leveraging personalized paper and pencil cognitive training to a tablet approach

Tomás Francisco Andrade Marcos 2037017

Orientado por:

Filipe Magno Gouveia Quintal

Ana Lúcia Faria

Sergi Bermúdez i Badia

Monday 1st April, 2024

Resumo

As deficiências cognitivas afetam a capacidade de uma pessoa para realizar atividades diárias. Existem evidências que sustentam o fato de que, por meio da reabilitação, é possível mitigar os efeitos do declínio cognitivo. Embora abordagens tradicionais sejam amplamente aceites, tais intervenções são limitadas devido ao tempo e aos recursos humanos. Com os avanços na tecnologia, novas soluções surgiram no campo das intervenções cognitivas computadorizadas. O NeuroRehabLab Task Generator (NTG) é uma ferramenta online gratuita que gera facilmente tarefas em papel e lápis altamente personalizadas que abordam vários domínios cognitivos. A tese atual aprimora as capacidades do NTG, implementando o algoritmo de geração de tarefas aleatórias do NTG e integrando-o em uma aplicação web. O foco da nova versão é o conteúdo interativo, que permite aos usuários executar imediatamente as tarefas geradas, ao mesmo tempo em que permite aos clínicos avaliar o desempenho do paciente. Propomos uma metodologia iterativa na qual cada tarefa presente no NTG será adaptada, testada e avaliada.

Keywords: Deficiências cognitivas · Task Generator · Tarefas · Frontend · Backend

Abstract

Cognitive impairments affect a person's ability to perform daily activities. There is evidence that support the fact that through rehabilitation it is possible to mitigate the effects of cognitive deterioration. Although traditional approaches are highly accepted, these interventions are limited due to time and human resources. With the advancements in technology, new solutions have emerged in the field of computerized cognitive interventions. The NeuroRehabLab Task Generator(NTG) is a free online web-tool that easily generates highly personalized paper-and-pencil tasks that address multiple cognitive domains. The current thesis, enhances the capabilities of the NTG, implementing the randomized task generation algorithm from NTG and integrating it into a web application. The focal point of the new version is the interactive content, that allows users to immediately perform the generated tasks, while also allowing clinicians to evaluate the patient's performance. We propose an iterative methodology in which each task present in the NTG will be ported, tested and evaluated.

Keywords: Cognitive impairment · Task Generator · Tasks · Frontend · Backend

Acknowledgements

My academic journey is now coming to an end. Over the many years of academic activities, I have met lots of people whom have had a tremendous impact in my pursuit of becoming a better individual.

I would like to express my deepest gratitude to my family, especially my parents and my sister, for all the emotional, financial support they gave me throughout the years, and for always being there even in times of hardship and failure.

I would like to extend my sincere thanks to my friends, who provided support and understanding when I needed it most, thank you for your friendship and camaraderie.

Thanks should also go to all the professors who were part of this long journey. I would like to express my deepest appreciation to my elementary school teacher Isabel Gonçalves for teaching me be more diligent and to thrive to improve myself, to my class director Cláudia Gouveia in junior high for her patience e commitment. To my thesis advisor Filipe Quintal, I would like to express my sincere thanks for all the patience, dedication, availability and the knowledge transferred that allowed me to conclude this thesis. To my co-advisors, Ana Lúcia Faria and Sergi Bermúdez i Badia, I am also thankful for all support and the feedback that was given to me during the realization of this project.

Lastly, I'd like to mention my colleagues whom I've had the pleasure to work with. Thanks to them, I learned to work with people who have different visions and perspectives

Table of Contents

List of Figures

List of Tables

Lista de Acrónimos

CSS Cascading Style Sheet

DOM Document Object Model

FDD Feature Driven Development

GNPT Guttmann Neuro Personal Trainer

HCI Human-Computer Interaction

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

MCI Mild Cognitive Impairment

MUI Material UI

NTG NeuroRehabLab Task Generator

PDF Portable Document Format

SUS System Usability Scale

TG Task Generator

UI User Interface

W3C World Wide Web Consortium

WHO World Health Organization

1 Introduction

Some of the consequences of the acquired, or degenerative neurological diseases are cognitive, motor and behavioral impairments. Cognitive impairments inhibit a person's ability to remember, learn, concentrate and effectively perform tasks with cognitive load, thus affecting a person's ability to perform basic and instrumental activities of daily living with ease [1]. Mild Cognitive Impairment (MCI) increases the risk of developing dementia. About 39% of those diagnosed with MCI develop dementia [2], which is currently the seventh leading cause of death and one of the major causes of disability and dependency among older people worldwide [3]. Dementia leads to the deterioration of cognitive functions in memory, thinking, orientation, comprehension, calculation, learning capacity, language, and judgement, without affecting consciousness. By 2050 older people should increase 35%, which raises the number of people at risk of developing dementia [3].

The World Health Organization reveals that over 1 billion people worldwide suffer from neurological disorders. These range from epilepsy to Alzheimer's disease, from stroke to headache, also including brain injuries, neuro infections, multiple sclerosis and Parkinson's disease. [4]

Given these concerning numbers, there is undeniably a need to develop tools to help those affected by these neurological disorders through rehabilitation to improve their cognitive functions. Evidence shows that cognitive interventions have benefits in maintaining cognitive functions and possibly restoring cognitive functions by taking advantage of the nervous system's plasticity. [5]

Traditional approaches to cognitive rehabilitation require a face-to-face approach, meaning that both the patient and the psychologist need to meet at a determined location, usually clinics, healthcare centers or nursing homes, and coordinate session schedules. These factors hamper the frequency, intensity and quality of these interventions since psychologists struggle to manage their patient's needs adequately and on time. Overall, the high demand for human resources and the need for transportation for the patients might limit the intensity and length of the treatment, thus compromising their impact [6, 7].

Information and Communication Technologies based solutions have the potential to tackle the limitations of traditional paper-and-pencil tasks. The use of these technological advancements allows for improvements in the quality of clinical services by enhancing the intensity and personalization of tasks [7] and allows for the extension of therapies outside of clinical facilities, for example, patients' homes. Subsequently, patients can perform part of their intervention sessions at home, thus reducing transportation costs and logistics [6].

In this thesis, we propose the "NeuroRehabLab Task Generator 2.0", an upgrade to the already existing "NeuroRehabLab Task Generator" [7, 8] This new version is designed to be compatible with tablets, enabling remote performance monitoring of patients by healthcare providers. Additionally, it aims to facilitate assessments for clinicians, extend cognitive training sessions to patients' homes, and streamline the creation of personalized cognitive training programs.

1.1 Problem Description

In the present time, there is an increasingly higher demand for tools to help people with neurological disorders. Although there are many computer based tools available, these tools are, fundamentally, technology based tools, requiring the use of technological devices which some clinical facilities might not have. Also, some of these tools require a paid subscription that some clinics might not be able to afford.

The "NeuroRehabLab Task Generator" [7] is a free tool that generates paper-and-pencil tasks in PDF format. Although paper-and-pencil tasks are commonly used in clinical settings due to their low cost, they (the tasks) are still tangible. This means that the tasks generated by the Task Generator need to be printed, executed by the patient and delivered back to the clinician to be evaluated. This process entails additional costs, related to time, transportation and human resources, not to mention the printing costs

1.2 Proposed Solution

Due to the problem described above, it is important to upgrade the existing Task Generator so that it can generate the same tasks but have them (the tasks) interactive.

That being said, to make it possible, there are some aspects of the current version that need to be to be taken into consideration. The task generation and execution logic must be kept in the proposed solution.

- The current algorithms to generate each task need to be kept
- The models for each task need to be kept
- The stimuli needed to fulfil each task must be the same as in version 1.0 of the system.

With the above statements in mind, another topic to touch upon is that this new version of the Task Generator is meant for tablet devices, meaning that the entire application needs to be re-designed while also considering Human-Computer Interaction (HCI) principles to make it user-friendly. It is also important to note that people with neurological disorders do not perceive an application the same way a healthy person does. Thus, the new design must take that into consideration.

To tackle this endeavour, we began by conducting extensive tests to understand how the previous version generated cognitive training tasks. Following this comprehensive analysis, multiple programming languages and frameworks were tested evaluating their characteristics and ease of use. This process informed our decision on which technologies would be employed for the final application version.

Some of the requirements that the final application needs to met are listed below:

- Abstract the model of tasks from how the tasks are presented (view)
- Accessible User Interface (UI)
- Responsive layout - the content adjusts to the size of the screen
- Interactive content

2 Literature Review

This section, will be used to briefly present the state of the art in which of neurological rehabilitation tools, such as the "NeuroRehabLab Task Generator 2.0". We also present similar projects and other research projects that have been published to inspire and influence the development of the Task Generator 2.0. Some of the research include rehabilitation projects, guidelines for application development and HCI principles.

2.1 Similar projects

In an era marked by rapid technological advancements, the landscape of cognitive training and rehabilitation has seen a notable shift towards computer-based solutions. A myriad of studies [9, 10] has delved into the analysis of various cognitive stimulation, training, and rehabilitation tools, tailored to diverse populations facing different cognitive conditions.

This section serves as a comprehensive exploration of influential projects that have significantly contributed to the evolution of the Task Generator 2.0. By examining key initiatives in cognitive rehabilitation, we aim to draw insights, identify best practices, and illuminate the innovative pathways that have paved the way for the development of our enhanced cognitive training tool.

2.1.1 NeuroRehabLab Task Generator

The development of the new version of the NeuroRehabLab Task Generator was intricately linked to the foundation laid by its predecessor. The existing "Task Generator" played a pivotal role in shaping the trajectory of the enhanced version. In this section, we delve into the evolution of the Task Generator, exploring its inception, key methodologies employed, and the transformative steps that paved the way for a more advanced and adaptable cognitive training tool.

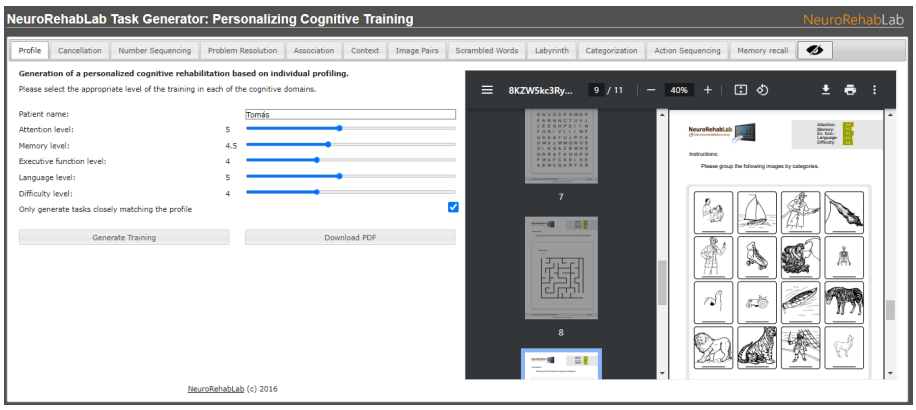


Fig. 1: Task Generator

The "NeuroRehabLab Task Generator" stands as a noteworthy web tool designed to dynamically create PDF-based cognitive rehabilitation tasks tailored to individual patient needs. Developed with a focus on cognitive rehabilitation programs, these tasks are meticulously parameterized to target various cognitive domains, including Attention, Memory, Executive Functions, and Language [7], as shown in Figure 1.

The process of task selection involved a participatory design approach, engaging experts in the field. In the initial stages of its creation, three rehabilitation experts - two seasoned neuropsychologists with extensive experience in cognitive assessment and interventions related to stroke and dementia, alongside an accomplished rehabilitation technology researcher - conducted a meticulous analysis of methodologies employed in clinical rehabilitation settings. This comprehensive exploration spanned various healthcare environments, including public hospitals, private clinics, and senior houses.

Drawing from their observations and insights gained from clinical practice, the experts systematically documented and curated a collection of 20 distinct paper-and-pencil task types commonly

utilized in cognitive training. This compilation included tasks sourced from published training material and those prevalent in clinical rehabilitation settings. To further refine the selection of tasks, the above mentioned experts proceeded to rank the identified cognitive tasks, narrowing the selection to 11 tasks that stood out as pivotal components of effective cognitive training programs. The chosen tasks include word search, problem resolution, numeric sequences, action sequencing, association, cancellation, categorization, comprehension of contexts, image pairs, mazes, and memory of stories [7, 8].

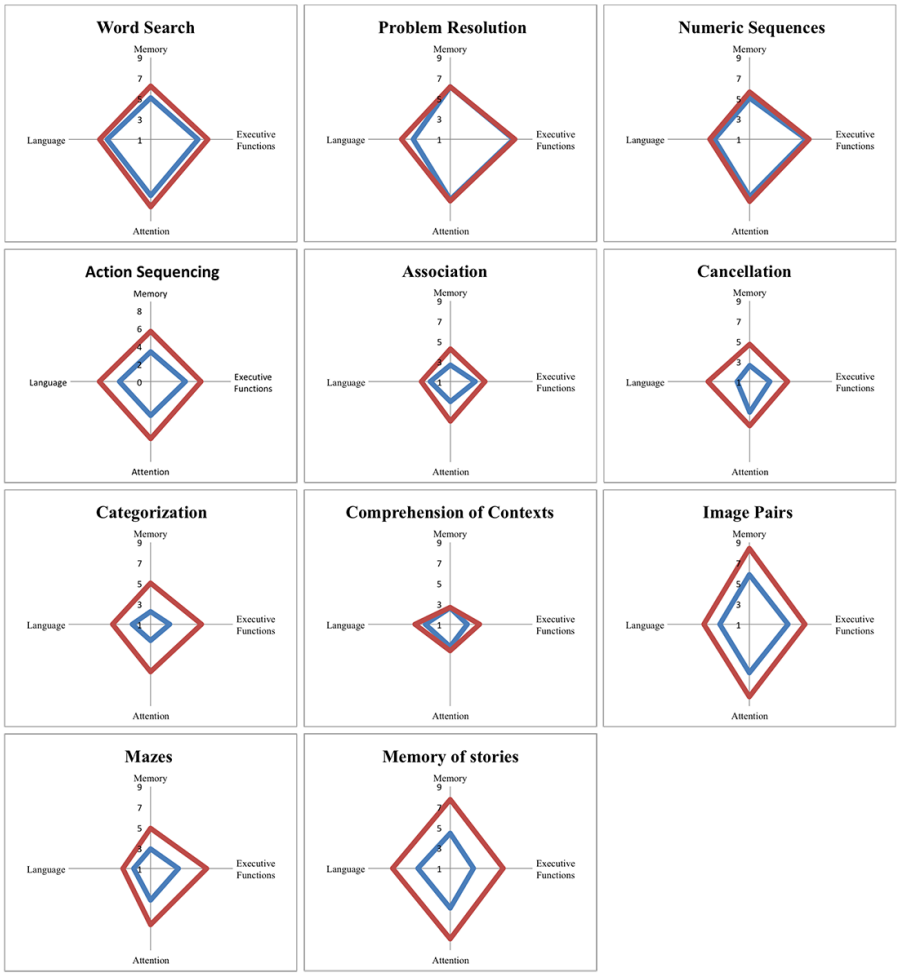


Fig. 2: Task adaptation profiles represented as radar plots

Following the operationalization of the 11 identified tasks and the meticulous identification of their inherent parameters, the task permutation process unfolded. The objective was to explore the expansive parameter space associated with each task, aiming for a comprehensive understanding of their cognitive nuances. The complexity of this task permutation became evident, considering the minimum requirement of studying 134 potential combinations. Acknowledging the impracticality of exploring every possible permutation, a judicious approach was taken. Task parameters were selectively chosen and combined, adhering to what was both feasible for implementation and amenable to mathematical modeling. This approach led to the creation of a total of 67 task variants across the 11 identified tasks. This permutation process not only maxi-

mized the exploration of cognitive dimensions but also optimized resources to derive meaningful insights into the potential cognitive training variations [8].

In the pursuit of a comprehensive understanding of task variants, the research team leveraged advanced computational approaches to enhance the generalization of each task's profile. The goal was to model the impact of untested parameters and their combinations on the cognitive demands of the tasks.

To construct the task profiles, shown in Fig. 2, the team employed a comprehensive analysis, assessing both the minimum and maximum ratings for each task variant across distinct cognitive domains. This analysis was driven by the input of 20 rehabilitation experts who were not involved in the earlier stages of the project. These experts meticulously rated each of the 67 task variants on a Likert scale ranging from 1 to 10. The domains under evaluation encompassed attention, memory, language, executive functions, and overall difficulty.

Through this methodology, the researchers crafted a cognitive rehabilitation program precisely tailored to individual cognitive domains based on the specific profile of each patient. The computational approach effectively encapsulates the implicit experience and knowledge of rehabilitation experts in a quantitative manner. This not only enriches the understanding of cognitive demands but also provides the team with explicit models, empowering the creation of an adaptation engine capable of personalized cognitive training.

2.1.2 Guttmann Neuro Personal Trainer

The 'Guttmann Neuro Personal Trainer' (GNPT) is a tele-rehabilitation platform that has become an integral part of clinical routines, with a significant impact on neurological disorder treatment worldwide, as evidenced by its use by over 12,000 patients and deployment in more than 210 clinical facilities globally, as reported on their official website [11]. GNPT represents the second generation of the PREVIRNEC tele-rehabilitation platform [12], which initially offered cognitive rehabilitation services starting in 2008 [13].

The system was designed to include a web application to be used by the therapists, in order to configure and schedule sessions, as well as a client application that the patients use to perform the scheduled tasks. The results would then be sent to the server and the therapist could evaluate them at any given time. This means that there is a considerable reduction in costs associated with treatment [6, 13].

One of the most unique features of GNPT is its incorporation of a decision support system, the Intelligent Therapy Assistant (ITA) algorithm. This algorithm not only schedules rehabilitation sessions for patients but also classifies each patient's characteristics. By comparing these characteristics to identical cases and drawing upon the most successful experiences stored in the system, the ITA empowers therapists to design personalized plans. This personalized approach is based on a comprehensive understanding of the patient's unique profile, derived from the wealth of information available within the system.

Moreover, the ITA algorithm is designed to be adaptive and flexible. While it offers automated suggestions for personalized treatment plans, therapists retain the autonomy to modify these recommendations. This blend of automated decision support and therapist customization ensures a finely tuned and tailored rehabilitation experience for each patient.

In a study conducted by Solana et al. [13], the ITA exhibited an impartial approach to task selection, considering both therapist preferences and clinical effectiveness. The study showcased the ITA's ability to provide a diverse array of exercises, offering patients a wide range of options. Importantly, the efficiency gains facilitated by the ITA were substantial, significantly reducing the time therapists needed to invest in planning sessions. The mean time for manual planning, which typically takes around thirty minutes for ten sessions, was drastically reduced to approximately 5 minutes when utilizing the ITA. This time-saving aspect contributes to a considerable increase in the overall efficiency of the scheduling and configuration process.

2.2 Accessibility

While several studies [10, 14, 15] have shown that similar tools, as the ones described previously have benefits, there is also a need to consider the extra side effects caused by using these tools [10], as well as take into consideration as to how people with cognitive deficits understand and use them [16].

2.2.1 Perceivable information

It is a fact that people with disabilities perceive the world differently and have several needs and preferences. Those needs should also be reflected in the design of applications.

One of the strategies used to help users better perceive information is to have alternative content. The most common one is text alternatives to non-text content, as presented by the Web Accessibility Initiate [17]:

- Short equivalents for images, including icons, buttons, and graphics
- Description of data represented on charts, diagrams, and illustrations
- Brief descriptions of non-text content such as audio and video files
- Labels for form controls, input, and other user interface components

Rotondi [16] mentions that when using labels, for people with cognitive deficits is it better to use explicit labels rather than using short succinct labels. These text alternatives help convey the intended purpose of the non-text content, while providing information that is equivalent to the media that is being presented.

Furthermore, textual content is present in every application which is why is it also important to have alternatives to it. However not every bit of text requires an alternative form of context. For example, having video or audio recordings when there are extensive bits of text. This benefits people who are blind, vision impaired, have cognitive disabilities or people who have trouble reading, since it allows them to have the text being read to them [18].

2.2.2 Operability

Sometimes users use different peripherals to utilize and navigate through a website or application. Thus, applications should be designed to accept different types of input devices, for example, mouse, keyboards,on-screen keyboards or switch devices [17].

Additionally, we have to consider that some people require more time to complete a certain task, either it being reading a text, typing in a form or understanding instructions. This implies that

users should have enough time to complete said tasks. The following are Web Accessibility Initiative's examples of mechanisms that provide enough time [17]:

- Stop, extend, or adjust time limits, except where necessary
- Pause, stop, or hide moving, blinking, or scrolling content
- Postpone or suppress interruptions, except where necessary
- Re-authenticate when a session expires without losing data

2.3 Web Application Technologies

This section will be used to present some of the technologies that were considered to be used to develop the Task Generator 2.0, implying that the project here described would be a web application.

2.3.1 HTML: HyperText Markup Language

HyperText Markup Language or HTML, as it is mostly know, is the most basic building block of the Web. It defines the meaning and structure of web content.

The word "HyperText" refers to the fact that links are an essential part of web pages, as that is the connection between websites. HTML uses "markup" text, that is each element in a page is identified by a "tag".

We will be using the official documentation [19] as a reference point to briefly explain an HTML element

An HTML element is composed of 3 main parts: an opening tag, the content and the closing tag.

- The opening tag consists of the name of the element wrapped in opening and closing angle brackets. It states where the element starts.
- The content, as the name indicates, is the actual content of the element.
- The closing tag, is very similar to the opening tag. It consists of the name of the element wrapped in opening and closing angle brackets but it also includes a *forward slash* before the name of the element. This indicates where the end of the element is. Not including a closing tag, leads to errors and is a common mistake for beginners.

2.3.2 CSS: Cascading Style Sheets

Cascading Style Sheets or CSS is a stylesheet language used to describe how elements should be rendered. CSS can be used for very basic styling such as changing color and size of text or it can be used to create more complex layouts. CSS is one of the core languages of web development and is standardized across Web browsers according to W3C specifications [20, 21].

CSS code is called a ruleset. The first element of a rule is the selector, which indicates the element that is going to be stylised. Inside the curly brackets, there can be one or multiple declarations. A declaration defines one or more rules, specifying an element's properties. Each declaration is defined by a pair of property and its value.

2.3.3 JavaScript

JavaScript is one of the most well-know web development programming languages. As stated in the official documentation [22] "It is the third layer of the layer cake of standard web technologies" the other two being the ones mentioned before, HTML and CSS. Although is mostly associated with scripting for web pages, it is also widely used in many browser environments such as Node.js [23].

JavaScript allows someone to add interactivity to a website. With enough practice a user can create animations, games comprehensive database-driven apps, display timely content updates, interactive maps and much more [22, 24].

2.3.4 React

React, also known as React.js, is a powerful open-source JavaScript library developed initially by Facebook and currently maintained by Meta, with contributions from a global community of developers [25]. It is designed for the purpose of creating interactive and efficient user interfaces for web applications.

React stands out for its declarative and component-based architecture. In React, user interfaces are constructed by composing individual, self-contained building blocks known as 'components.' These components encapsulate their own state, logic, and rendering, making it easier to manage and maintain complex user interfaces.

One of React's key advantages is its efficient rendering model. Instead of updating the entire user interface when a change occurs, React uses a Virtual DOM (Document Object Model) to optimize updates and ensure minimal rendering. This results in improved performance and a smoother user experience.

React's flexibility, along with its vast ecosystem of libraries and tools, has made it a popular choice for developing web applications, from simple websites to large-scale, data-driven platforms. Its emphasis on component reusability, clear code structure, and a vibrant developer community have contributed to its widespread adoption in the world of web development [26, 27].

2.3.5 Material UI

Material UI is a popular open-source React component library that implements Google's Material Design principles, providing a seamless way to create visually appealing and highly functional user interfaces.

One of its primary strengths lies in its extensive collection of prebuilt components, offering a diverse range of UI elements that are ready for production without the need for extensive customization. This includes everything from buttons, forms, and navigation components to more complex items like tables and modals.

Material UI's components are known for their aesthetic appeal and user-friendly design, adhering closely to the principles of Material Design. This means you can easily create interfaces with a modern and polished look, promoting a cohesive and visually pleasing user experience.

One notable advantage of Material UI is its flexibility and ease of customization. While it provides a wealth of out-of-the-box components, it also empowers developers to tailor the library to

match their specific design needs. This allows for the creation of unique and brand-specific design systems on top of Material UI's components.

Furthermore, Material UI boasts a supportive and active community, which contributes to its continuous development and improvement. This makes it a reliable choice for building robust and stylish user interfaces within the React ecosystem [28].

2.4 Feature Driven Development

The Feature Driven Development (FDD) method is an agile development approach for developing systems. The FDD approach uses an incremental iterative development process to manage the software development based on the list of required features needed. The focus of this method is to deliver high quality outputs throughout the entire development process [29, 30].

FDD method consists of five sequential processes, in which the designing and building of the system is carried out in a incremental, iterative way, leading to the final product.

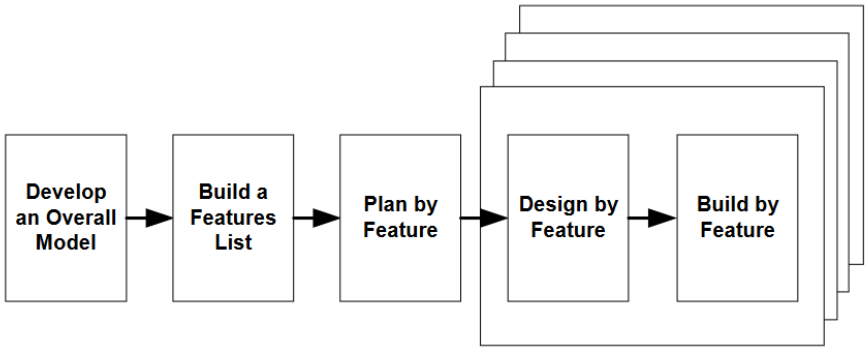


Fig. 3: FDD process

Figure 3 illustrates the five FDD processes. The processes are as follows:

- Develop an overall model: In this step the team members and experts define the overall project context and scope. It is presented a high-level description of the system. Many models are generated in this stage, which are then reviewed and the most appropriate ones are selected
- Build a features list: In this step, the overall model, from the previous step, and the existing requirement documentation is used to build the overall feature list. This list is then reviewed by users to check their validity.
- Plan by feature: A high-level plan is created from the previously approved list of features. The plan is ordered based on the priority and the dependency of the features. Major milestones and schedules may be determined for the set of features.
- Design by feature: This is an iterative step. Here the features are designed and reviewed to be approved.
- Build by feature: The last step of the FDD. It is also an iterative step. In this step the designs from the previous step are implemented, inspected an tested. After all the previous iterations are completed, the developed features are published to the main build and the iteration of designing and building a new feature begins and so on.

The selection of the Feature Driven Development (FDD) method for this project was based on its suitability for managing the development process in an incremental and feature-centric manner.

In the initial stages of planning, we recognized the need for an agile methodology that could effectively handle the evolving changes of the project. FDD aligns with this requirement by employing an iterative and structured approach to develop the system based on the identified features.

The step-by-step progression of FDD, from defining an overall model to incrementally designing and building features, appealed to our need for a well-organized and systematic development process. It provides a clear roadmap for feature prioritization, planning, design, and implementation.

2.5 Summary

In this chapter, we conducted an in-depth review of existing cognitive training tools, extracting valuable insights that profoundly informed the development of the latest iteration of the Task Generator. By delving into the literature, we not only gained a comprehensive understanding of established guidelines in software development but also identified key trends and advancements in web development technologies. This informed approach allowed us to integrate cutting-edge techniques and methodologies, ensuring that our proposed solution aligns with the latest industry standards and effectively addresses the evolving needs of cognitive training and rehabilitation.

3 Methodology

In this section, we will talk about some of the methods used in the development process of the Task Generator 2.0, as well as explain the reasons behind why certain methods and technologies were chosen.

3.1 Understanding the NeuroRehabLab Task Generator

First and foremost, before starting any research or developing a prototype, we had to develop and understanding of how does the Task Generator work. For starters we had to access the TG and then we explored it from the user point of view. This allowed us to have a basic knowledge as to how a user might interact with it, while also exploring the platform.

One of features we first noticed was that it generated a PDF file with the generated task or training program, depending on which tab we selected. When choosing the "Profile" tab, the generated PDF had a full training program with a maximum of 11 tasks, while the other tabs only produced their respective task. For example, the "Cancellation" tab (Figure 4) produces 1 page PDF file with a cancellation task. Noticing this, we immediately hypothesized the idea of making it interactive, by replacing the PDF generation with interactive content, such as buttons or text boxes.

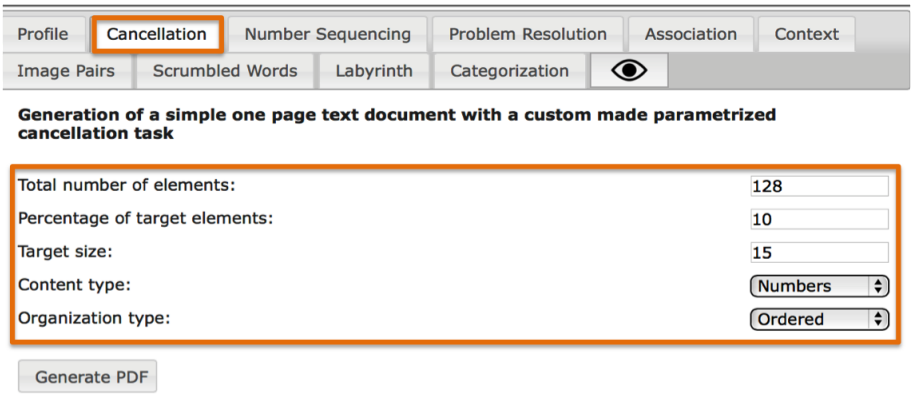


Fig. 4: Example of an individual task (cancellation task) and its parameters

Another feature we noticed was that each generated file had a chart with the following attributes: "Attention", "Memory", "Ex. func.", "Language", "Difficulty", as shown in Figure 5. After going through the literature [7, 8, 15] we realized that these attributes referred to the cognitive domains and the values associated with each attribute were the cognitive demands on each one of them. In other words, each task has a certain demand on a certain cognitive domain. For example, a "Word Search" task has a high demand on "Memory", "Language", "Attention" and "Executive Functions" while a "Maze" task has a low demand for Language but high demand for "Attention" and "Executive Functions". For more information, please refer to the article [8].

After developing a basic understanding of how the platform worked, it was time to further investigate the TG and learn how its' algorithms work and how it generated the PDF file. By looking into the TG's code it was clear as to how it generated each task. Each task has its own function that takes in the parameters from the form shown to the user and generates the task, using a

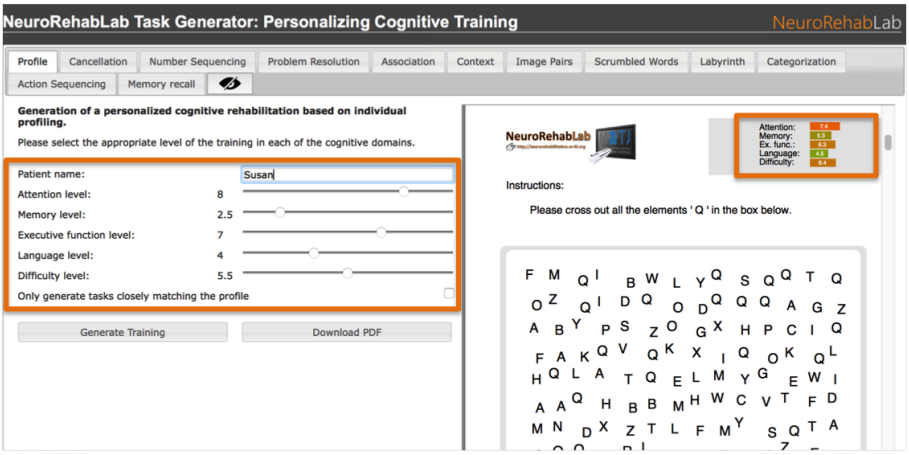


Fig. 5: A cognitive training program parameters

series of randomizing functions and mathematical equations, as well as, creating the PDF file. Every task has its own model that shows how much it affects every cognitive domain, which is also printed in the PDF file as mentioned before.

3.2 Requirements

At the inception of the NeuroRehabLab Task Generator 2.0 project, it was imperative to establish a clear set of requirements and objectives. These served as the guiding principles that shaped the development process and steered the project towards its intended outcomes. The following sections outline the core project requirements and objectives.

In the initial project meetings with the project and team members, several critical aspects of the original Task Generator were identified, underscoring their significance in the development of the new version. These discussions served as the foundation for outlining the system requirements. Through subsequent meetings and ongoing collaboration, we refined and adapted the initial requirements, accommodating changes as needed and introducing new ones where necessary.

3.2.1 Functional requirements

Functional requirements are statements that describe what the system should do or provide for the users and stakeholders [31]. These are:

1. The system must generate a variety of cognitive tasks based on defined algorithms and models.
2. The system must abstract the task generation logic from task presentation
3. The application must facilitate user interaction with the generated tasks.
4. Users should be able to submit their task data for analysis and storage.
5. The system should seamlessly integrate multiple tasks to create a comprehensive cognitive training program.

3.2.2 Non Functional requirements

Non-functional requirements, often referred to as system qualities or quality attributes, specify the characteristics and constraints that define how a system should behave rather than what it should do.

1. The application must feature a user-friendly interface that is easy to navigate and understand, promoting usability.
2. The system should be accessible to a broad range of users, considering different abilities and needs.
3. The application should be compatible with various devices, including tablets and computers
4. The application should provide the capability to add new stimuli.
5. The system should support translations in multiple languages.
6. The system must operate without critical errors, ensuring reliability and stability.

3.3 Development Process

After understanding the TG's functionality, we adopted an agile software development methodology, as described in Section 2.4. This agile approach favors incremental development. Each task was developed individually, enabling us to test and validate them before proceeding to the next. This approach allowed for swift adjustments if needed without affecting other tasks, reducing error propagation.

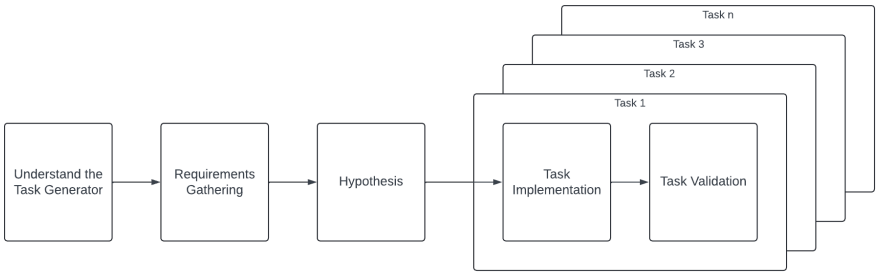


Fig. 6: Overview of the development process

To achieve the goal of abstracting the task model from the view (as discussed in Section 1.2), we opted for a client-server architecture. This architectural choice yielded two primary components: the backend (server) and the frontend (client). The server handles computational tasks, such as task generation, while the client presents the generated content.

Given the intended compatibility of TG 2.0 with tablet devices and personal computers, web development emerged as a suitable option. This choice, detailed in Section 2.3, provides a versatile solution.

At first, we developed a prototype application to test the viability and feasibility of these technologies. This prototype is described below, in section 3.3.1. While developing the React application, we simultaneously began work on a prototype using React Native. However, due to its

similarities with the React application and the substantial effort required to make it a viable alternative, we ultimately left this prototype unfinished.

During the initial stages of development, our primary focus was on ensuring the proper functionality of both components and establishing secure communication between them. This phase also involved making necessary modifications to the original code to facilitate data exchange between the components. At that time, we were actively testing various web technologies and experimenting with our hypothesis of implementing a client-server approach. However, with the successful construction and operation of the React app, our prototyping phase transitioned into a more refined development stage.

3.3.1 Proof of concept

In this section, we delve into the implementation of a proof of concept for the NeuroRehabLab Task Generator 2.0. A proof of concept serves as a preliminary demonstration, showcasing the feasibility and core functionalities of the proposed system. This initial implementation provides valuable insights into the practicality of the project, guiding further development and refinement.

A prototype was developed using the basic web development programming languages, that is HTML, CSS and JavaScript. This proof of concept application is only composed of one of the tasks of the original Task Generator, the Cancellation Task.

As mentioned earlier there are two main components: the client (front end) and the server (back end). The client shows a form similar to the one of the original TG, although not heavily stylised. After submitting the form data, a POST request is sent to the server. The server then calls upon a function that generates the task, using the form data. Finally, the server returns an object with several properties, as shown, by Figure 7 and listed below:

Fig. 7: Object returned from the server

- model: the model is an object that has the information related to the cognitive domains that are affected, as well as their values
- instructions: the instructions of the task
- taskArray: this is an array containing all the elements that make up the task. These can be either letters, numbers or symbols.
- vectorTargets: this is also an array containing the positions of the targets. Note that this array may not have the same number of elements as the taskArray.

The client then uses this object sent from the server and creates a "grid" like structure, similar to the one in the generated PDF file from the original Task Generator.

Each element inside this grid can be interacted with. Upon clicking on an element, a red circle is used to highlight the selected element. This way we can simulate how a user would highlight the same element on paper. Only one element could be highlighted, but it is intended to have multiple elements being selected.

At the time of development of the first prototype application, a new prototype was being tested using React, and other technologies, such as Flutter and NextJS, were being evaluated for viability and feasibility.

The primary focus of the prototype described in this section, was to test our hypothesis of separating responsibilities between frontend and backend, establishing communication between client and server, while maintaining the core functionality intact.

3.3.2 React application

Once we were able to determine the viability of using web technologies, we moved on to developing the React application, that is described in this paper. We used the already built JavaScript prototype as a base for building the rest of the application.

The React application, at a basic level, functions similarly to the prototype described above. For each task, there is a form that a user has to fill. After the form is submitted, the data is sent to the server (backend). The server will generate the respective task and send the generated task data back to the frontend. The frontend will then display the task, with interactive elements, such as buttons, radio buttons or select fields. After completing the task, the user can then submit the task.

The development of the application was structured around the step-by-step development of each task. Initially, we migrated the task generation algorithm from the original Task Generator and adapted it to fit the new project's architecture. This implementation took place on the backend, where we defined all the task generation logic. Each algorithm underwent rigorous testing to ensure accurate task generation. To validate the correctness of the generated tasks, we compared them to the 'models' generated by the original Task Generator. The *model*, that can be seen in Figure 7, for each task is based on the "*Cognitive Profile of Tasks*" as outlined by Faria et al. [8]. This profile comprises of five parameters, four of which specify the cognitive load on different domains: language, attention, executive functions, and memory, while the fifth parameter gauges the overall task difficulty. Other task specifics were factored in for comparison, namely the stimuli. Due to the nature of randomized generation, we had to make sure that the stimuli generated were of the same type as those used by the original Task Generator. Once we confirmed the similarities between our generated tasks and the reference model, we shifted our focus to user interaction with the data. In other words, we began developing the frontend.

The frontend incorporated various elements such as buttons, checkboxes, text inputs, and select inputs in order for users to interact with the data. Each task is divided into two components, the '*TaskForm*' and the '*Task*'. The former is responsible for feeding the parameters, provided by the form, to the backend for it to generate the task. The latter, displays the task generated by the backend, which can then be interacted with, through the use of interactive elements.

3.4 Architecture

The proposed architecture for this project follows a client-server model, where the backend and frontend work synergistically to deliver dynamic and interactive cognitive tasks.

The backend is responsible for generating cognitive tasks, processing user interactions, managing data, and responding to frontend requests. It encapsulates the logic behind each task's generation, ensuring that tasks are diverse, engaging, and tailored to stimulate specific cognitive func-

tions. Communication between the frontend and the backend is facilitated through HTTP requests, with 'axios' being employed to manage this interaction.

The frontend is responsible for presenting the interactive cognitive tasks, capturing user interactions, and transmitting data between the user's device and the backend server.

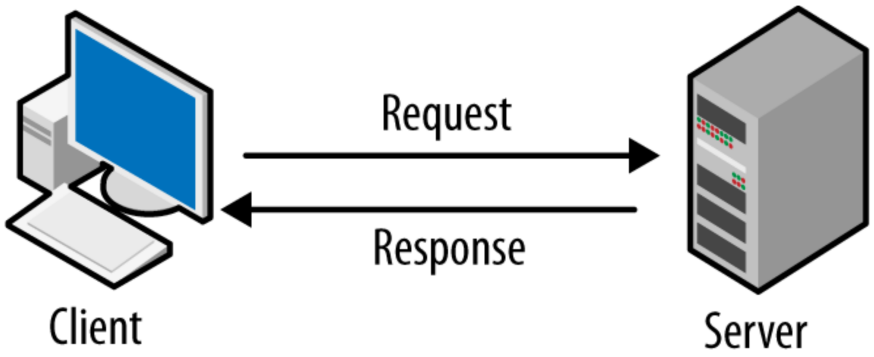


Fig. 8: Example of a simple client-server architecture

The architecture is designed with scalability and extensibility in mind. It provides a solid foundation for adding new cognitive tasks, enhancing existing ones, and adapting to evolving research needs. The modular nature of the architecture allows for the seamless integration of additional features and technologies in the future.

4 Implementation

Having outlined the conceptual framework and the development process, we now turn our attention to the practical implementation of the React application described in this thesis, where will meticulously examin each task, presenting a comprehensive understanding of their individual implementations. It is paramount to underscore that each task's implementation was not only a technical endeavor but was rigorously validated..

As the development process progressed, a significant usability enhancement was introduced. After submitting the task form, it is discreetly hidden, promoting an enhanced user experience by allowing users to concentrate solely on the task at hand. Moreover, we implemented two essential functionalities: the ability to revisit and edit the form without the need to start anew, and the option to return to the task without submitting a fresh form, preserving the data from the previous submission. These user-centric features were thoughtfully integrated into each task, ensuring a seamless and intuitive interaction.

4.1 Shared modifications

In the pursuit of a unified and streamlined implementation approach across all tasks, we present a dedicated subsection outlining the shared modifications and enhancements that transcend individual task implementations.

For improved readability and consistency, a key adjustment made across all tasks involved the transformation of the model function. Instead of returning an array, it now returns an object, enhancing the clarity of task data.

Additionally, to foster a unified structure and facilitate streamlined handling of task data, the '*taskObject*' function was introduced. While each task maintains its own unique algorithmic implementation, the resulting object, '*taskJSON*', shares a common structure. This structure includes a 'model' property, encompassing the object returned from the model function, task-specific 'instructions,' and a variable set unique to each task, which may vary from one task to another. These modifications promote cohesion in data representation while accommodating task-specific nuances.

To facilitate the submission of task data upon completion, we implemented a dedicated function utilizing 'axios' to perform a POST request to the 'store-data' route on the server. The server, upon receiving these requests, captures and stores the incoming data in a JSON file. To ensure uniqueness, 'nanoID' is employed to generate a unique identifier, which serves as the filename. These JSON files are systematically organized within folders corresponding to their respective task type.

4.2 Cancellation Task

The initial task we undertook for implementation was the Cancellation task, which marked the starting point of our development journey. We approached this task with meticulous consideration, beginning with the backend. The algorithm, sourced from the original Task Generator, served as our foundation, but significant adjustments were necessary.

First, we initiated the implementation process by strategically removing code pertaining to PDF file generation. Subsequently, we proceeded to make essential adjustments while ensuring that

the core functionality remained intact. For instance, rather than directly positioning elements within the PDF file, we introduced a novel array called '*taskArray*', encompassing all task elements, including both *distractors* and *targets*. Another noteworthy variable to highlight is the '*vectorTargets*' array, which exclusively comprises positions containing targets. These variables are returned as an object.

The implementation of the '*taskObject*' function, combining the model and task-specific porperties, resulted in the '*taskJSON*' object having the following properties:

- model: The object provided by the model function
- instructions: Task-specific instructions.
- target: The target element to be selected in the task.
- taskArray: The array of elements of the task to be transformed into a grid like structure.
- vectorTargets: The array of target positions.
- numbers: A variable that determines whether symbols, letters, or numbers are used.
- size: A variable that sets the size of displayed elements.

After defining the '*taskObject*' function and its properties, it plays a central role in the server-side operations. When a request originates from the frontend, transmitted using 'axios,' the server intercepts it, invoking the '*taskObject*' function to generate the task according to the provided parameters. The server then forwards this dynamically created task back to the frontend for user interaction.

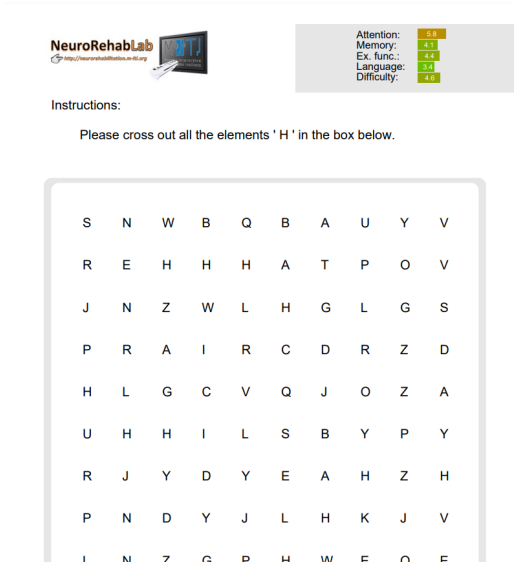
With a clear understanding of how the server leverages the 'taskObject' function to generate tasks dynamically, our focus shifted to the frontend implementation. Beginning with the '*CancellationTaskForm*' component, our approach prioritized user familiarity by retaining the structure of the original form while infusing it with a fresh visual identity. We achieved this by incorporating Material UI elements like 'Textfield' and 'Select.

With the form completed, our next step was to manage the submission of data to the backend. To handle form submission, we employed 'axios' to send a POST request to the backend. Upon receiving a response, the data is stored in a state variable, specifically 'cancelTask.' This state variable is then passed as a prop to the '*CancellationTask*' component.

After successfully managing form submission and receiving data from the backend, our focus shifted to providing users with an interactive interface for completing the task. In our approach, we aimed to maintain consistency with the original design, making efforts to recreate the same user experience, see Figure 9. To achieve this objective, we employed a table element, strategically chosen to replicate the grid-like design of the original. Within each table cell, an element from the '*taskArray*' is thoughtfully presented for user interaction. Furthermore, a checkbox input element was employed, thoughtfully styled to resemble a button. This allowed users to seamlessly select or deselect elements.

When a user selects an element within the grid, a corresponding function is triggered, capturing the position of that selected element. These positions are then stored in a state variable named '*selectedPosition*', which accumulates all the positions of the selected elements. Upon task com-

pletion and submission via the designated submit button, the data is sent to the backend to be stored in its' appropriate folder.



(a) Cancellation Task generated using the original TG

(b) Cancellation Task generated using the TG2.0

Fig. 9: Comparison of Cancellation Tasks between TG 1.0 (left) and 2.0 (right)

4.3 Number Sequencing Task

The implementation of the Number Sequencing task followed a parallel workflow. Initially, we sourced the algorithm from the original Task Generator and subsequently removed code associated with PDF file creation. While ensuring that the core functionality remained unaltered, we made adjustments, primarily by storing data in variables rather than directly inserting it into the PDF. One of these variables is the '*sequenceArray*,' aptly named for its role as an array of sequences. These sequences are represented as arrays themselves, housing the numbers of a numerical sequence.

The model function underwent a parallel modification, mirroring the changes implemented in the previous task, where the returned array was transformed into an object.

Likewise, the '*taskObject*' function maintains consistency by combining the model and the task, resulting in an object featuring the following properties:

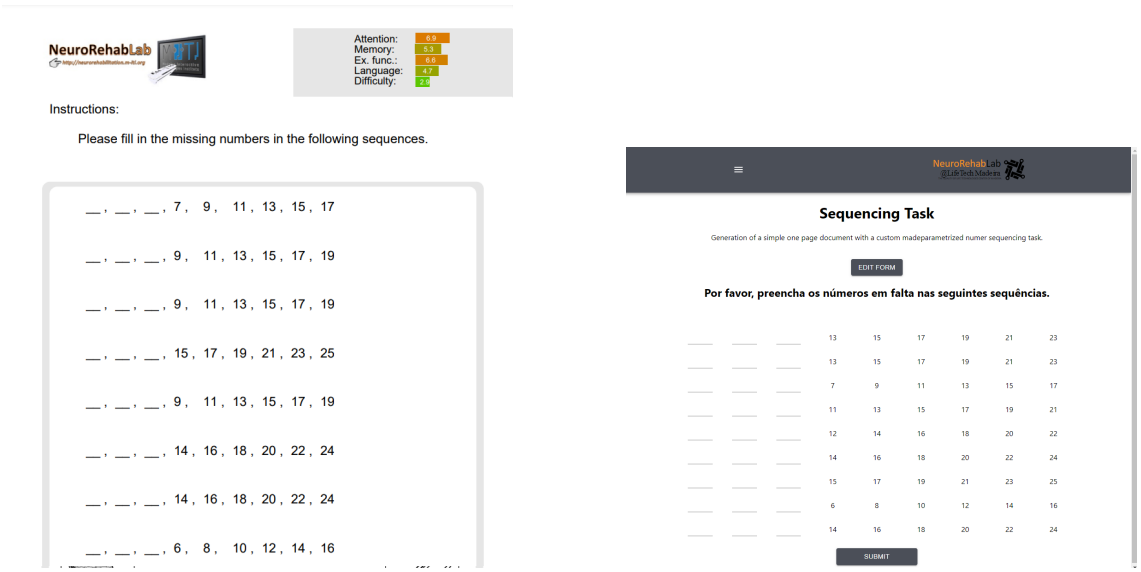
- model: The object provided by the model function.
- instructions: Task-specific instructions.
- sequence: The '*sequenceArray*' shortened to sequence, an array of sequences

With the algorithm successfully adjusted, our focus shifted to the server-side implementation. To accommodate the frontend's request for task generation, a new route had to be created to intercept the request and return the generated data back to the frontend.

The transition from server configuration to frontend implementation started with the development of the '*SequencingTaskForm*' component. To modernize its appearance while retaining a

touch of the original design, we leveraged Material UI elements. Handling data submission involved the use of 'axios' to transmit a POST request to the server, with the returned data being stored in a state variable named '*sequenceTask*.' This variable, in turn, serves as a prop for the '*SequenceTask*' component.

With the newly generated task, the '*SequenceTask*' component can now be assembled. Using Material UI's 'Grid' we managed to create a grid-like structure, that not only resembles the original but also has a responsive design. Using the provided '*sequence*' array, we mapped the grid to display the elements of each sequence. In cases where the element is not a number, we implemented Material UI's 'TextField' component, allowing for user interaction.



(a) Number Sequencing Task generated using the original TG

(b) Number Sequencing Task generated using the TG2.0

Fig. 10: Comparison of Number Sequencing Tasks between TG 1.0 (left) and 2.0 (right)

Once a user has changed the value of one of the 'TextField' elements, a function is triggered to update the state variable '*inputValues*', which is structured as a 2 dimensional array, based on the row and column indexes of the modified 'TextField'. As an example, if a user inputs a value into the fourth 'TextField' of the third row, it updates the 'inputValues[2][3]' with the introduced value. Whenever a user has finished executing the task, the '*inputValues*' can be submitted through the submit button.

4.4 Problem Resolution Task

The implementation of the Problem Resolution task followed a workflow akin to the tasks mentioned previously. The initial stage of implementation involved sourcing the original algorithm and subsequently removing sections related to PDF file creation. In its place, we introduced new variables to capture data, paralleling the prior direct placement within a PDF while preserving functionality.

One crucial variable introduced was '*sequenceArray*', an array designed to accommodate two distinct types of data depending on the problem to be generated.For explicit problems, '*sequenceAr-*

ray' stores multiple arrays, each containing elements that compose mathematical equations, complete with basic operators. In the case of implicit problems, it stores multiple strings, each describing a mathematical problem in a textual format.

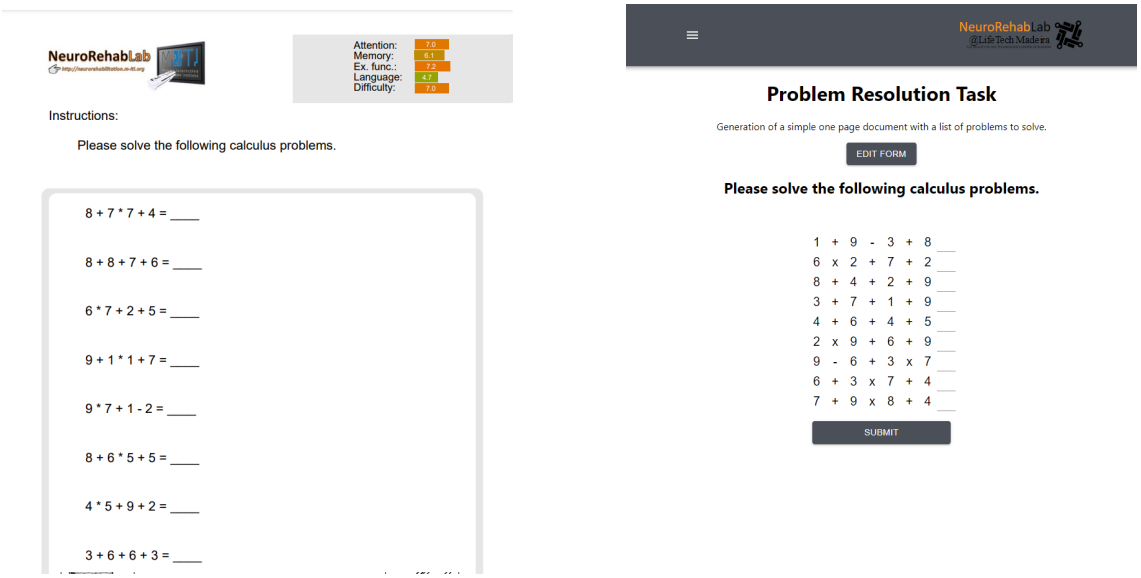
The '*taskObject*' function when combining both the model function and the task generation function, resulted in the 'taskJSON' object with properties as listed below:

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- sequenceArray: The '*sequenceArray*', an array that stores mathematical equation either in the form of an array of elements or in textual format
- explicit: A variable that determines whether it is an explicit problem or implicit

Subsequently, we established a new server route to intercept requests originating from the frontend for the generation of the task and to return the corresponding data for display.

With the completion of the task generation logic, we transitioned to the frontend, commencing with the '*ProblemTaskForm*' component. To enhance the user interface, we once again revamped the form fields using Material UI elements. Data submission was managed via 'axios,' employing a POST request to communicate with the server. The resulting data was stored in a state variable, '*problemTask*,' subsequently passed as a prop to the '*ProblemTask*' component.

Within the '*ProblemTask*' component, we leveraged the data provided through props to construct the user interface, recognizing that it could accommodate two distinct types of problems. For explicit problems, we utilized 'Grid' elements to organize the display. The numbers and operands contained in '*sequenceArray*' were center-aligned to mimic an equation format. Additionally, a 'Textfield' was incorporated to enable users to provide answers, as can be observed in Figure 11b. In the case of implicit problems, we presented the textual content to users while also including a 'Textfield' for inputting answers.



(a) Problem Resolution Task generated using the original TG

(b) Problem Resolution Task generated using the TG2.0

Fig. 11: Comparison of Problem Resolution Tasks between TG 1.0 (left) and 2.0 (right)

Whenever an input field's value changes, these changes are promptly stored in the state variable '*inputValues*.' This variable, structured as an array, correlates each item's position (index) with its corresponding equation.

4.5 Association Task

Following the same steps as previously done, the original code was sourced and adjusted. The initial steps involved the removal of code pertaining to PDF file creation and the positioning of elements within the PDF.

To replace these removed sections, we introduced new variables designed to store the association pairs. '*leftObjects*' and '*rightObjects*' are both arrays that house objects. Each object consists of an 'id' property, generated using npm's 'nanoid' library, and an 'image' property containing the image. The '*pairings*' array serves as the consolidation of elements sharing the same 'id' from both arrays. This consolidation results in the creation of objects, each featuring properties named 'left' and 'right,' respectively associated with their respective arrays. Both '*leftObjects*' and '*rightObjects*' are both randomly rearranged before the data is returned.

Implementing the '*taskObject*' function, combining the model function and task generation algorithm, resulted in the 'taskJSON' having the floowing properties:

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- leftObjects: An array of objects to be displayed on the left side.
- rightObjects: An array of objects to be displayed on the right side.
- pairings: An array of objects, each representing a pair of objects, with one object from each of the previous two arrays.

– note: A note referencing images sourced from the IPNP database, visible only when clipart images are utilized.

A new server route was created to intercept the request from the fontend to generate the task and return the task data.

With the backend implementation completed, the frontend implementation swiftly followed. Beginning with the '*AssociationTaskForm*', the original was recreated using Material UI's elements to achieve a contemporary appearance.

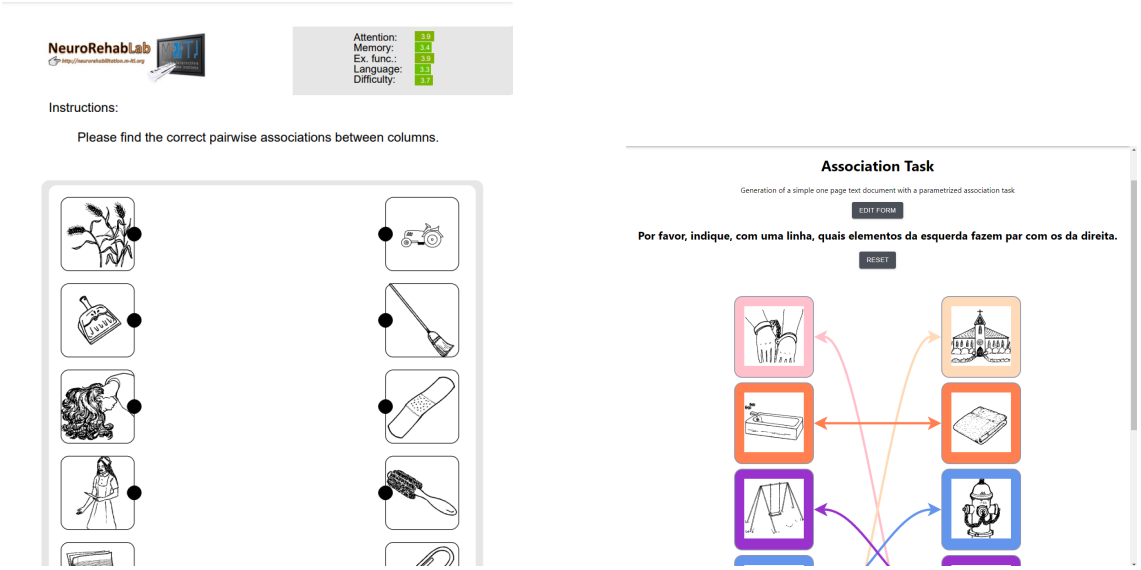
To manage data flow from the frontend to the backend, a POST request using 'axios' was used. The data returned from the backend is saved on '*associationTask*' state variable, which subsequently serves as a prop for the '*AssociationTask*' component.

Leveraging the generated data, we constructed the user interface, harnessing the power of Material UI's 'Grid' to showcase the '*leftObjects*' and '*rightObjects*' on their respective sides. Additionally, we introduced a custom-built interactive component - the '*Card*' component - designed to display images within frames and to facilitate color matching for pairs.

To ensure a structured user experience, rules were established to enforce specific behavior. Users are not permitted to select elements on the same side, and the task mandates that selections begin on the left side.

When a user interacts with a 'Card' on the left side by clicking it, the 'Card' is highlighted with a grey color, and simultaneously, the 'choiceLeft' state variable is updated. Users can then select a 'Card' on the right side to form a pair, leading to the update of the 'choiceRight' state variable.

Upon creating a pair, the pair's details are stored within the 'associatedPairs' state variable. This storage maintains a structure similar to that of '*pairings*,' with an additional property, 'color.' The color is selected from a predefined list and assigned to the pair. This color plays a crucial role in displaying an arrow using 'react-xarrows'.



(a) Association Task generated using the original TG

(b) Association Task generated using the TG2.0

Fig. 12: Comparison of Association Tasks between TG 1.0 (left) and 2.0 (right)

Using 'react-xarrows,' we map through the 'associatedPairs' array to draw arrows, utilizing the pair's IDs. These arrows originate on the left side and conclude on the right side.

We also introduced a 'Reset' button, providing users with the option to clear their selections and initiate the task anew.

4.6 Context Task

Adhering to the same systematic approach, we obtained the algorithm code from the original source and proceeded with necessary adjustments. Code segments related to PDF file creation were eliminated, and textual content was relocated to a distinct file within a designated 'translations' folder, facilitating translation into multiple languages.

Further enhancements were implemented to ensure seamless generation of randomized content, in line with the revised structure involving object return. As part of these modifications, we transitioned from string concatenation to storing strings within an 'sentences' array.

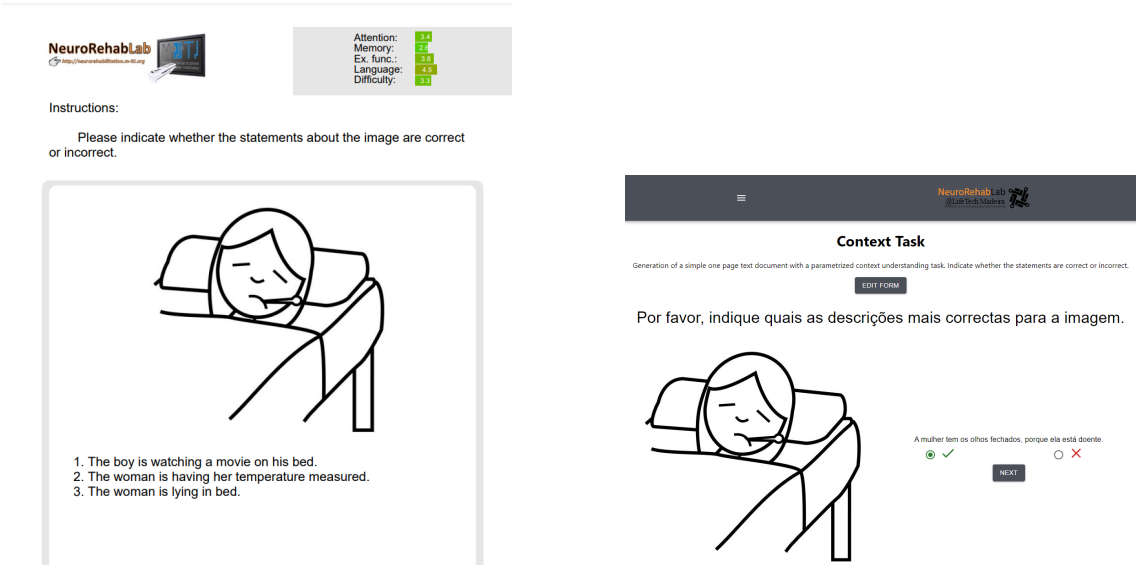
The '*taskObject*' function was implemented, combining the model function and the task generation algorithm, resulting the the following 'taskJSON' object.

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- image: The image that is displayed for interpretation.
- sentences: An array of sentences to be evaluated as correct or incorrect
- correct: An array of truthful statements
- incorrect: An array of false statements

To intercept the request to generate the task originating from the frontend, a new route was created in the server, to them return the task data.

With the backend development successfully completed, our focus naturally shifted towards the frontend. We designed the '*ContextTaskForm*' using Material UI to give it a modernized appearance. To manage form submissions, we employed 'axios' to facilitate communication with the backend through a POST request. Upon receiving the server's response, which is then provided to the '*ContextTask*' component, via the props.

The 'ContextTask' component utilizes Material UI elements such as 'Grid,' 'Typography,' and 'Radio' inputs. Within this component, the image received from the backend is displayed on the left side, while sentences are presented on the right side. Only one sentence is visible at a time in this interface, as can be see in Figure 13.



(a) Context Task generated using the original TG

(b) Context Task generated using the TG2.0

Fig. 13: Comparison of Context Tasks between TG 1.0 (left) and 2.0 (right)

As users interact with the 'Radio' buttons to indicate whether a statement is correct or incorrect, a function is triggered. Initially, the 'Next' button remains hidden, becoming visible only after the user selects an answer for the first time. Upon clicking 'Next,' not only does it allow users to proceed and submit their responses, but it also reveals the next sentence in the series. This interaction triggers the function, updating the state variable '*selectedOption*,' which temporarily holds the user's choice until they click 'Next.' Subsequently, the 'formData' variable is updated. 'formData' is an array of objects, with each object containing a 'sentence' property representing the displayed sentence and a 'selection' property indicating the user's selected answer.

Once all sentences have been shown, the 'Next' button is dynamically replaced with a 'Submit' button. This transition signals to the user that they have completed all sentence evaluations and can proceed to submit their final responses.

4.7 Categorization Task

Following our consistent approach, we retrieved and adapted the source code to align with the project's specific requirements. Notably, we undertook substantial revisions, particularly within the algorithm itself.

A significant part of these revisions involved the removal of code segments associated with PDF file creation. However, the most substantial changes were introduced directly into the core of the algorithm

Within this context, the algorithm responsible for extracting base64 encoded images underwent several key changes. Notable additions included the introduction of new variables, such as 'category,' which now holds the category information for selected images. Additionally, we implemented a function designed to return objects comprising both the 'category' property and the 'dataUrl' property, which stores the image data.

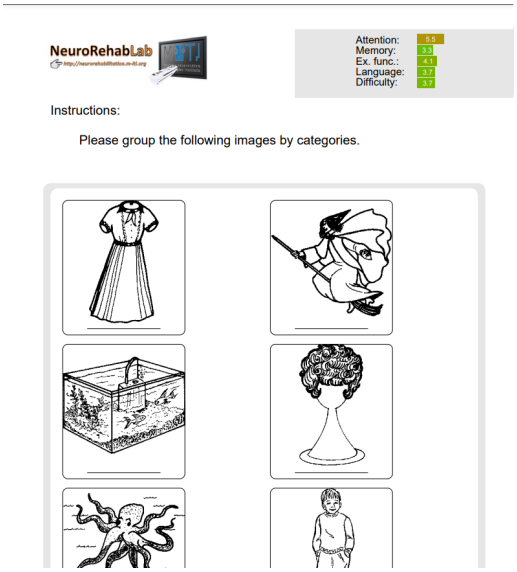
Following this, the algorithm proceeds to extract these objects while simultaneously gathering all available categories for use in the select field.

Subsequently, we implemented the '*taskObject*' function, which seamlessly merges both the model and task generation components. This integration results in the 'taskJSON' object, structured as follows:

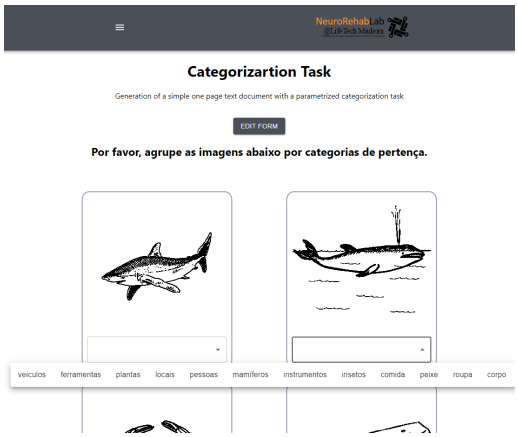
- model: The object provided by the model function.
- instructions: Task-specific instructions.
- taskArray: An array of objects each containing the 'category' and 'dataUrl' properties
- note: A note referencing images sourced from the IPNP database, visible only when clipart images are utilized.
- allCategoriesList: An array containing all the categories
- categories: A variable referencing the number of categories
- elements: A variable referencing the number of elements per category

Following the successful implementation of the backend, our attention turned to the frontend development. The first step in this phase was the redesign of the '*CategorizationTaskForm*' using Material UI components for the form elements. To handle form submissions, we utilized 'axios' to establish seamless communication with the backend via a POST request. Once the backend processed the request and provided the necessary data, it was stored in the '*categorizationTask*' state variable, which we then passed as a prop to the '*CategorizationTask*' component.

The '*CategorizationTask*' component was designed with a grid-like appearance, reminiscent of the original task format. To determine the number of columns in this grid, we leveraged the 'categories' and 'elements' properties from the 'taskJSON' object. The smaller value between these two properties determined the number of columns in the grid. Additionally, we utilized Material UI's 'MenuItem' component to display all available categories within a 'Select' input, with the an image above it, see Figure 14.



(a) Categorization Task generated using the original TG



(b) Categorization Task generated using the TG2.0

Fig. 14: Comparison of Categorization Tasks between TG 1.0 (left) and 2.0 (right)

The user is required to select the category that corresponds to the displayed image, which triggers a function responsible for updating the 'categorySelection' state variable. This variable maintains the same structure as the 'taskArray' variable provided by the server, with the only distinction being that the 'category' property starts as empty. When updating the 'categorySelection' variable, the function utilizes the index to identify the specific object to be modified, and the selected category is assigned to that object.

4.8 Action Sequencing Task

Following our established methodology, we retrieved the original code and performed necessary adjustments. As we did with previous tasks, we removed all code associated with PDF file creation. However, we encountered significant challenges in the previous data organization.

In the earlier version, implicit actions occupied a fixed set of 6 positions within the implicit array, while explicit actions took up 7 positions. This rigid structure made it challenging to make adjustments and prone to errors, especially when dealing with actions that required a varying number of steps.

To address these challenges and enhance flexibility, we undertook a major restructuring of the data organization. We transitioned from a fixed array structure to arrays of objects, with each object representing an individual action. This restructuring was implemented to enhance readability, facilitate future maintenance, and provide the flexibility needed for actions with varying numbers of steps.

For 'implicit' actions, each action now solely comprises the 'steps' property, which organizes the steps based on their indexes. Conversely, 'explicit' actions include the 'action' property to identify the specific action and the previously mentioned 'steps' property. These changes have not

only improved the maintainability of the code but have also mitigated the challenges associated with fixed-length arrays.

With these structural changes, it became imperative to adapt how the algorithm selects data. Subsequently, we made some minor yet crucial modifications. Previously, the algorithm had to meticulously ensure it selected the correct number of elements from a predetermined array. However, with the transition to arrays of objects, the algorithm's task simplified significantly. It now merely needs to select a single object from the array and extract the necessary information.

These adjustments allowed us to preserve the core functionality of the algorithm while significantly streamlining the data selection process. The algorithm's logic remained intact, ensuring the seamless generation of tasks.

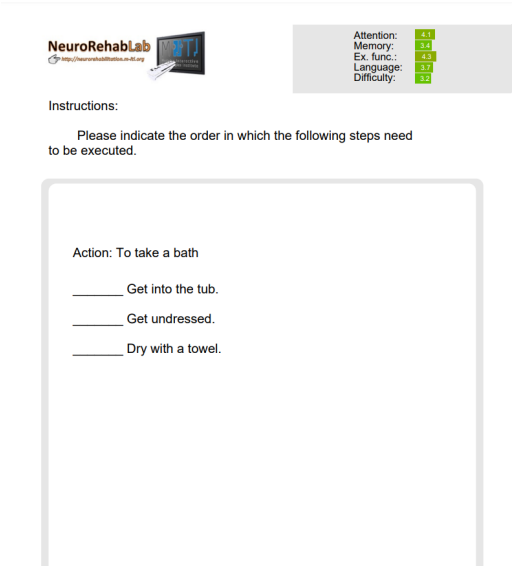
The implementation of the taskObject function played a pivotal role in shaping the structure of taskJSON, the object responsible for storing task data. As mentioned earlier, this function harmonizes the model function's output with task-specific properties. The result is a unified structure as follows:

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- action: The action represented, only specified if an explicit action
- sentences: An array of the steps selected

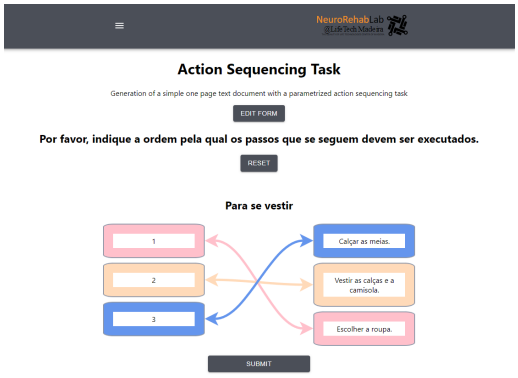
To facilitate the frontend request for task generation, we established a new route to intercept these requests. This route orchestrates the task generation process by invoking the 'taskObject' function and subsequently relays the generated data back to the frontend.

Having successfully implemented the backend, our next focus shifted to the frontend. We initiated this phase with the redesign of the '*ActionSequencingTaskForm*' using Material UI components to revamp the form's aesthetics and ensure a modern look. For efficient data exchange between the client and server, we harnessed 'axios' to execute a POST request. Following server response, the received data is stored in the state variable '*actionSeqTask*', subsequently passed to the 'ActionSequencingTask' component via props.

For the user interface, we adopted a similar approach to the one used in the 'AssociationTask.' We employed Material UI's 'Grid' to present the action 'steps' on the right side, while numbers were displayed on the left side, each element encapsulated within a custom 'Card' component, as can be seen in Figure 15b. Much of the logic utilized in the 'AssociationTask' was carried over to this component.



(a) Action Sequencing Task generated using the original TG



(b) Action Sequencing Task generated using the TG2.0

Fig. 15: Comparison of Action Sequencing Tasks between TG 1.0 (left) and 2.0 (right)

To ensure a structured user experience, we established specific rules. Users are restricted from selecting elements on the same side, and the task mandates that selections commence on the left side.

When a user interacts with a 'Card' on the left side, clicking it highlights the 'Card' in grey, simultaneously updating the 'choiceLeft' state variable. Users can then choose a 'Card' on the right side to form a pair, leading to an update of the 'choiceRight' state variable.

Upon forming a pair, the pair's details are stored in the 'associatedPairs' state variable. This variable stores data as an array of objects, each object having a 'left' and 'right', each representing the respective elements as they are displayed on screen. An additional 'color' property was also added. The color is selected from a predefined list and assigned to the pair. This color plays a crucial role in displaying an arrow using 'react-xarrows.'

Employing 'react-xarrows,' we map through the 'associatedPairs' array to draw arrows, utilizing the pair's IDs. These arrows originate on the left side and conclude on the right side.

Furthermore, we introduced a 'Reset' button, offering users the option to clear their selections and initiate the task anew.

4.9 Labyrinth Task

For the labyrinth task, a different approach was used. Due to the nature of the task, a more detailed examination of the original code was necessary to gain a comprehensive understanding of how the algorithm functions and to identify potential modifications. Below is a concise overview of how the original algorithm operates:

1. **Initialization:**
   - Create a grid of cells with dimensions x by y.

   - Set up variables for horizontal walls (*horiz*) and vertical walls (*verti*).
   - Choose a random starting cell within the grid.
   - Initialize a path stack with the starting cell.
   - Create a 2D array named *unvisited* to track unvisited cells, marking all cells as unvisited initially.
2. **Maze Generation Loop (until all cells are visited):**
   (a) Find neighboring cells that are unvisited and can be potentially visited next.
   (b) If valid neighboring cells exist, randomly select one.
   (c) Mark the chosen neighbor as visited.
   (d) Remove the wall between the current cell and the chosen neighbor (either horizontal or vertical).
   (e) Push the chosen neighbor onto the path stack.
3. **Backtracking:**
   - If no valid neighbors are available for the current cell, backtrack by popping cells from the path stack until a cell with unvisited neighbors is encountered.
4. **Repeat steps 2 and 3 until all cells in the grid are visited.**
5. **Final Maze:**
   - The generated maze includes information about its structure, including the placement of horizontal and vertical walls.

Although, the algorithm is somewhat flexible, due to how it is implemented and how the maze is printed on to the PDF, it became clear that modifying it was a complex endeavour and it would require a higher than expected effort to adjust it to the project needs.

With this in mind, we began searching for a lower-effort alternative on the Internet that would align with the project's requirements. We found a suitable alternative using React on *CodeSandbox* [32]. This new algorithm offered several advantages compared to the previous one:

1. **Improved Flexibility:** The React-based algorithm provided greater flexibility in adapting to the project's specific needs. Unlike the previous algorithm, which proved complex to modify, this alternative allowed for more straightforward adjustments.
2. **Enhanced Efficiency:** The new algorithm demonstrated improved efficiency in maze generation. It offered a faster and more reliable approach to creating randomized mazes, aligning with our project's requirements for quick task generation.
3. **Integration with React:** Leveraging React allowed for seamless integration with our frontend components, simplifying the overall development process.

While retaining some similarities to the previous algorithm, such as maze generation principles, the new approach brought significant enhancements that better suited our project's goals.

Considering these factors, we made the decision to adopt the new algorithm, tailoring it to our specific needs. While we retained the maze model, the generation algorithm was integrated into

our '*mazeTask*' function by importing it from a separate file, along with the inclusion of specific instructions. Subsequently, we proceeded to implement the 'taskObject' function which returns an object with the following properties:

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- maze: An array, provided by the new algorithm, representing the maze

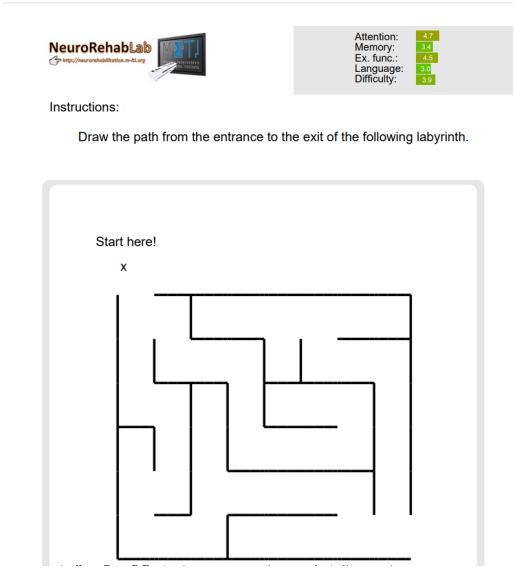
The remaining task within our backend was to establish a server route dedicated to processing requests for maze generation and subsequently provide the data to the frontend.

In transitioning to the frontend development phase, we opted to capitalize on the code base available in the CodeSandbox project. Subsequently, we enhanced it by incorporating additional functionalities to meet our specific requirements. While retaining the original maze creation functionality, we introduced a virtual directional pad, commonly referred to as a D-pad, as can be seen in Figure 16b. The purpose of this addition was to facilitate user navigation within the maze. Given the project's tablet-oriented nature, it was imperative to provide a means for user interaction. Moreover, users were also given the option to use their keyboard for navigation.

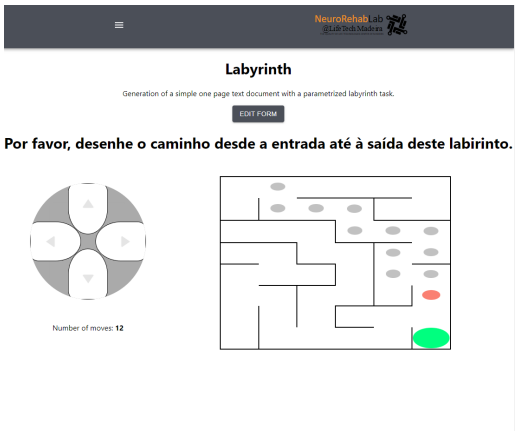
To manage and record the user's path, we introduced a new state variable called '*userPath*.' This variable is dynamically updated each time the user makes a move, whether by utilizing the D-pad or the keyboard. The '*userPath*' variable functions as an array, where each element is another array representing the positions visited by the user. It records both new and previously visited positions, enabling us to keep a comprehensive log of the user's movements throughout the maze. This data serves multiple purposes, primarily allowing us to monitor and display the total number of moves made by the user as a statistical metric.

We organized the content using Material UI's 'Grid' for improved layout. This arrangement places the D-pad on the left side (later it was changed to the right side) and the maze to fit on the right side effectively (later it was changed to the left side). Through the use of CSS we were able to resize the maze for a better user experience.

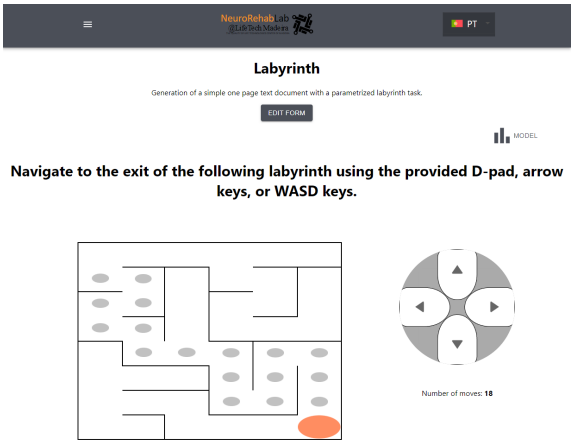
After completing the modifications to the maze task's frontend, we proceeded to build the 'MazeTaskForm.' In contrast to the previous methodology, the 'Maze' component was constructed before the 'MazeTaskForm.'. Material UI elements were used to modernize the original form. To manage client-server communication we employed 'axios' to execute a POST request for task generation, and the data received in the server's response was stored in the 'mazeTask' state variable. This data was subsequently passed to the 'Maze' component.



(a) Labyrinth Task generated using the original TG



(b) Labyrinth Task generated using the TG2.0



(c) Labyrinth Task after completion using the TG2.0

Fig. 16: Comparison of Labyrinth Tasks between TG 1.0 (a) and 2.0 (b)

4.10 Memory Recall Task

Following our established methodology of analyzing and adapting code from the original Task Generator, it became evident that several changes were necessary, particularly in how the data was structured.

Similar to the 'Action Sequencing Task,' the data for this task was initially stored in multiple arrays, each dedicated to a specific type of story (short, medium, long, and image). Additionally, there was an array containing questions associated with each story type, with varying numbers of questions for each type - 4 for short stories, 6 for medium stories, 10 for long stories, and 9 for image stories.

This approach to data storage proved to be inefficient and challenging to manage. To address these issues, we decided to implement a similar data structure as in the 'Action Sequencing Task.' Simultaneously, we made a significant change in how users would interact with this task. Instead

of answering questions in text form, users were presented with 'True or False' statements and were required to select whether each statement was true or false.

To facilitate this transition, separate files were created for each story type, and each story followed the following structure:

```
{
    "title": "",
    "story": "",
    "statements": [
        {
            "text": "",
            "evaluation": true
        },
        {
            "text": "",
            "evaluation": false
        }
    ]
}
```

Fig. 17: Structure of story object

This restructuring not only improved the efficiency of data management but also enhanced the user experience by introducing a more interactive element to the task.

After completing the major restructuring, we made smaller adjustments to ensure the algorithm could properly extract the newly structured data. Additionally, we introduced the '*taskObject*' function, which merges the task model and the generated task, resulting in an object with the following properties:

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- image: The image extracted from image stories
- text: A variable holding the story
- sentences: An array holding the 'statements' objects extracted from the story

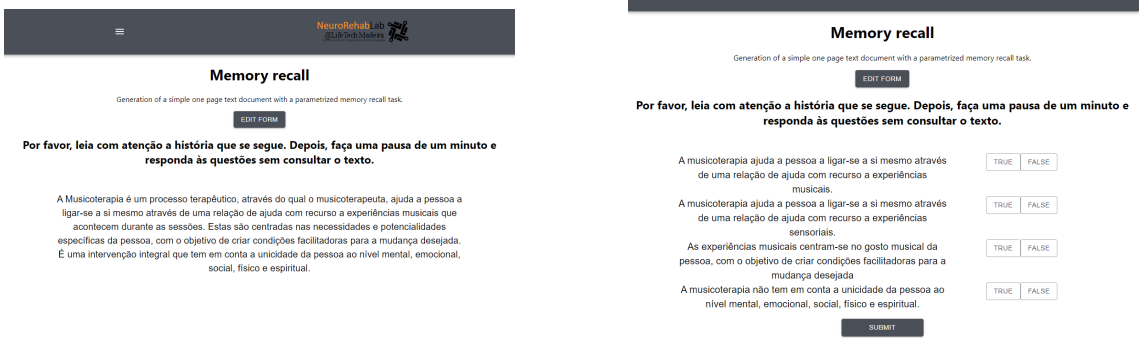
To complete the backend implementation, we created a dedicated route to handle frontend requests for task generation, returning the generated data.

The frontend implementation commenced with the development of the '*MemoryRecallTaskForm*'. Following our established approach, we utilized the original form as a foundation and redesigned it using Material UI to achieve a more modern appearance. For the management of client-server communication, we employed the 'axios' library to facilitate a POST request. Subsequently, the received data is stored within the '*storyTask*' state variable and subsequently passed to the '*MemoryRecallTask*' component.

The 'MemoryRecallTask' was designed with the premise that users would first engage with a story or an image and subsequently evaluate the veracity of provided statements. Accordingly, the task was structured to initially present the story, whether in text or image format, see Figure

18a. Utilizing a timer mechanism, when the timer reaches a designated threshold (currently set at one minute), the story becomes hidden, and the statements become visible.

By making effective use of Material UI's 'Grid' layout, we organized the sentences on the left side of the user interface, with Material UI 'Button' elements on the right side, see Figure 18b.



(a) Memory Recall Task displaying story

(b) Memory Recall Task displaying sentences and buttons

Fig. 18: Memory Recall Task generated using TG2.0

For the management of user interaction and data handling, we introduced the state variable '*selectedAnswers*.' Whenever a 'Button' is clicked, a function is triggered to update '*selectedAnswers*' based on the index of the statement and the value of the clicked 'button' (true or false). If the sentence already exists in the variable, its 'evaluation' property is accordingly modified with the selected value.

Lastly, a 'Submit' button was incorporated to provide users with the option to submit the completed task.

4.11 Scrambled Words Task

For the implementation of the Scramble Words Task, we followed our established methodology, beginning by retrieving the original code and conducting a comprehensive analysis to adapt it to our project's requirements. This process commenced with the removal of code related to PDF file creation. We further modified one of the arrays present in the original code, namely the 'target' array. Its data was relocated to the 'translations' folder and is now retrieved based on the selected language.

An intriguing observation during this process was that the original algorithm, while not explicitly saving these positions, inherently retained the positions where the letters were placed within the grid. In other words, the potential for capturing this data existed, although it had not been capitalized upon. Recognizing this opportunity, we introduced the variable '*wordPositions*' to the algorithm, thereby enabling us to record and utilize the positions of each letter within the grid effectively.

To merge both the model and the generated task, we implemented the '*taskObject*' function. The resulting object encompasses the following properties:

- model: The object provided by the model function.
- instructions: Task-specific instructions.
- n_elements: A variable denoting the number of letters to be displayed on the grid.
- words: An array containing the words to be located.
- matrix: A 2D array representing the grid for display.
- matrix_words: A 2D array rendering the grid, displaying only the words.
- wordsWithPositions: An array of objects, with each object representing a word and its corresponding letter positions.

To complete the backend implementation, a server route was established to intercept frontend requests for generating this specific task. The server then returns the generated data to the frontend.

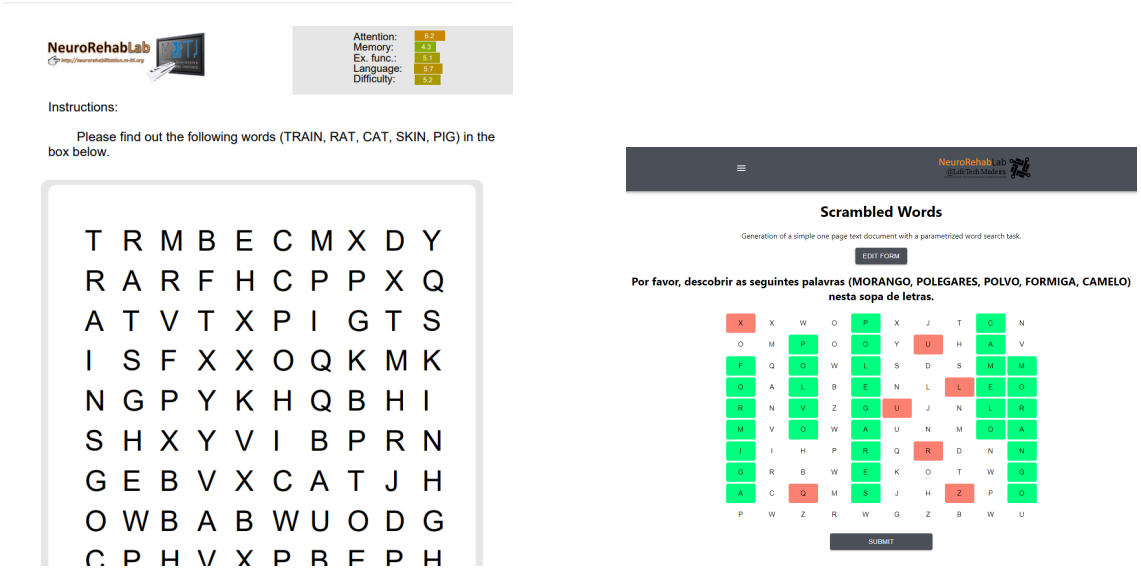
The frontend implementation commenced with the '*SoupTaskForm*' component. In this phase, the original form underwent a transformation using Material UI to enhance its visual appeal. For managing form submissions, we utilized 'axios' to execute a POST request to the backend. Subsequently, the data received from the backend's response was stored in the '*soupTask*' state variable, ready for transmission to the '*SoupTask*' component.

To develop the 'SoupTask' component, we initially concentrated on displaying the grid, making use of Material UI's 'Grid' element in combination with HTML's 'table' element. However, before delving into the details of how we implemented interactivity, it's essential to explore the different interaction methods we considered.

We evaluated several interaction methods for this task, each with its own set of advantages and challenges. One of the methods we contemplated was the click-and-drag approach, which initially seemed promising. However, it revealed to be more complex than expected and required significant time and resources to develop.

We also considered simpler approaches to enhance user interaction. One option was to allow users to click on each letter of a word individually, with the word being highlighted only when all its letters were selected. Another approach involved highlighting the entire word upon clicking any of its letters. We opted for the latter, due to its simplicity as well as lower effort requirements to develop.

With this approach in mind, we replaced the content of each table cell with Material UI 'Button' elements. Additionally, we created a new function to handle click events.



(a) Scrambled Words Task generated using the original TG

(b) Scrambled Words Task generated using the TG2.0

Fig. 19: Comparison of Action Sequencing Tasks between TG 1.0 (left) and 2.0 (right)

Whenever a letter is clicked, it updates the state variable '*highlightedLetters*' with the letter's 'rowIndex' and 'colIndex'. If the letter belongs to a word, the state variable '*highlightedWords*' is also updated to include that word, resulting in its highlighting.

Conversely, if the letter is not part of a word, we update the state variable '*missedPositions*' and highlight the letter in red, see Figure 19b.

To complete the task, a 'Submit' button was added to allow for users to submit the task.

4.12 Image Pairs Task

Following our established methodology, we initiated the implementation by sourcing the original code and analyzing it to identify necessary adjustments, starting with the removal of any code related to PDF file creation.

Subsequent adjustments followed a similar approach to what we employed for the 'Association Task.' We introduced two new arrays, '*left*' and '*right*,' each representing the images to be placed on their respective sides of the page. The '*leftObjects*' and '*rightObjects*' arrays house these images, encapsulated in objects that consist of an 'id' property (generated using npm's 'nanoid' library) and an 'image' property. It's worth noting that objects at corresponding indices in both the '*leftObjects*' and '*rightObjects*' arrays share the same 'id'.

Further adjustments were made to accommodate the current data structure while preserving the underlying logic.

Subsequently, we implemented the 'taskObject' function to unify both the model function and the task generation function, resulting in an object with the following properties:

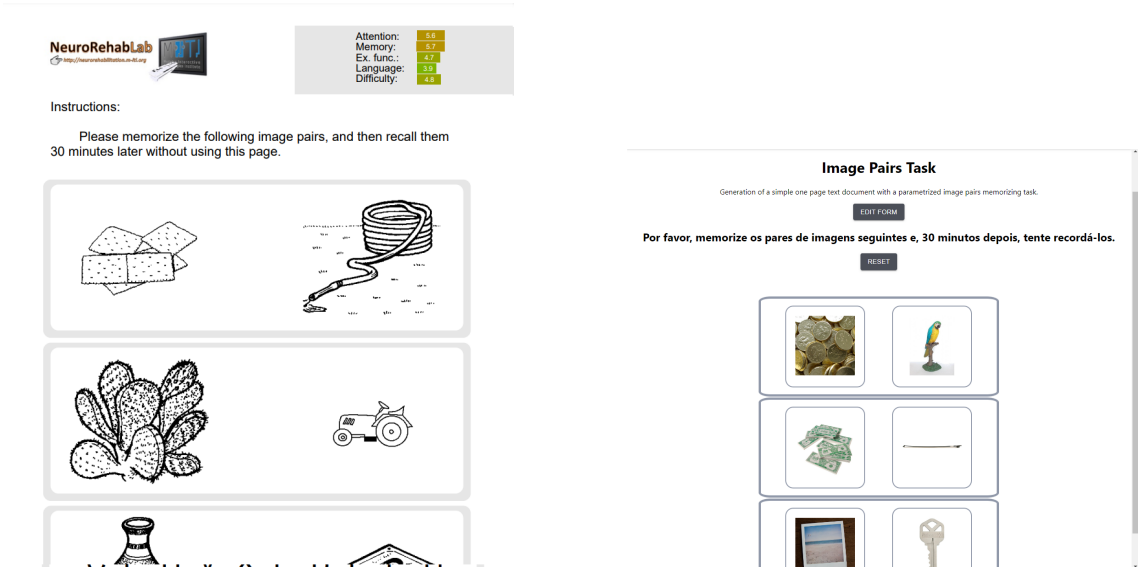
- model: The object provided by the model function.
- instructions: Task-specific instructions.

- leftObjects: An array of objects to be displayed on the left side.
- rightObjects: An array of objects to be displayed on the right side.
- pairings: An array of objects, each representing a pair of objects, with one object from each of the previous two arrays.
- note: A note referencing images sourced from the IPNP database, visible only when clipart images are utilized.

As observed, this object shares similar properties with the one used in the 'Association Task'. This consistency in object structure helps streamline the implementation.

To complete the backend implementation, we established a server route to intercept frontend requests and return the generated task object to the frontend.

In the frontend implementation, we initiated the process with the 'ImagePairsTaskForm' component. This form underwent a comprehensive visual makeover using Material UI to enhance its appearance. For form submission management, we employed 'axios' to execute a POST request, and the received data was stored in the 'imagePairs' state variable. This variable was subsequently passed to the 'ImagePairsTask' component, where we focused on constructing the user interface.



(a) Image Pairs Task generated using the original TG

(b) Image Pairs Task generated using the TG2.0

Fig. 20: Comparison of Image Pairs Tasks between TG 1.0 (left) and 2.0 (right)

The 'ImagePairsTask' component was constructed with a similar approach to the 'AssociationTask' component. We utilized Material UI's 'Grid' and HTML's 'table' to organize elements on their respective sides. Additionally, a custom 'Card' component was employed to frame the images. To preserve the originality and maintain clarity of the image pairings, we added borders to separate each pair, see Figure 20.

We implemented a timer mechanism to control the display duration of the image pairs. Once the timer expires, the image pairs are hidden, and a message prompts users to return to the task after a specified interval (30 minutes).

4.13 Patient Profile

The 'Profile Task,' serving as the culmination of our efforts, was the final task to be implemented. Adhering to our systematic methodology, we initiated the process by retrieving the original code and conducting a comprehensive analysis to identify necessary adjustments. As with previous tasks, sections of code pertaining to PDF file creation were meticulously excluded.

We introduced a new object variable, denoted as '*taskData*,' dedicated to the storage of task-specific data. This variable stores each task generated as a property, for better readability and maintenance.The structure of '*taskData*' includes the following properties:

- actionSeq: An object representing the Action Sequencing Task
- association: An object representing the Association Task
- cancellation: An object representing the Cancellation Task
- categorization: An object representing the Categorization Task
- context: An object representing the Context Task
- imagePairs: An object representing the Image Pairs Task
- maze: An object representing the Labyrinth Task
- memrecall: An object representing the Memory Recall Task
- problem: An object representing the Problem Resolution Task
- sequencing: An object representing the Number Sequencing Task
- soup: An object representing the Scrambled Words Task

It's important to note that, given the nature of the algorithm, certain tasks may not always be generated, which can result in the '*taskData*' variable not consistently containing all the properties listed above.

Additionally, adjustments were made to the way data is accessed. In the original code, data was accessed using indexes, primarily because model data was returned as an array. However, with the new data structure in place, data is now accessed using object properties.

To finalize the backend implementation, we established a server route designed to intercept frontend requests and return the '*taskData*' object.

Moving on to the frontend implementation, we began with the '*ProfileTaskForm*' component, where we overhauled the original form using Material UI. In order to retain some of the original design elements, we also incorporated the 'Slider' element.

To manage form submission, we utilized 'axios' to perform a POST request to the server, triggering the task generation process. The resulting data from the server's response is stored in a state variable named '*profileTask*', which is subsequently passed as a prop to the '*ProfileTask*' component.

For the '*ProfileTask*' component, a unique approach was adopted. Given that the 'Profile Task' combines elements from all other tasks, it was deliberately saved as the final task to implement. Instead of creating a new component from scratch, we chose to import and render the existing task components as needed.

We introduced a variable named '*taskArray*' for this purpose, which takes the 'task' object received in the props and transforms it into an array of objects using the built-in 'Object.entries()' function. Each object in this array represents a task and contains a 'name' property along with the 'data' property.

Additionally, we introduced another variable called 'taskComponents' to hold references to various task components. Using the state variable '*currentTaskIndex*', the 'CurrentTaskComponent' holds a reference to the current task to be rendered. If valid 'data' is available for the current task, it is rendered. However, if there is no data for the current task, it proceeds to the next task by incrementing '*currentTaskIndex*'.

We preserved the original task execution order from the Task Generator, beginning the cognitive training program with the 'Image Pairs Task.' Throughout the entire training program, a 'Next' button is consistently displayed, allowing users to transition to the next task. When users reach the final task, the button transforms into a 'Submit' button, signifying the program's completion.

4.14 Electron.js App

Throughout the development and testing phases, collaboration with psychology experts played a crucial role. Engaging in regular meetings, these experts provided invaluable insights. During these collaborative sessions, a pivotal requirement emerged: the need for the application to operate seamlessly in offline environments.

In response to this new requirement, the NeuroRehabLab Task Generator 2.0 underwent a significant transformation, transitioning from a web-based React application to a standalone Electron.js application. By encapsulating the React app within the Electron framework, the application gained the ability to function independently, providing users with seamless access to cognitive training tasks without the dependency on an internet connection.

While seeking a solution to make the NeuroRehabLab Task Generator 2.0 accessible in offline environments, various frameworks were considered. Electron.js emerged as the optimal choice, facilitating the transformation of the React app into a standalone application.

It's noteworthy that this decision followed meticulous experimentation with alternative frameworks such as Cordova, Ionic, and Capacitor. Cordova faced deprecation concerns, and Ionic and Capacitor didn't align with the specific needs of our project. The possibility of revisiting the React Native prototype was also explored, but this option presented challenges, requiring substantial resources and posing limitations. Notably, React Native primarily supports Android, iOS, and web platforms, whereas the NeuroRehabLab tablets operate on Windows OS.

In the process of adapting the existing application, we adhered to the guidelines outlined in the official Electron documentation [33], which necessitated specific configurations. As part of the adaptation process, notable adjustments were made to the communication infrastructure. In the original React app, communication between the client and server relied on HTTP requests.

However, for the application to function seamlessly without internet connectivity, a shift to Electron's IPC (Inter-Process Communication) was imperative.

To illustrate the modifications implemented, let's consider the 'Cancellation' task. As previously highlighted, one significant change involved transitioning from traditional HTTP requests to Inter-Process Communication (IPC) due to the shift towards offline functionality.

In the original implementation, as depicted in Figure 21, an HTTP POST request utilizing the Axios library was employed to send data to the server:

```
const handleSubmit = (e) => {
    e.preventDefault();

    const task = {elements,probability,size,numbers,order}
    //console.log(task)
    setIsLoading(true);

    axios.post(`${REACT_APP_API_URL}/${selectedLanguage}/cancellation`, task)
        .then(
            console.log('posted ' + JSON.stringify(task))
        )
        .then(response=> {
            //return response.json()
            console.log(response)
            console.log("data: " , response.data)
            setCancelTask(response.data)
            setFormSubmitted(true);
        })
        .catch(err => {
        })
        .finally(() => {
            setIsLoading(false);
        });
}
```

Fig. 21: Example of an HTTP POST request using Axios

Figure 21 showcases the code snippet responsible for initiating an HTTP request. Axios, a widely used HTTP client for JavaScript, is employed to perform a POST operation to the server, facilitating data transmission.The corresponding server-side code responsible for handling this HTTP request is represented in Figure 22.

```
app.post('/:language/cancellation', (req,res) =>{
    console.log('POST')
    const  lang  = req.params.language;
    //console.log(language)
    console.log(req.body)

    const cancelTask = ({
        elements: req.body.elements,
        probability: req.body.probability,
        size: req.body.size,
        numbers: req.body.numbers,
        order: req.body.order
    })

    const task = cancel.taskObject(cancelTask.elements,cancelTask.probability,
    cancelTask.size,cancelTask.numbers,cancelTask.order, lang)

    //res.send(task)
    res.json(task)
})
```

Fig. 22: Server-side code handling the HTTP POST request

The server-side code snippet demonstrates the handling of the incoming HTTP POST request related to the 'Cancellation' task.

Now, with the transition to Electron's Inter-Process Communication (IPC), we redefine the communication model. IPC allows seamless communication between the main (backend) and renderer (frontend) processes within the Electron app. Let's explore how this change is reflected in the 'Cancellation' task implementation.

In contrast to the previous HTTP request approach, the IPC implementation leverages Electron's native communication channels. Figure 23 illustrates the modified code snippet for initiating an IPC request:

```
const handleSubmit = (e) => {
    e.preventDefault();

    const task = {elements,probability,size,numbers,order}
    //console.log(task)
    setIsLoading(true);

    window.electron.ipcRenderer.send('cancellation-request', {
        language: selectedLanguage,
        task: task,
    })
}

window.electron.ipcRenderer.on('cancellation-response', (event, response) => {
    // Update your component's state with the response data
    console.log("response", response);
    setCancelTask(response)
    setFormSubmitted(true);
    setIsLoading(false)
})
```

Fig. 23: Example of an IPC request in Electron

In the renderer process, represented by Figure 23, the previous code utilizing Axios for an HTTP POST request has been replaced with Electron's IPC mechanism. The new 'handleSubmit' function now sends an IPC request named 'cancellation-request' to the main process, carrying relevant task data. The response, received through the 'cancellation-response' event, is then processed to update the component's state.

```
ipcMain.on('cancellation-request', (event, requestData) => {
    const { language, task } = requestData;
    const { elements, probability, size, numbers, order } = task;

    // Set the default language to 'pt' if it's not defined
    const defaultLanguage = language || 'pt';

    // Implement the logic for generating the action sequencing task
    const taskObj = cancel.taskObject(elements,probability,size,numbers,order, defaultLanguage)

    console.log('Cancellation task: ', taskObj)
    // Send the response (the task) back to the renderer process
    event.sender.send('cancellation-response', taskObj)
})
```

Fig. 24: Main process handling IPC requests

On the main process side, represented by Figure 24, the server-side logic is adjusted to handle IPC requests. Upon receiving a 'cancellation-request', the main process processes the task data and generates the corresponding task object using the 'cancel.taskObject' function. This task object is then sent back as a response through the 'cancellation-response' event, completing the IPC communication cycle.

5 Results

In this section, we present the results and findings obtained during the development and implementation of the NeuroRehabLab Task Generator 2.0. Our objective in this section is to provide a comprehensive overview of the results obtained during the course of this project.

5.1 Implemented Tasks

In this section, we provide an overview of the tasks that were implemented as part of the NeuroRehabLab Task Generator 2.0. For each task, we assessed the level of implementation and identified any remaining work required to achieve full completion. The task completeness is an estimated value based on the following factors:

1. The algorithm responsible for task generation must be correctly implemented in the backend, ensuring that tasks are generated accurately and as intended.
2. Effective communication between the backend and frontend components is crucial. This includes the proper transmission of task data from the server to the client, ensuring that the frontend receives the necessary information to display the task correctly.
3. The frontend must incorporate user interaction elements, allowing users to engage with and complete the task as intended. This involves implementing interactive features and ensuring their functionality.
4. The ability to record user interactions and submit data to the server for storage is a critical aspect. The completeness assessment considers whether data submission and storage mechanisms are functioning correctly.

The 'Cancellation Task' was successfully integrated into the system, with a great majority of its features implemented, classifying it as 'Mostly Implemented'. The fundamental functionalities of the activity, such as task generation, user interaction recording and data submission, have been effectively implemented. However, it's important to note that the graphical representation of cognitive data, a feature present in the original Task Generator, has not been incorporated into the current version. Additionally, minor refinements to the user interface are required to enhance the overall user experience.

The 'Number Sequencing Task' was successfully implemented, with a substantial portion of its features in place, it was categorized as 'Substantially Implemented'. Core functionalities, including task generation, data recording during interactions, and data submission, have been seamlessly integrated. However, it's important to note that the graphical representation of cognitive data, a feature present in the original Task Generator, has not been included in the current version. Additionally, while we aimed to incorporate a virtual numerical keyboard to facilitate user interaction, this feature could not be delivered as part of the current implementation.

The 'Problem Resolution Task' was effectively implemented, with a significant portion of its functionalities in place, warranting a classification of 'Partially Implemented'. Core functionalities, such as task generation, data recording, and data submission, have been seamlessly integrated into the new system. However, it's important to note that the graphical representation of cognitive data, a feature present in the original Task Generator, has not been included in the

current version. Moreover, we were unable to incorporate a virtual numerical keyboard as originally intended. Additionally, further modifications are required to support translated content in this task.

The 'Association Task' was successfully implemented into the system, with a great majority of its features implemented, warranting the tag of 'Mostly Implemented'. Core functionalities have been integrated such as, task generation, user interaction, data recording and data submission. However, the task could benefit from user interface adjustments. Additionally, the cognitive data graph has not been incorporated, in the current version.

The 'Context Task' has been successfully implemented, with a great majority of its features implemented, it was labelled as 'Mostly Implemented'. Core functionalities were effectively integrated to the application such as task generation, data recording during activity and data submission. However, the graphical representation of cognitive data, present in the original version, has not been integrated. Moreover, the task could benefit from user interface adjustments as well as the inclusion of a 'Back' button, to allow users to go back if they needed to change their selection.

The 'Categorization Task' was successfully implemented into the system, with a great majority of its features implemented, it was categorized as 'Mostly Implemented'. Core functionalities have been integrated such as, task generation, user interaction, data recording and data submission. However the task would benefit from user interface adjustments. In addition, the cognitive data graph has not been integrated.

The 'Action Sequencing Task' was successfully implemented, with a great majority of its features implemented, classifying it as 'Mostly Implemented'. Core functionalities have been integrated such as, task generation, user interaction, data recording and data submission. Moreover, the structural change made to the data storing method was successfully implemented, see figure 25. However, the cognitive data graph has not been implemented. Additionally, some user interface enhancements could be made.

```
var explicit = [
  "To do rice",
  "Pick up the pan.",
  "Put olive oil in the pan.",
  "Put onion in the pan.",
  "Light the fire.",
  "Put water in the pan.",
  "Put rice in the pan.",
  "To take a bath",
  "Get undressed.",
  "Get into the tub.",
  "Adjust water temperature.",
  "Wet the body.",
  "Washed with soap.",
  "Dry with a towel.",
  "To plant potatoes",
  "Smooth the soil.",
  "Fertilize the land.",
  "Dig the soil.",
  "Throw the seed.",
  "Cover the seed with land.",
  "Watering.",
```

(a) Action in the original TG

```
"explicit" : [
    {
        "action" : "To do rice",
        "steps" : [
                "Pick up the pan.",
                "Put olive oil in the pan.",
                "Put onion in the pan.",
                "Light the fire.",
                "Put water in the pan.",
                "Put rice in the pan."
            ]
    },
    {
        "action" : "To take a bath",
        "steps" : [
                "Get undressed.",
                "Get into the tub.",
                "Adjust water temperature.",
                "Wet the body.",
                "Washed with soap.",
                "Dry with a towel."
            ]
```

(b) Action in TG2.0

Fig. 25: Example of action storage method

The 'Labyrinth Task' was implemented by adapting the CodeSandbox example [32]. In addition to this adaptation, we introduced several enhancements, including navigation through the use of the D-Pad and the inclusion of a 'Move Counter'. These combined additions, along with the core functionalities of task generation and data submission, constitute the great majority of the features, thus it was classified as 'Mostly Implemented'. Nevertheless, it's worth noting that there is room for improvement in the user interface through smaller adjustments and optimization of data handling.

The 'Memory Recall Task' required several structural modifications, namely the way data was stored. The data storage method was changed to object based storage, see Figure 17. In addition to the modifications made, core functionalities such as task generation, data recording and data submission were effectively integrated, classifying it as 'Mostly Implemented'. However, the user interface could benefit from enhancements for visual clarity.

The 'Scrambled Words Task' was successfully implemented into the system, with a great majority of its features implemented, classifying it as 'Mostly Implemented'. Core functionalities have been integrated such as, task generation, user interaction, data recording and data submission. However, the task could benefit from user interface adjustments.

For the 'Image Pairs Task', approximately half of the task features have been implemented, thus it was classified as 'Partially Implemented'. Currently, the core functionality of task generation and data submission is in place. However, two critical features are still pending completion: the ability for users to return to the task, and the functionality to rearrange the images, once returning to the task. These remaining features are essential to achieving full task functionality.

The 'Patient's Profile', was successfully implemented, classifying it as 'Substantially Implemented'. Core functionalities, including task generation, data recording and data submission, have been effectively integrated. However, its implementation is strongly dependent on the functionality of the other tasks within the system. Therefore, any missing features or functionalities in those tasks will also affect the 'Patient's Profiles' component's overall completion and functionality. Furthermore, the introduction of a 'Back' Button could enhance the user experience by allowing the user to backtrack and redo a certain task.

5.2 Usability tests

Usability tests were conducted, featuring the application of the System Usability Scale (SUS) questionnaire. These tests served as a pivotal method for assessing the Task Generator 2.0's overall usability, user experience, and operational effectiveness.

To assess the usability of the NeuroRehabLab Task Generator 2.0, participants were engaged in a series of tasks as part of our usability testing procedure. Participants were initially provided with a brief introduction to the project's background, objectives and significance. A tablet was given to the participants equipped with the online version of the application.

The initial task assigned to participants involved completing a comprehensive training program through the 'Patient Profile' task, where they configured all parameters with a predefined value of 3 and left the checkbox 'Only generate tasks closely matching the profile' unchecked. These specific parameter values were strategically chosen based on internal testing: firstly, to ensure the generation of a full training program without encountering issues, and secondly, to generate tasks of relatively short duration and complexity. This approach allowed us to systematically test every task within a full training program without overwhelming the participants. In addition, we maintained consistency across participants by using the same parameter values throughout the study, ensuring uniformity in the results and facilitating comparative analysis.

To foster a comfortable testing environment, participants were assured that none of the data submitted during the tasks would be recorded. This approach aimed to encourage participants to engage freely with the platform, alleviating concerns about performance evaluation and allowing us to focus on their experience and interaction with the system.

The second task involved providing participants with the opportunity to freely explore the application, employing the 'think-aloud' method. Participants were encouraged to articulate their thoughts and actions, providing valuable insights into their user experience. This open exploration aimed to uncover user interactions, preferences, and potential challenges, contributing to a more comprehensive understanding of the application's usability and user-friendliness.

Age Group	Gender	Occupation	Education Level	Field of Work/Study	What is your comfort level with technology?
18-25	Male	Student	Bachelor's Degree	Education, Healthcare / Medical, Science / Research	5
18-25	Female	Student	Bachelor's Degree	Healthcare / Medical	3
36-45	Female	Employed full-time	Doctoral Degree	Social Sciences	4
18-25	Male	Student	High School Diploma	Psychology	4
18-25	Male	Student	High School Diploma	Education, Healthcare / Medical, Science / Research	5
18-25	Male	Student	High School Diploma	Healthcare / Medical, Science/Research	5
18-25	Male	Student	Bachelor's Degree	Finance / Business	5
18-25	Male	Student	High School Diploma	Healthcare / Medical, Science / Research	4
18-25	Female	Student	High School Diploma	Healthcare / Medical	4
Under 18	Female	Student	High School Diploma	Healthcare / Medical	4
26-35	Female	Student	High School Diploma	Education	3

Table 1: Participants' demographic data

Finally, participants were asked to provide their subjective feedback by responding to the System Usability Scale (SUS) questionnaire.

The demographic data, detailed in Table 1, was gathered using the SUS questionnaire following the completion of the assigned tasks. Eleven participants, comprising 5 females and 6 males, were recruited across various age groups and professional backgrounds.

Noteworthy, we have adopted an inclusive recruitment approach, and participants were not selected based on specific criteria, except for the exclusion of individuals involved in the development of the project, here described, to ensure unbiased testing results.

5.2.1 Quantitative results

This section aims to present the quantitative data obtained from the System Usability Scale (SUS) questionnaire. Results for each question will be detailed, with responses recorded on a Likert scale for comprehensive analysis.

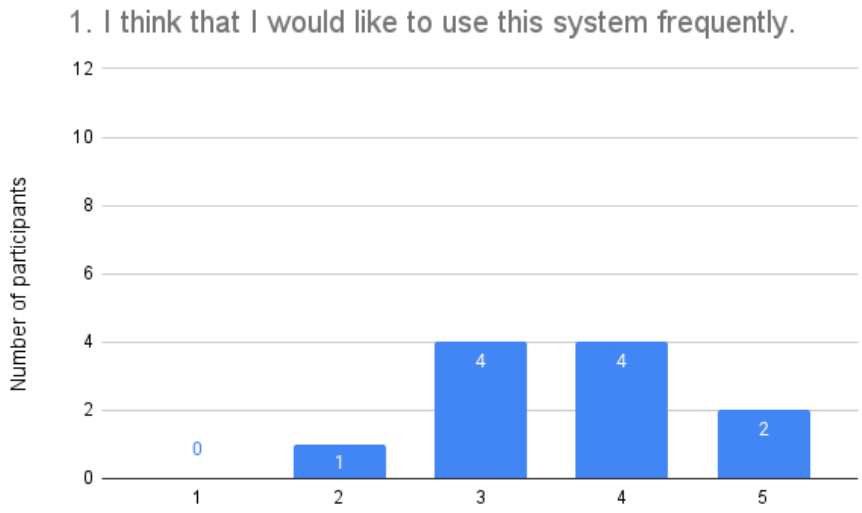


Fig. 26: Results of question "1. I think that I would like to use this system frequently.".

Figure 26 illustrates that 54.5% of participants (6 participants) expressed a positive inclination toward frequent use of the system, while 36.36% remained neutral, and one participant disagreed.

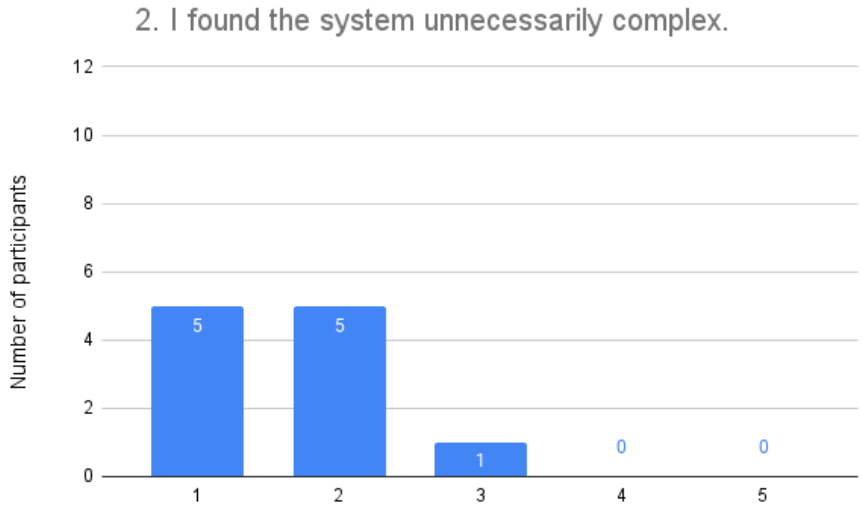


Fig. 27: Results of question "2. I found the system unnecessarily complex.".

The majority of participants (90,9%) disagree that the system was too complex, as evident by Figure 27.

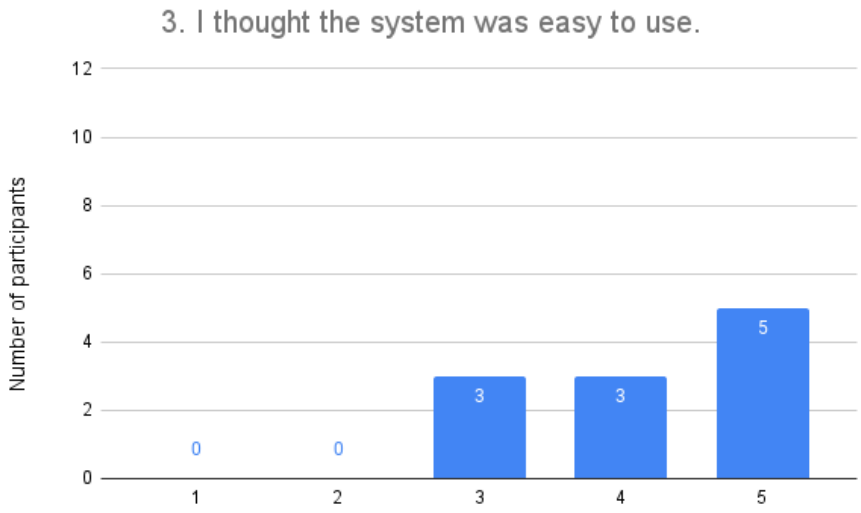


Fig. 28: Results of question "3. I thought the system was easy to use.".

On Figure 28, the results indicate that a majority of participants (72,7%) though the system was easy to use, while 3 participants remained neutral.

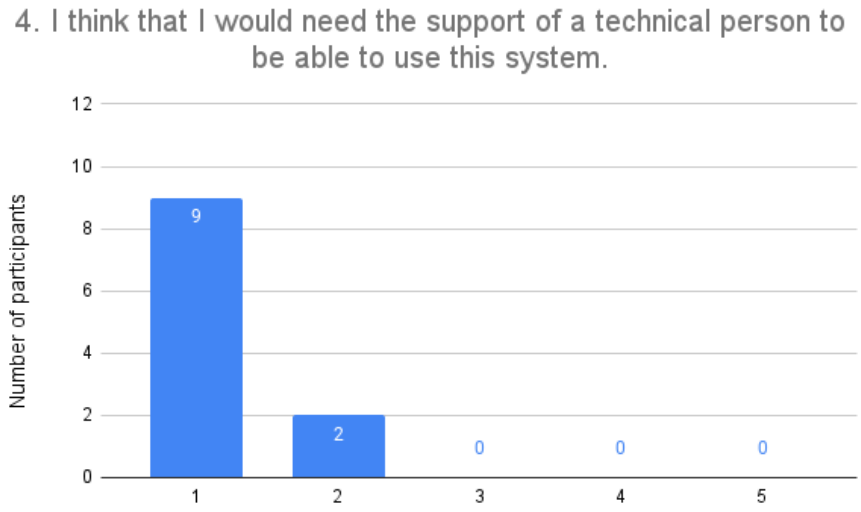


Fig. 29: Results of question "4. I think that I would need the support of a technical person to be able to use this system.".

Regarding the question whether participants required help from a technician the results are positive with 81,8% of participants indicating that they completely disagree and the remaining 18,2% indicating that they disagree.

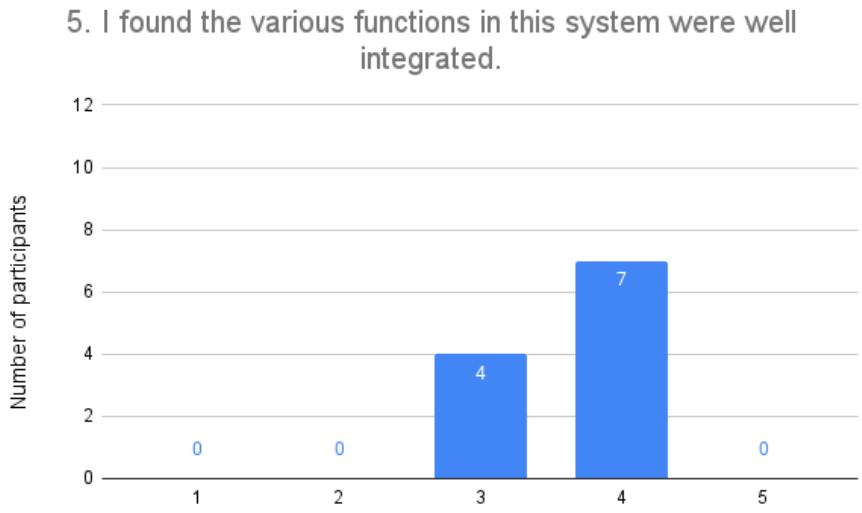


Fig. 30: Results of question "5. I found the various functions in this system were well integrated.".

Analysis of Figure 30 reveals that a majority (63.6%) perceived the integration of multiple features in the system as satisfactory, while 36.4% maintained a neutral stance. Although some participants reported encountering bugs, these issues did not significantly impede task execution.

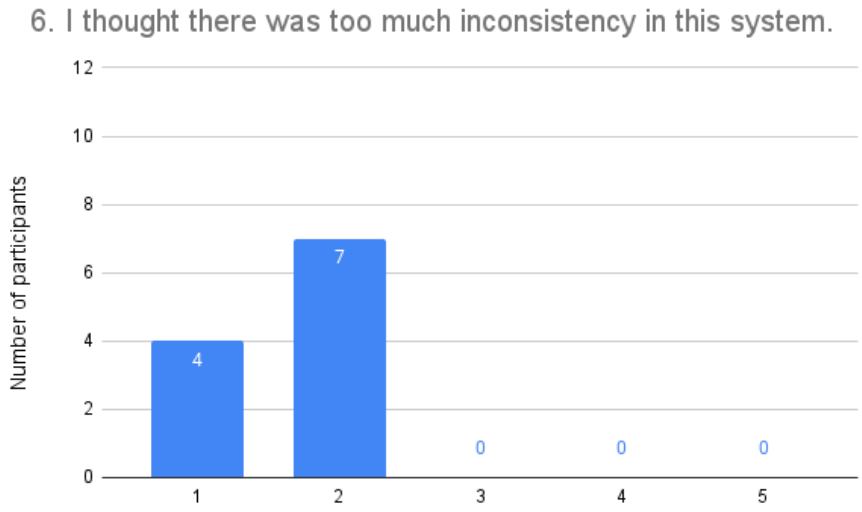


Fig. 31: Results of question "6. I thought there was too much inconsistency in this system.".

Observing Figure 31, 4 participants strongly disagree and 7 disagree that there are inconsistencies in the system, an overall positive result.

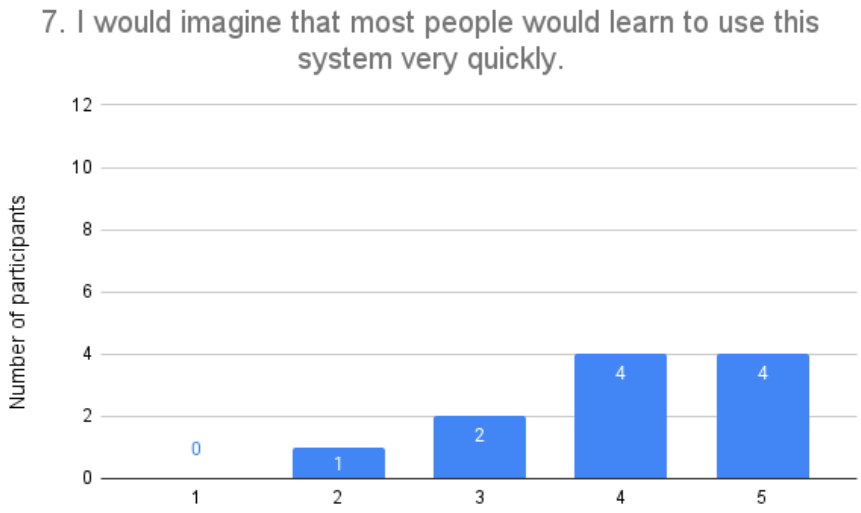


Fig. 32: Results of question "7. I would imagine that most people would learn to use this system very quickly.".

Regarding question 7, the majority (72,7%) agrees that most people could use the system quickly, 2 participants remained neutral and 1 participant disagrees.

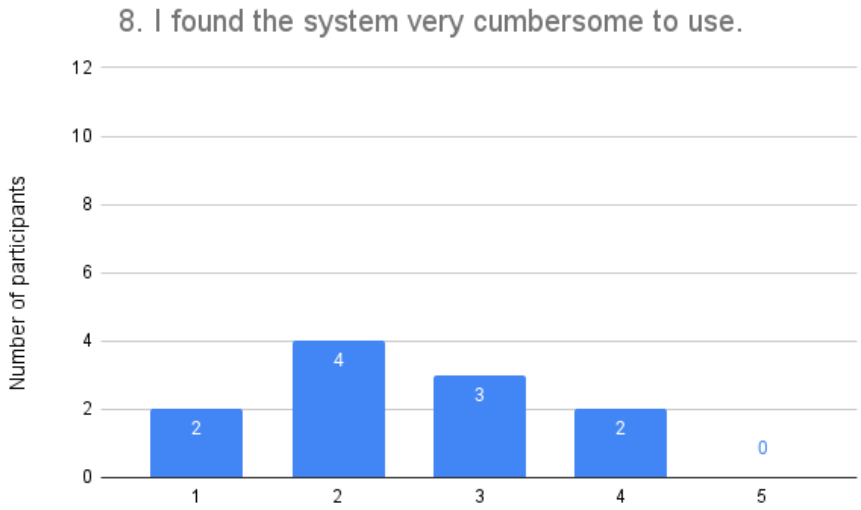


Fig. 33: Results of question "8. I found the system very cumbersome to use.".

When asked if the system was cumbersome to use, 6 participants disagreed, of which 2 strongly disagreed, 3 participants neither agreed of disagreed and 2 participants agreed.

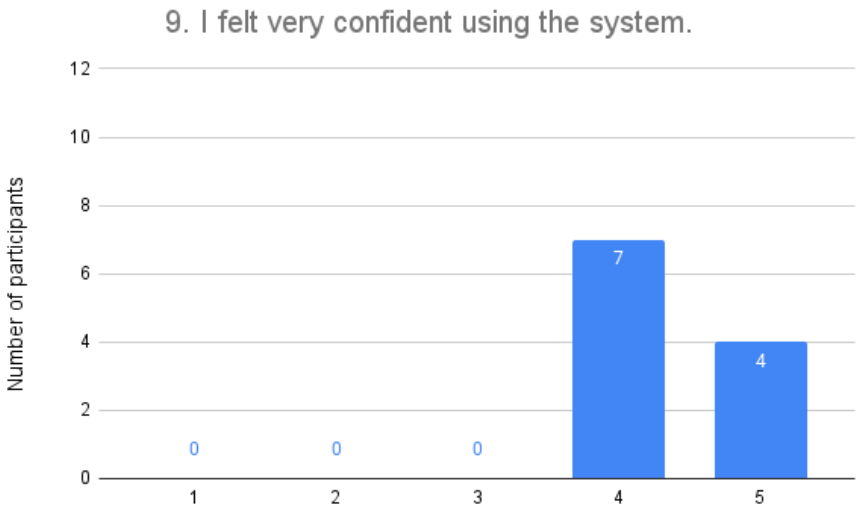


Fig. 34: Results of question "9. I felt very confident using the system.".

Figure 34 shows the results regarding how confident participants were using the system. The results were positive with all participants reporting being confident.

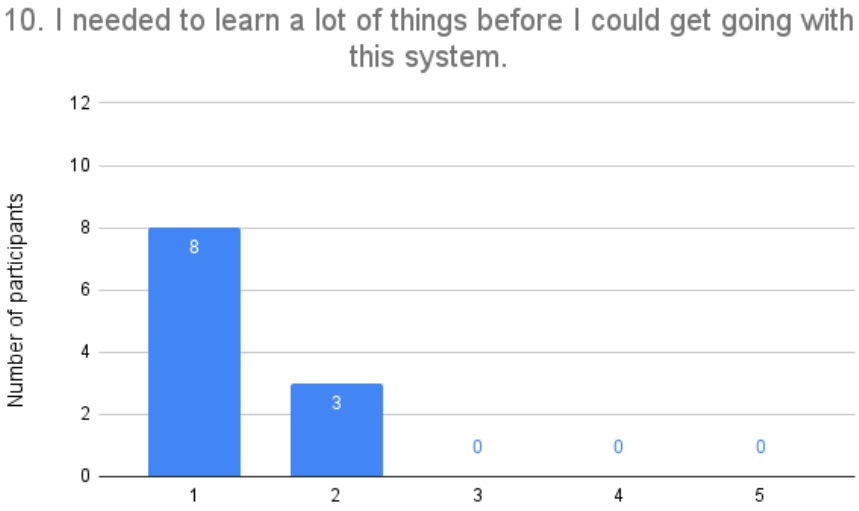


Fig. 35: Results of question "10. I needed to learn a lot of things before I could get going with this system.".

For the final SUS question, the results are represented in Figure 35, where participants were asked if they needed to learn a lot before using the application. All participants disagreed, with eight of the eleven strongly disagreeing.

The data collected from the SUS questionnaire was employed to calculate the overall SUS score for every participant, a value ranging from 0 to 100 points. The scoring formula, as defined by

Brooke in 1996 [34], is expressed as:

$$\begin{aligned} SUS =& (Q1 - 1) + (5 - Q2) + (Q3 - 1) + (5 - Q4) + (Q5 - 1) \\ & + (5 - Q6) + (Q7 - 1) + (5 - Q8) + (Q9 - 1) + (5 - Q10) \times 2.5 \end{aligned}$$

The individual SUS scores from each participant were calculated using the formula, and the average SUS score across all participants was determined. The resulting average SUS score for the NeuroRehabLab Task Generator 2.0 usability evaluation was 77,5.

Additionally, participants were provided with an open-ended question, asking, 'What would you like to see changed/added?' This query aimed to capture insights into aspects of the application not covered by the SUS questionnaire and to gather generalized feedback regarding potential improvements to the system. Furthermore, it's worth noting that not all participants provided feedback in response to the open-ended question. However, some of the received responses included suggestions for improvement, such as:

- Enhanced clarity in instructions for certain tasks.
- Proposal for visualizing the movement of particles by drawing on the screen instead of using a cross pad.
- Identification of potential bugs that need correction.
- Recommendation to reposition the 'Submit' button below the 'Next' button for better intuitiveness

6 Discussion

In the preceding sections, we have outlined the development process and technical aspects of the NeuroRehabLab Task Generator 2.0. Now, we delve into a comprehensive discussion of the implications, insights, and outcomes of our work. This section aims to contextualize our achievements, evaluate their significance, and explore potential avenues for future research and development.

We successfully transformed the traditional Task Generator into its digital counterpart by meticulously adapting and optimizing the original algorithms. This digital version represents a significant leap in accessibility and usability, enabling users to engage with cognitive training and rehabilitation tasks through web-based interfaces.

We successfully achieved several key objectives in our project, including the separation of the task generation logic from the task presentation, as well as the integration of interactive elements. However, it's essential to note that the validation of the remaining requirements hinges on usability testing.

6.1 Implementing the TG in a digital artifact

To transform the original Task Generator, several important decisions were made that would affect the development of the application described in this thesis, and, consequently, many challenges were encountered along the way.

During the planning and brainstorming phases, we gathered some key requirements, one of which led to the decision of using a client-server architecture for the application. This architecture allowed us to separate the presentation layer (client) from the computational and data handling layers (server), providing several advantages in terms of scalability and maintainability. However, implementing this client-server architecture presented its own set of challenges. Transitioning the task generatiom logic to the backend and displaying tasks on the frontend required a seamless means of communication between the two components. We resolved this challenge by employing HTTP requests, with 'axios' as our chosen library. This decision was influenced by our adoption of React for the frontend, and 'axios' was selected for its widespread usage and user-friendly features.

In addition, another critical choice was the selection of the technology for the frontend development. React, a popular JavaScript library for building user interfaces, was chosen for several reasons. React's declarative and component-based approach offered a structured and efficient way to manage the user interface components, facilitating the development process. React's strong community support, extensive ecosystem of libraries, and broad adoption across the industry made it a suitable choice for our project.

It's important to note that we explored alternative approaches during the early development phases. One such approach involved the consideration of React Native, a framework for building native mobile applications using React. A prototype was developed which offered the potential to create a cross-platform native application that could run on various devices. However, after careful evaluation, we decided to drop it. This decision was driven by considerations of development effort and resource allocation, as developing and maintaining a React Native application while also developing a React application posed additional challenges and complexities.

Another critical aspect to consider is the transformation of the data storage structure employed in the previous version. We implemented fundamental structural changes in how data is managed, transitioning from arrays to JSON objects. This transformative change was applied across multiple task generation algorithms, yielding several benefits. Notably, it enhances maintainability and readability while significantly improving the project's scalability.

We implemented a uniform 'data model', as discussed in Section 4.1, to streamline our development process and maintain consistency in data handling. Although our initial aim was maximum abstraction, we successfully achieved a consistent data structure.

We chose to replace certain HTML elements with Material UI components to modernize the application's appearance and leverage the user-friendliness and capabilities offered by Material UI.

In the development of each task, we chose to divide it into two distinct components: the 'TaskForm' and the 'Task.' Each component serves a specific purpose. The 'TaskForm' is responsible for sending information to the server, which in turn responds with the generated task. This generated data is then passed to the 'Task' component, which not only handles the task's display but also manages its interactivity. This clear separation of responsibilities between the 'TaskForm' and 'Task' components proved beneficial for troubleshooting errors and overall application management.

In the later stages of development, after having tested our web application, new requirements emerged from analysing user feedback and experts suggestions. This led to the development of the standalone version of the application using Electron, as described in section 4.14. This adaptation not only addressed user feedback but also enabled the use of the application in offline environments.

6.2 Validating TG as a digital artifact

Upon analyzing the results obtained from our usability tests, it is noteworthy that the overall average SUS score stands at 77,5. Comparatively, when benchmarked against the industry standard of 68, this score indicates that our application is evaluated as above average, reflecting a positive perception of usability among participants.

Upon closer inspection of the questionnaire responses, the overall results indicate a positive perception among users. Nonetheless, specific questions, particularly questions 1, 5, and 8, stand out with less favorable results.

Examining the responses of question 1, shown in Figure 26, it's noteworthy that four participants selected a response of 3 (Neutral), indicating a neutral stance on their inclination to use the system frequently. Another participant rated the system as 2 (Disagree), suggesting a lower willingness to use it frequently. It is crucial to consider that the system is designed for application in clinical settings, and the usability tests were conducted with healthy individuals. This context provides a potential explanation for these results, as the participants may not see themselves frequently using a system designed primarily for cognitive rehabilitation, a scenario that might not align with their immediate needs.

Analyzing the responses to Question 5, it is observed that four participants chose a rating of 3 (Neutral). This suggests that these participants neither expressed satisfaction nor disapproval of

the implemented features. Importantly, during the testing sessions, participants provided valuable feedback in the form of suggestions for improving the application, aligning with these neutral responses.

Analyzing the responses to Question 8, we observe that three participants chose a response of 3 (Neutral), and two participants chose a response of 4 (Agree), indicating a perception that the system might be somewhat cumbersome to use. These responses are consistent with our observations during the usability tests, where some participants demonstrated a degree difficulty in executing certain tasks. It further underlines the importance of addressing usability issues to enhance the overall user experience.

In addition to quantitative data, we collected valuable qualitative feedback during usability testing. Participants shared insights, suggestions, comments, and critiques, offering a deeper understanding of their experiences with the system, but we also gained valuable insights by closely watching how participants interacted with the application. For instance, a common comment provided was that task instructions were not clear or did not align with how the task was intended to be execute.

Other insights were gathered in the 'Scrambled Words' task. Users initially attempted to drag their fingers across the screen to select a word. However, the intended interaction was designed differently - users were meant to select a letter belonging to a word, to highlight it and thus selecting it. This initial discrepancy led to confusion among participants, prompting a valuable suggestion for improvement: users recommended changing the interaction to require selecting every letter for the word to be chosen. Participants also recommended incorporating a visual indicator or highlighting mechanism to denote which words had already been selected. This enhancement aims to provide users with clearer progress tracking during the task.

Additional suggestions were related to the 'Labyrinth' task. One participant recommended drawing the path on the maze instead of using the provided Dpad, proposing a new method of navigation that allows users to draw their path directly on the screen, as described in the instructions. However, this suggestion highlighted a discrepancy with the instructions and the intended interaction. While we considered and evaluated the feasibility of this suggestion, we ultimately decided not to change the interaction method due to its complexity and the resources required for implementation. Instead, adjustments were made to the instructions to better clarify the intended interaction method. Another participant suggested a dynamic adjustment for the 'Labyrinth' task by proposing that the Dpad should follow the user's position within the maze. This entails having the Dpad float above the maze, with its center aligned to the user's position. However, while this suggestion could potentially offer a larger display of the maze by integrating the Dpad into the maze itself and saving screen space, it introduced challenges. The implementation led to the maze appearing 'crowded,' causing layout inconsistencies and potential interference with the user's interaction. Addressing these issues would require additional time for adjustment and refinement, and after careful consideration, we opted to maintain the existing design for a more seamless user experience.

6.3 Considerations

In this subsection, we trace the evolutionary path of our application's implementation, highlighting the journey from its initial development as a web app running on a single machine, to its

deployment on diverse platforms and, finally, the development of a standalone version to enable the use in offline environments.

In our endeavor to enhance the portability of the Task Generator, our initial strategy involved leveraging the capabilities of the web. Given the widespread access to the internet across almost every device nowadays, this approach seemed to be a logical choice.

Our initial iterations focused on making the Task Generator functional on a single machine, on localhost, and later exploring third-party services like Render and Firebase Hosting to extend its accessibility and reach. Our approach of separating backend and frontend functionalities, proved beneficial.

Firstly, it allows for modularity and maintainability, enabling independent updates to backend business logic without impacting the user interface and simplifying maintenance. Secondly, this separation facilitates scalability by allowing the independent scaling of the server and frontend to efficiently handle increased loads and distribute resources based on demand. Thirdly, it promotes reusability, allowing backend business logic to be reused across different frontend applications and facilitating the reuse of frontend components within the same application or across various projects.

One significant drawback of a web application split into backend and frontend is the reliance on internet connectivity. Both components, residing on separate servers, necessitate a stable and active internet connection for the application to function correctly. This can be problematic in scenarios where users have limited or no internet access, leading to potential service disruptions or rendering the application unusable.

The need to have the application working in an offline environment, meant that adjustments were needed, mainly the need to have the backend and frontend logic working simultaneously in the same machine.

Bundling frontend and backend in the same application poses challenges. It hampers scalability, making it difficult to handle growth, and complicates maintenance, risking bugs with updates. Flexibility is limited, hindering independent technology stack changes. Resource allocation issues may impact performance, and security risks increase when both components run on the same machine.

Independently scaling frontend and backend becomes complex, and code reusability suffers. Collaboration among developers is impeded, testing becomes intricate, and deployment coordination is challenging. In essence, while it simplifies certain aspects, bundling introduces hurdles in scalability, maintenance, flexibility, security, resource allocation, code reusability, team collaboration, testing, and deployment. Careful consideration is vital based on project requirements and future plans.

6.4 Limitations

While our work has enabled the transformation of the traditional Task Generator into its digital counterpart, it is important to acknowledge the areas where further refinement is needed.

The transition to a web-based platform brings with it the requirement of a stable internet connection for users to access and utilize the application. This limitation poses challenges in regions with limited connectivity, impacting the availability of cognitive training and rehabilita-

tion tasks. These limitations might affect the application on a more technical level, one example being the timers that could malfunction, impacting the user experience.

Unlike traditional paper-and-pencil tasks, which are accessible without specific devices, our digital application relies on access to a computer or tablet. This device dependency may limit the application's accessibility to a certain user demographic.

The transition from traditional paper-and-pencil tasks to their digital counterparts has shifted the role of fine motor skills. While digital tasks, like the labyrinth, reduce errors related to manual dexterity, this shift poses a nuanced challenge. On one hand, it places a stronger focus on cognitive abilities, allowing for purer cognitive assessments. However, it may obscure certain nuances; motor-related errors can provide valuable insights into cognitive functioning. Thus, the reduced room for motor-related errors offers advantages but also calls for a deeper exploration of its impact on task interpretation and future assessment design.

7 Conclusion

In this paper, we have presented significant modifications to the existing NeuroRehabLab Task Generator, with the primary goal of transitioning it into a tablet-compatible web application. Our approach involved the transformation of the Task Generator into a client-server architecture using contemporary web technologies.

Through the development of a proof of concept, we successfully validated our hypothesis regarding the adaptability of the Task Generator into a client-server model. This transition effectively segregates the client and server functionalities, thereby satisfying one of our initial project requirements - 'The system must abstract the logic from the view'.

The development process and insights gained during this project underscore the importance of adopting modern web technologies and agile development methodologies to address the evolving needs of our target users. While we have achieved a substantial milestone in this work, there remains ample room for further enhancements and refinements.

Looking ahead, our vision is to continue refining the NeuroRehabLab Task Generator, embracing user feedback, and exploring avenues for expanding its capabilities. This application represents not just an innovation but a commitment to cognitive training and rehabilitation, making it a versatile tool for the future.

7.1 Reflections

In retrospect, the process of developing the NeuroRehabLab Task Generator 2.0 has offered valuable insights and opportunities for growth. This project has reinforced the importance of a structured and iterative approach to software development, as well as the importance of effective communication and teamwork.

One key takeaway, is the recognition of the dynamic nature of software development. As the project advanced, we faced many challenges that demanded creativity and ingenuity. One notable challenge was the adaptation of the task generation algorithms to store data instead of printing it onto a PDF. This, coupled with substantial modifications to data structures, introduced considerable complexity into the project.

Another notable challenge we encountered was related to time management. During the development process, we created multiple prototypes, which led to some time being spent on refining these prototypes when it could have been allocated more efficiently towards building the final version.

In addition to the challenges mentioned, a significant milestone in our development journey was the transformation of the React web application into a standalone Electron app. This shift was prompted by the need for offline functionality, requiring a thoughtful transition to a technology that could seamlessly package our application and provide an independent, platform-agnostic solution. The exploration and experimentation with Electron not only presented technical hurdles but also underscored the flexibility and adaptability required in navigating the ever-evolving landscape of software development.

7.2 Future Work

While this new version of the application fulfills some of our requirements, there remains significant room for enhancement and expansion. Notably, a substantial portion of the text within the application remains untranslated or partially translated.

Additionally, integrating a database into the application represents a critical improvement. As mentioned in this thesis, our current data storage mechanism relies on JSON files on the backend. While the current JSON file-based data storage is functional for initial development, it presents scalability challenges, limited data query capabilities, security concerns, and lacks support for real-time collaboration. Transitioning to a database-driven system would address these limitations, providing enhanced scalability, robust querying, better security, real-time collaboration, and automated data backup.

Moreover, the integration of a database into the application could enable the implementation of user sessions, ensuring persistent data across the application. With this capability, data would remain intact even in cases of network connection disruptions or when users navigate away from the page. This enhancement would contribute to a seamless user experience, reinforcing the reliability and continuity of data interactions within the application.

In addition to the aforementioned improvements, another exciting avenue for future development is to offer the task generation functionality as a service. This would involve exposing an API that allows external applications to request and receive fully functional tasks generated by the NeuroRehabLab Task Generator. Such an API would not only benefit researchers and practitioners in the field but also open doors to collaborations and integration with other cognitive training and rehabilitation tools. This expansion of capabilities could further enhance the versatility and impact of the NeuroRehabLab Task Generator in the broader academic and clinical community.

Furthermore, to fulfill the remaining requirements and ensure the optimal usability of the application, it is imperative to conduct comprehensive usability tests with the target audience, specifically patients and clinicians. This phase represents a critical aspect of the project that demands careful attention and investigation. These tests will provide invaluable insights into user preferences and requirements, allowing us to refine and tailor the application to better meet the needs of our target audience.

In summary, we envision several areas for potential enhancements to the NeuroRehabLab Task Generator. These include but are not limited to user interface refinements, expanded translation support, seamless database integration, rigorous performance optimization and usability testing.

References

[1] H. Australia, "Cognitive impairment," May 2021, accessed: 2022-10-26. [Online]. Available: https://www.healthdirect.gov.au/cognitive-impairment

[2] G. Livingston, A. Sommerlad, V. Orgeta, S. G. Costafreda, J. Huntley, D. Ames, C. Ballard, S. Banerjee, A. Burns, J. Cohen-Mansfield, C. Cooper, N. Fox, L. N. Gitlin, R. Howard, H. C. Kales, E. B. Larson, K. Ritchie, K. Rockwood, E. L. Sampson, Q. Samus, L. S. Schneider, G. Selbæk, L. Teri, and N. Mukadam, "Dementia prevention, intervention, and care," *The Lancet*, vol. 390, no. 10113, p. 2673–2734, Dec. 2017.

[3] "Dementia," September 2022, accessed: 2022-10-26. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/dementia

[4] "Neurological disorders affect millions globally," accessed: 2022-10-26. [Online]. Available: https://www.who.int/news/item/27-02-2007-neurological-disorders-affect-millions-globally-who-report

[5] C. Freitas, J. Perez, M. Knobel, J. M. Tormos, L. Oberman, M. Eldaief, S. Bashir, M. Vernet, C. Peña-Gómez, and A. Pascual-Leone, "Changes in cortical plasticity across the lifespan," *Frontiers in Aging Neuroscience*, vol. 3, 2011. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnagi.2011.00005

[6] J. Solana, C. Cáceres, A. García-Molina, E. Opisso, T. Roig, J. M. Tormos, and E. J. Gómez, "Improving brain injury cognitive rehabilitation by personalized telerehabilitation services: Guttmann neuropersonal trainer," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 1, p. 124–131, Jan 2015.

[7] A. L. Faria and S. B. i. Badia, "Development and evaluation of a web-based cognitive task generator for personalized cognitive training: a proof of concept study with stroke patients," in *Proceedings of the 3rd 2015 Workshop on ICTs for improving Patients Rehabilitation Research Techniques*, ser. REHAB '15. New York, NY, USA: Association for Computing Machinery, Oct 2015, p. 1–4. [Online]. Available: https://doi.org/10.1145/2838944.2838945

[8] A. L. Faria, M. S. Pinho, and S. B. i. Badia, "Capturing expert knowledge for the personalization of cognitive rehabilitation: Study combining computational modeling and a participatory design strategy," *JMIR Rehabilitation and Assistive Technologies*, vol. 5, no. 2, p. e10714, Dec 2018, company: JMIR Rehabilitation and Assistive TechnologiesDistributor: JMIR Rehabilitation and Assistive TechnologiesInstitution: JMIR Rehabilitation and Assistive TechnologiesLabel: JMIR Rehabilitation and Assistive Technologiespublisher: JMIR Publications Inc., Toronto, Canada.

[9] E. Irazoki, L. M. Contreras-Somoza, J. M. Toribio-Guzmán, C. Jenaro-Río, H. van der Roest, and M. A. Franco-Martín, "Technologies for cognitive training and cognitive rehabilitation for people with mild cognitive impairment and dementia. a systematic review," *Frontiers in Psychology*, vol. 11, 2020. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpsyg.2020.00648

[10] S. Ge, Z. Zhu, B. Wu, and E. S. McConnell, "Technology-based cognitive training and rehabilitation interventions for individuals with mild cognitive impairment: a systematic review," *BMC Geriatrics*, vol. 18, no. 1, p. 213, Sep 2018.

[11] "Qué es guttmann, neuropersonaltrainer®?" accessed: 2022-12-22. [Online]. Available: https://gnpt.es/que-es-gnpt-2/

[12] J. Solana Sánchez, C. Cáceres Taladriz, E. J. Gómez Aguilera, S. Ferrer Celma, M. Ferre Bergada, and P. Garcia Lopez, "Previrnec a new platform for cognitive telerehabilitation," in *Proceedings of COGNITIVE 2011, The Third International Conference on Advanced Cognitive Technologies and Applications | COGNITIVE 2011, The Third International Conference on Advanced Cognitive Technologies and Applications | 25/09/2011 - 30/09/2011 | Roma, Italia.* Roma, Italia: E.T.S.I. Telecomunicación (UPM), 2011, p. 59–62. [Online]. Available: http://www.iaria.org/

[13] J. Solana, C. Cáceres, A. García-Molina, P. Chausa, E. Opisso, T. Roig-Rovira, E. Menasalvas, J. M. Tormos-Muñoz, and E. J. Gómez, "Intelligent therapy assistant (ita) for cognitive rehabilitation in patients with acquired brain injury," *BMC Medical Informatics and Decision Making*, vol. 14, no. 1, p. 58, Jul. 2014. [Online]. Available: https://doi.org/10.1186/1472-6947-14-58

[14] S. L. Willis, S. L. Tennstedt, M. Marsiske, K. Ball, J. Elias, K. M. Koepke, J. N. Morris, G. W. Rebok, F. W. Unverzagt, A. M. Stoddard, E. Wright, and f. t. ACTIVE Study Group, "Long-term effects of cognitive training on everyday functional outcomes in older adults," *JAMA*, vol. 296, no. 23, p. 2805–2814, Dec 2006.

[15] A. L. Faria, A. Andrade, L. Soares, and S. B. i Badia, "Benefits of virtual reality based cognitive rehabilitation through simulated activities of daily living: a randomized controlled trial with stroke patients," *Journal of NeuroEngineering and Rehabilitation*, vol. 13, no. 1, p. 96, Nov 2016.

[16] A. J. Rotondi, J. Sinkule, G. L. Haas, M. B. Spring, C. M. Litschge, C. E. Newhill, R. Ganguli, and C. M. Anderson, "Designing websites for persons with cognitive deficits: Design and usability of a psychoeducational intervention for persons with severe mental illness," *Psychological Services*, vol. 4, p. 202–224, 2007.

[17] W. W. A. Initiative (WAI), "Accessibility principles." [Online]. Available: https://www.w3.org/WAI/fundamentals/accessibility-principles/

[18] B. Caldwell, M. Cooper, L. G. Reid, and G. Vanderheiden, "Web content accessibility guidelines 2.0," *W3C Recommendation*, vol. 11, 2008.

[19] "Html basics - learn web development | mdn." [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[20] "Css: Cascading style sheets | mdn." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS

[21] "Css basics - learn web development | mdn." [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics

[22] "What is javascript? - learn web development | mdn." [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[23] “Javascript | mdn.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[24] “Javascript basics - learn web development | mdn.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

[25] “Team - react.” [Online]. Available: https://reactjs.org/community/team.html

[26] “React – a javascript library for building user interfaces.” [Online]. Available: https://reactjs.org/

[27] “Tutorial: Intro to react.” [Online]. Available: https://reactjs.org/tutorial/tutorial.html

[28] “Mui: The react component library you always wanted.” [Online]. Available: https://mui.com/

[29] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis,” no. arXiv:1709.08439, Sep 2017, arXiv:1709.08439 [cs]. [Online]. Available: http://arxiv.org/abs/1709.08439

[30] S. Alsaqqa, S. Sawalha, and H. Abdel-Nabi, “Agile software development: Methodologies and trends,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 1111, p. 246–270, Jul 2020.

[31] “How do you leverage functional requirements and use cases to support agile and iterative development?” [Online]. Available: https://www.linkedin.com/advice/0/how-do-you-leverage-functional-requirements

[32] [Online]. Available: https://codesandbox.io/s/g6vuy?file=/src/util.js

[33] [Online]. Available: https://electronjs.org/docs/latest/tutorial/tutorial-first-app

[34] J. Brooke, “Sus: a “quick and dirty’usability,” *Usability evaluation in industry*, vol. 189, no. 3, pp. 189–194, 1996.

8 Appendix

8.1 Appendix A - Diagrams

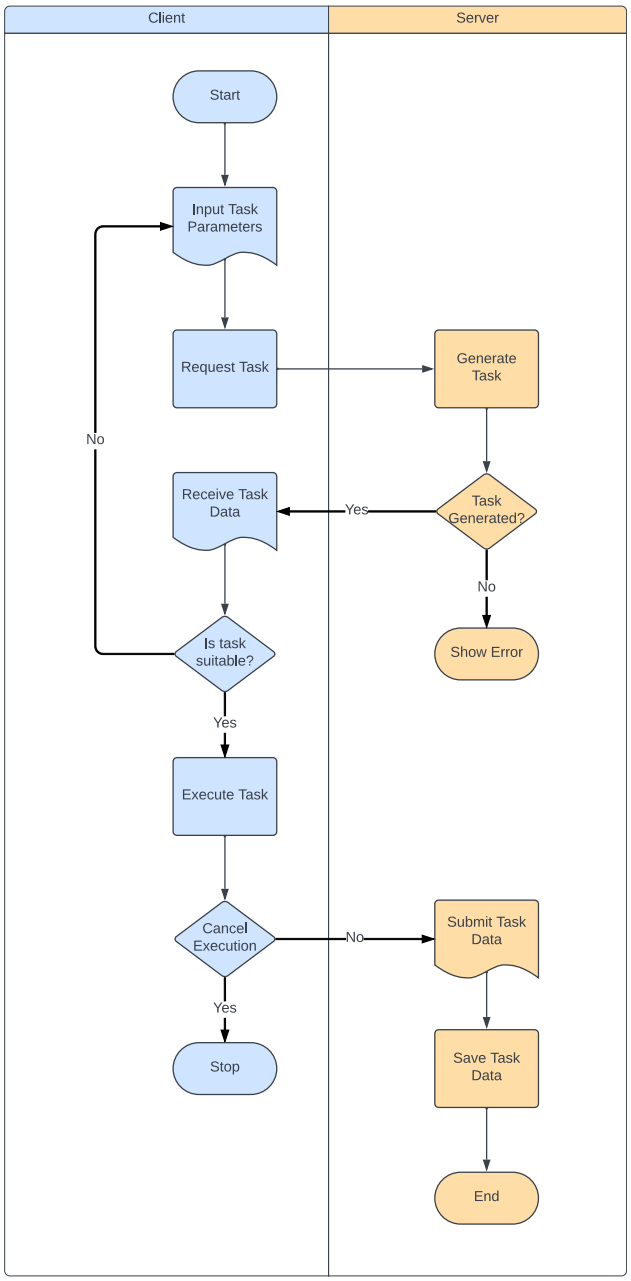


Fig. 36: General Flowchart

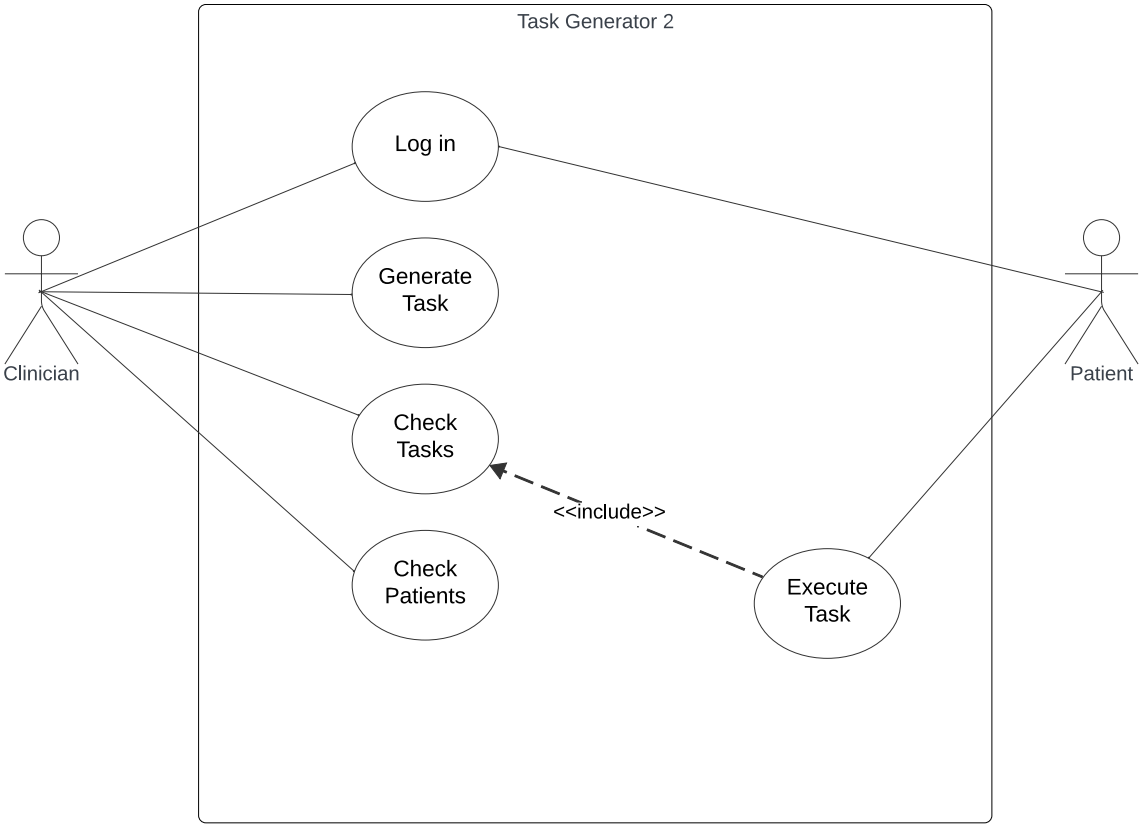


Fig. 37: Use Case Diagram