Nazarbayev University

SEDS

Computer Science

The APRIL Project

Senior Project Final Report

Submitted by:

Olzhas Toleutay

Salauat Kakimzhanov

Yerassyl Segizbayev

Advisor:

Professor Anara Sandygulova

April 25, 2025

Contents

1 Executive Summary

The APRIL project provides a therapy infrastructure designed to support children with ASD through the use of social robots. The project consists of three main components: an online website, an offline web-application, and robots (NAO and Furhat). Legacy systems were re-engineered using Svelte, FastAPI and databases (MongoDB for the online website, and DictDataBase library for the offline application). Robots compatibility issue was solved by integrating a deprecated naoqi library for NAO written onPython 2.7. Our team implemented a unified control interface for NAO and FURHAT, with the database system for children and therapy session data. Online website, a platform that allows parents to view summary information about the progress of their children, was implemented. The aim of this project is to create software that allows specialists to control robots for therapy sessions and evaluate sessions using the offline application, while providing parents with data about their children's progress through the online website with an intuitive UI.

2 Introduction

2.1 Project Overview

The APRIL project is an initiative made to offer therapy for kids who are struggling with ASD (autism spectrum disorder) via social robots. The APRIL project is an infrastructure consisting of three entities:

- **The online website**
- **The offline web-application**
- **The robots (NAO and Furhat)**

The APRIL project's main target audience are special centers that specialize in therapy of ASD among kids. Besides, the software project provides the aforementioned social robots. Predictable and scripted nature of robots are best suited for kids who struggle in socializing with other people, which is a common symptom among kids with ASD. Primary stakeholders, and what they gain from the APRIL project are

- **Autism treatment centers.** - hey receive a modern therapy approach for an ASD treatment using robots, full software and hardware support.
- **Therapists and specialists.** All-in-one software solution for interaction with robots with intuitive UI/UX.
- **Parents and caregivers.** Child's progress tracking system.

- **Researchers**. Gathered data can be used in research to improve robo-therapy sessions in future.

2.2 Project Goals

The April project is an ongoing legacy project that was previously developed by students from the previous study year. The overarching goal of the senior project, viewed from a high-level perspective, is to develop a reliable and maintainable infrastructure that enables specialists to conduct therapy sessions using social robots while providing parents with data about their child's progress through a website with intuitive UI/UX.

The project has to offer following services:

- Robots control interface for specialists. An application should connect to robots, send commands, and gather information about child performance during therapy sessions.
- The database system that manages therapy sessions and gathers data from robots and gathers evaluation feedback from specialists, with the database containing information about the children and session history
- Website for parents that shows data gathered by robots, and feedback provided by specialists

The infrastructure has to follow two main criteria:

- The software should be reliable in terms of low to none bugs and connectivity issues (among every entities including robots)
- The software's source code should be "legacy-friendly" in terms of code-readability, maintainability, frameworks and libraries availability and scalability

To achieve this goal, our team had to research all the necessary libraries and APIs to interact with the robots. Then we studied the previous software solutions and adapted some of them to our system or discarded them by implementing alternative solutions. With the software base ready we implemented additional features requested by the users that were either unavailable or not working properly. Then we designed system architecture that ensures synchronization and scalability of the system.

3 Background and Related Work

Our team did not work with the ReactJS framework that was used in legacy systems before, we had to learn the basics of ReactJS to understand the legacy system. The NAO robot uses the naoqi library that also had to be studied. Moreover, the NAO robot was working on a C++ application that also required investigation to properly integrate it into an offline application. To get a better understanding and general view on the project**,** we visited one of such autism treatment centers and based on the feedback from specialists updated some features in our product. Centers before this application were using three different pieces of software:

1. C++ application, it can connect to NAO and launch certain behaviours (actions). TXT files with NAO categories, actions
2. The offline legacy application on React and FastAPI that can connect to Furhat only. Proto-database system. Single JSON file database of children, TXT files with Furhat categories, actions and category descriptions. Backend was not up to best practices (no Python type-hinting, weak validation, no database ACID compliance, no Database schema files, different code organization). Basic mock child dashboard. Outdated database schema, no skills and evaluation.
3. The online legacy application on React and FastAPI. Registration for parents and center specialists. Basic management of children and parents. Basic management of centers. Viewing children. Backend was not up to best practices (few Python type-hinting and validation, synchronous and blocking API). Outdated database schema, no skills and evaluation.

4 Project Approach

4.1 Offline Web Application's Database Diagram

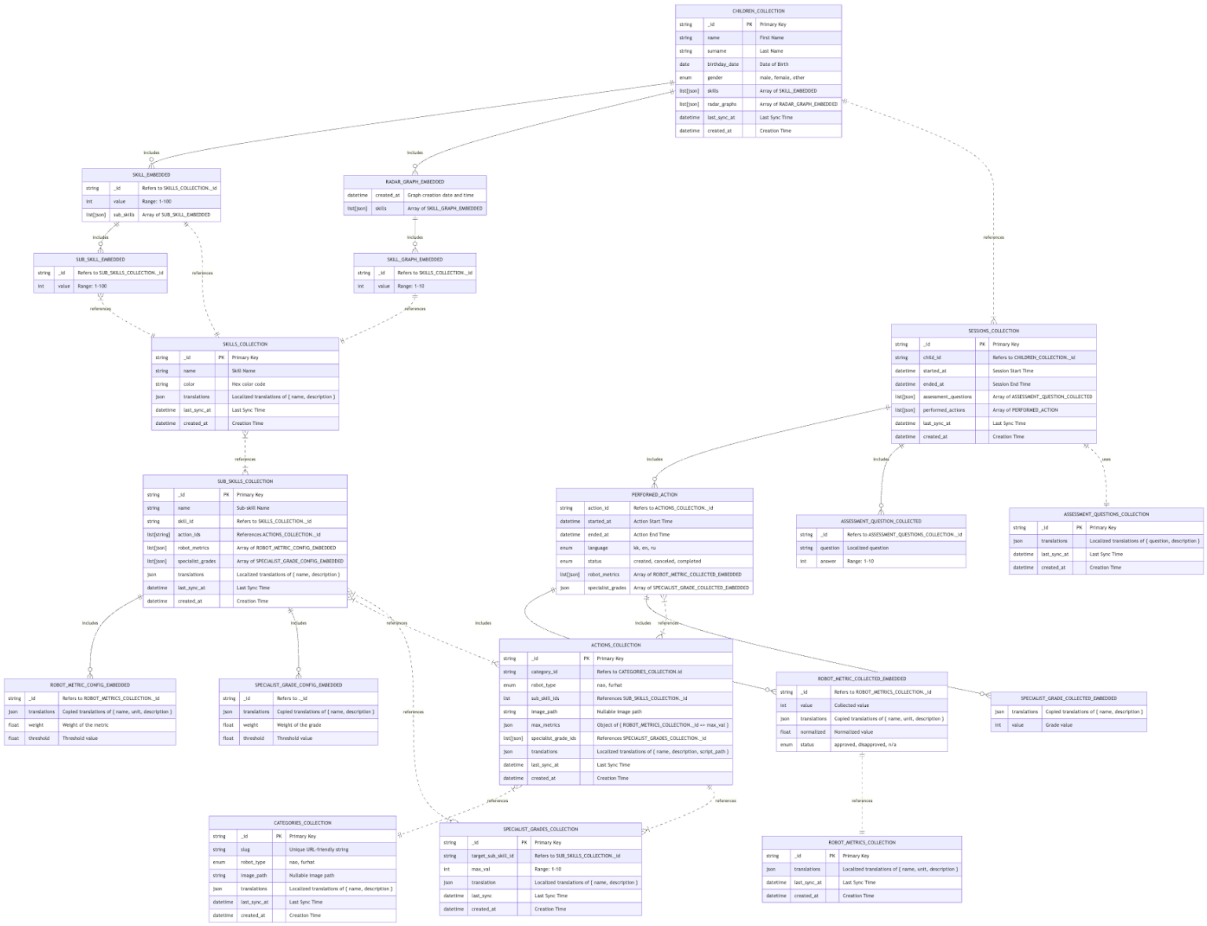

Figure 1: Offline Tables

To solve the problem of progress tracking we designed a NoSQL database that will ensure efficient data transfer between master server and slave nodes. The main entities in the class diagram above are skill, sub_skill, action, robot metrics and specialist grades. The main challenge is to unify all subjective and objective data into one session that will be easily calculated to children skills. Furthermore, the NoSQL database for MongoDB that was chosen by the previous developers team added complexity in overall design. For

example, there are no foreign keys and cascaded updates so the main decision was to store them in one place in order to avoid data duplication. Moreover, multi language support was an additional challenge in designing and implementing it. To resolve many to many relationships between actions, robot metrics, specialist grade, skills and sub_skills, we added bind to action, so each action assesses different metrics and those metrics calculated differently based on the sub_skill it is developing. To ensure proper skill development, a threshold was added to evaluate negative or positive impact of metric on sub_skill. All new metric logic wraps around sub_skills that centers all metrics (subjective and objective). Sub_skills describe weight and threshold of robot metric and specialist grade and grouped around actions, because action can be calculated differently.

Data hierarchy is as follows:

1. Children skill
    - Radar_Graph (snapshot of skill progress)

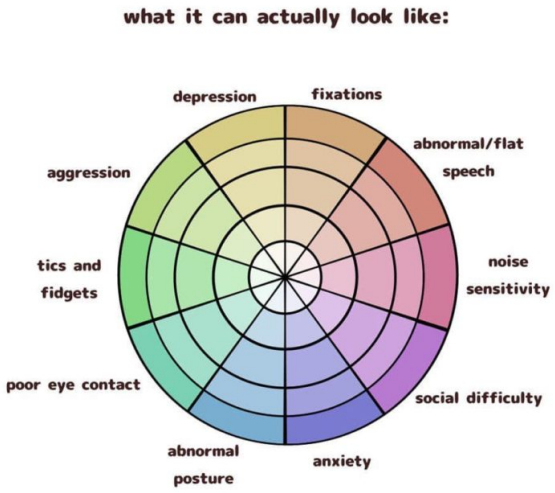

Figure 2: Radar Graph Example

    - Children sub_skill
2. Sessions
    (a) Performed actions
        i. Specialist grade
            A. Evaluated grade name
            B. Grade value

ii. Robot metrics

A. Collected value

B. Normalized value = Collected value / max_metric

C. Status: Approved,Disapproved,N/A

Following data hierarchy shows a bottom-up approach, because all metrics collected from performed action and based on approved robot metrics and specialist grades they contribute to sub_skill that adds up to children skill.

- Calculation formula:

$$subSkillValue = \frac{(robotMetric * threshold * weight) + (specialistGrade * threshold * weight)}{2}$$

- Skills:
  - Aggregate sub-skills:

$$skilValue = skilValue + ln(avg(SubSkills))$$

2.2. Dynamic Action Impact

- Learning Curve:
  - Repeated actions have diminishing returns using:

$$actionimpact = \frac{baseimpact}{\sqrt{actioncount + 1}}$$

  - Example: After 15 executions, an action contributes only 25% of its initial impact.

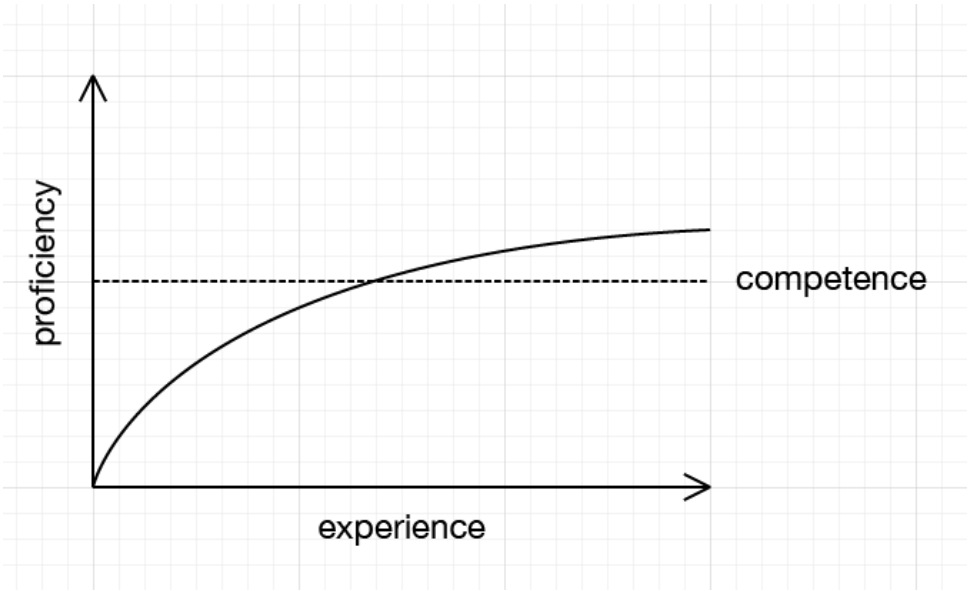

Figure 3: Radar Graph Example

4.2 High-level overview of the system

Every node contains multiple instances of application and designed to be separated by responsibilities as microservices to support scalability in terms of centers offline and users online. This flow eases new script additions via online application. For example, there are red lines within the diagram that show data flow. The data about a script added to the database and new scripts/audio delivered through git CI/CD. After every git update, offline application passess needed files through SSH to nao robots, which eliminate manual work from every step, automating all processes we encourage of new actions development.

1. System architecture
   (a) System components
      - Master Node:
        - Central server with an online application and MongoDB.
        - Manages reference data (‘SKILLS_COLLECTION‘, ‘CENTERS‘, ‘USERS‘).
        - Triggers CI/CD for updates via Git and Kubernetes.
      - Slave Nodes:
        - Local nodes in therapy centers.
        - Include offline apps, a local database (‘LocalDB‘), and robot control (NAO/Furhat).

– Synchronize with the master node via Sync Microservice.

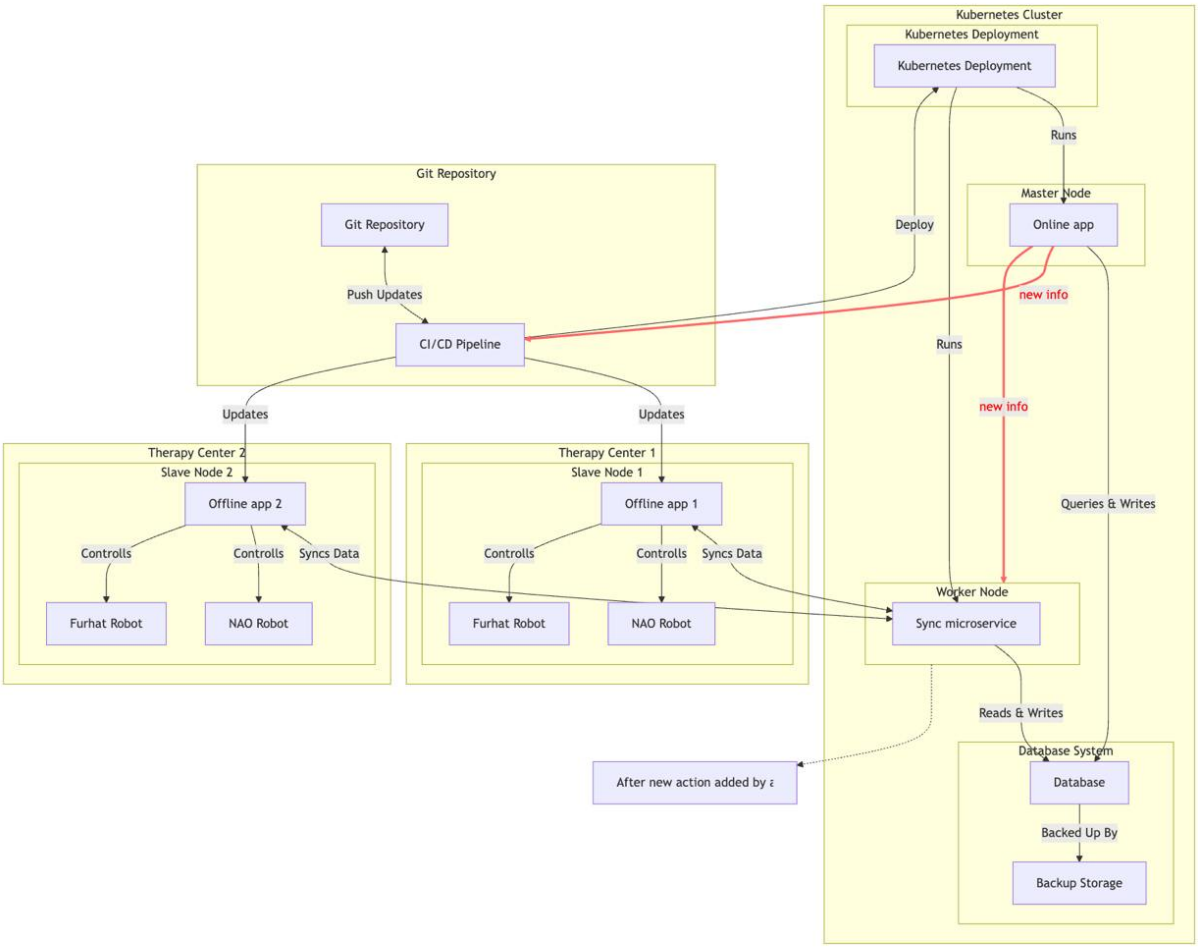

Figure 4: System level architecture

4.3 Online Database Diagram

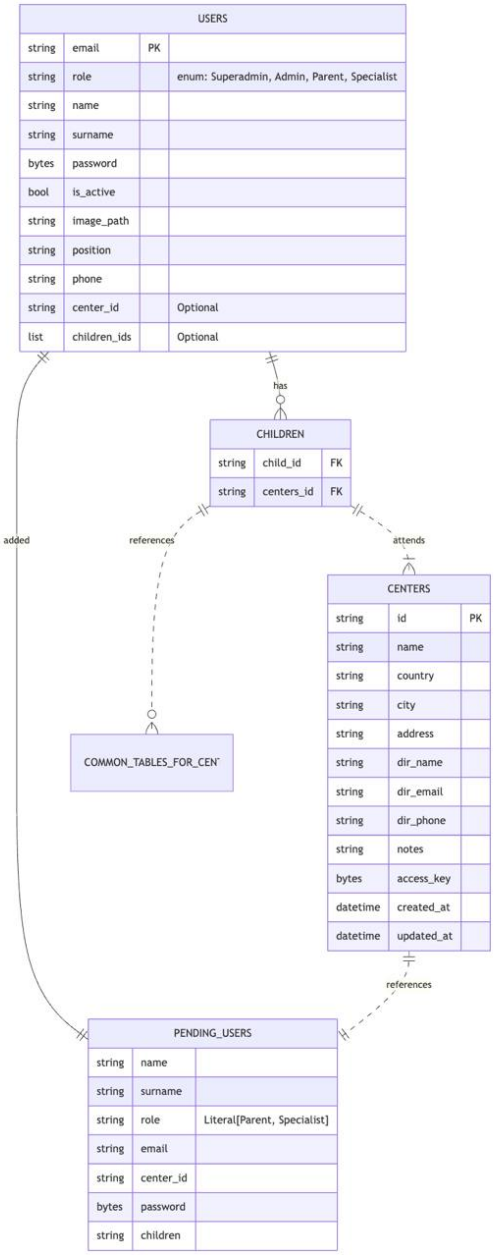

Figure 5: Online Tables

The main logic was centered in offline application logic, so we copied the existing solution and expanded it to support more centers and users. Only H-Case scenario is two parents can have more than one child attending multiple centers in our system. To resolve this problem, we mapped children and centers in different tables that serve as mappers.

4.4 Syncrhonization Flow Diagram

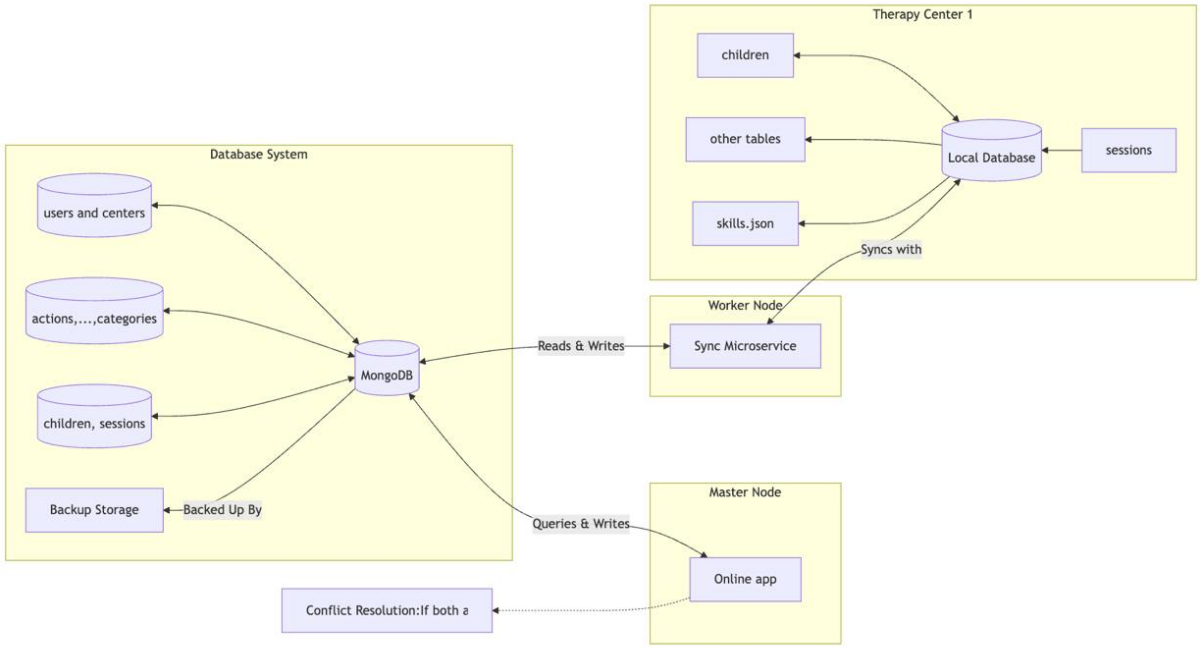

Figure 6: Database Level

The diagram above shows workflow of data between center and online application. Only children and sessions passed upstream to the online app, whereas all updated tables of skills, action and other tables passed down to microservice that convert to json files and are replaced so sync flow offline happens with no overhead on the system. All merge conflicts are solved by in sync microservice.

Asynchronous Model

- Workflow:

  - Changes in the master node propagate via task queues.
  - Slave nodes send data in batches when connectivity is available.
- Advantages:
  - Offline functionality.
  - Reduced network load.

Data Synchronization Process Syncrhonization Stages:

1. Data Collection:
   - Check ‘last_sync_at‘ and ‘updated_at‘ fields.
2. Priority:
   - Sync reference data first (‘SKILLS_COLLECTION‘, ‘CATEGORIES_COLLECTION‘)
   - Then sync dependent data (‘CHILDREN_COLLECTION‘, ‘SESSIONS_COLLECTION‘).
3. Conflict Resolution:
   - Use ‘last_modified‘ timestamps.
   - Admin notification for equal timestamps.

Data Migration Challenges

- Issues:
  - Migrating flat data to a nested structure took 3 weeks.
  - Missing ‘sub_skill_ids‘ in historical data.
- Solution:
  - Python scripts for data transformation
  - Manual verification of records

Integration with NAO/Furhat Robots

- Updates:
  - Robot scripts and configs delivered via CI/CD
  - Testing: automated checks + manual approval.
- Issues:
  - Failed to compile ‘qi‘ framework for Windows.

  - Cause: Outdated library (v4.0.5) and lack of documentation.

- Workaround:
  - Use WSL (Ubuntu) for development.

Conclusions and Recommendations

Key Outcomes

- The system achieves:
  - Sub-skills unifying robot and specialist data.
  - Learning curves preventing metric inflation.
  - Asynchrony enabling offline operation.
- Challenges:
  - Data migration.
  - NAO library compilation.
- Recommendations:
  1. For NAO:
     - Adopt updated ‘qi‘ framework versions.
     - Use Docker for environment isolation.
  2. For syncrhonization:
     - Implement RabbitMQ/Kafka for task queues.
  3. For metrics:
     - Add custom learning curves (logarithmic, exponential).

4.5 Third-party compoents

- Python and JS libraries, including DictDataBase, NAO naoqi library, Furhat API library
- MongoDB
- FastAPI and SvelteKit frameworks
- Assisting software, including GitLab and Apache.

- Robot scripts for NAO and Furhat (scripts that call robot APIs and run files at specific locations on the robots)

5 Project Execution

5.1 What we did

In the first semester we spent time on getting acquainted with the legacy software left from the previous developers. After we studied the React application and backend, we decided to re-engineer the offline application and online website. It was easier for our team to write the website from scratch using a simpler, modern, framework that was familiar to us, than to learn React and update an old software. While choosing the framework we made sure that framework is modern and most certainly will not be left unsupported by developers. Another criteria for framework is low entry threshold for novice developers to minimize entry cost for future developers.

Since the project consisted of two softwares (online website and offline web-application), we had to decide on the stack for two separate development environments. After considering several frameworks, libraries and databases we stuck with the following choice:

- For the online website
  - Svelte framework for frontend
  - FastAPI for backend
  - MongoDB for database
- For the offline web-application
  - Svelte framework for frontend
  - FastAPI for backend
  - DictDataBase Python library.

The previous web application had a primitive database with a single JSON file. We decided to upgrade this database. Launching MongoDB on an offline application would be overkill and might put computational pressure on the simple computers used in centers, while keeping local JSON files as databases is too unreliable. As a middle ground, we chose the DictDataBase library for Python, which had enough complexity to store the database locally and ensure proper data management, but did not require a lot of computational power.

With these decisions in mind, we re-implemented the legacy software while keeping its visual design choices. After adapting and rewriting everything for our own and future developers' convenience, we started to implement new features. Initially, the website only supported connections with Furhat, while NAO robots worked only with a separate C++ application. Our plan was to integrate NAO support into the offline application to create a single control interface for both robots. To do so, the naoqi library for NAO was essential. The naoqi library is a deprecated library that works on Python 2.7, but the rest of the environment, including Furhat, works on Python 3. We had to figure out how to launch Python 2.7 scripts through Python 3, and how to ship Python 2.7 inside an offline web application.

First, we unsuccessfully attempted to update the firmware of NAO so it would support Python 3. Then we decided to stick with Python 2.7 and ship a portable version of Python alongside the application. By including Python 2.7, we could include scripts for NAO in the offline web application, and thus we created a single piece of software that controls both types of robots. The previous C++-based application is no longer required to interact with NAO.

After the integration of NAO, other questions were raised: What metrics data should be collected from robots, how should they be displayed, and how can they be synchronized with the online application? We came up with the system architecture discussed in the previous section to solve these problems.

In the second semester, we had an opportunity to visit Asyl Miras, a center for supporting children with ASD. There, we saw an exemplary software solution designed for specialists that aimed to satisfy the same needs as our solution. The software was a fine-tuned off-the-shelf solution based on a CRM system that supported center management. We were shown the exact processes through which evaluation was collected in that system from a specialist's point of view. Observing the advantages and disadvantages of their solution, we decided to refine our requirements and database schemas to create better software.

As a result, we added new feature targets related to session management, overall session assessments, per-action assessments, and child skill levels in radar graphs. Our system was redesigned to include both an objective and subjective view of child progress—through collected robot measurements for an action, and specialist evaluation for the same action, respectively. It is important to note that we decided to move away from paperwork and other formal processes exemplified in the center, instead focusing on creating solutions that will digitally track children's progress and give centers an intuitive control plane to work with robots.

The offline application theoretically could have been another online website, however, this decision was made to meet the requirements of the target audience. A considerable portion of centers don't have internet access or have unreliable connections. To meet this inconvenience, the application was made offline. However, the offline application expects occasional internet access to synchronize with the online website and/or get updates from it. Updates may contain new scripts for robots, updates for database management, or any other quality-of-life features.

The offline application is implemented in the website form factor for several reasons. Firstly, since this project is a legacy project, the previous application was also a web application—even though the legacy code was rewritten using a modern framework. The second reason is to ensure a better experience for future developers. The application will be handed off to a developer team next academic year. Most computer science students are at least familiar with web development; to increase the likelihood of compatibility for future developers, the application is kept as a web application. Thirdly, a web-based UI/UX provides users with a more recognizable and intuitive experience.

This unconventional form factor for an application led to the problem of delivering the application to end users. The frontend can be easily compiled to static files, but the backend couldn't be compiled due to the need for multiple Python versions (2.7 and 3) and their packages. Initially, we shipped the source code and asked users to install Python and run a script that automatically installed all dependencies and opened a web browser with the offline application. Then we decided to ship Python 3 and Python 2.7 along with the application and wrapped everything inside an installer script. Now, users can simply download the installer and launch it like any other program that can be downloaded from the internet.

5.2 What went wrong

- Automatization of Furhat remote API startup. Historically, Furhat API does not respond to external API calls unless this option is manually turned on in the robot's web interface. Specialists have to log in to the web interface hosted on the robot's IP address, where detailed robot settings are always available. Our idea was to automate this process using webdriver for a virtual browser, i.e. make software that automatically opens the web interface and performs necessary actions. However, this eventually depends on the availability of specific webdivers for specific browsers that would have to be pre-installed or shipped based on target software and OS. Since such a solution is inherently inflexible, we decided to leave this issue for future work.

- We have spent a lot of time throughout the year trying to eliminate our dependence on Python 2.7 that is required by naoqi library because running NAO actions with subprocessesses has blocked us from implementing robot metrics collection scripts. Since we run actions as subprocesses, it is not possible to establish data flow between the backend, the process that calls NAO API to run the action and another script that uses the same API to collect data. Further, the outdated naoqi library uses blocking methods of interaction with the robot, blocking us from collecting the data using a separate script in parallel execution.
- Another issue is related to a third type of social robot, Moxie. As planned, we tried to incorporate the Moxie social robot to the APRIL project. The major inconvenience related to the Furhat and NAO is their price, these robots cost a lot and were designed to be humanoid. Meanwhile kids were interested in robots that had more cartoon-like appearance like Moxie. Since moxie is cheaper and more attractive for kids, it could have been a perfect addition to the project.. However Moxie did not have a publicly available API. To integrate Moxie we had to reverse engineer the robot and try to add custom scripts or custom API. After disassembling and failing an attempt to overwrite scripts (the chip was made by an unpopular supplier and there were no publicly available SDKs) the startup that launched Moxie has unfortunately closed. We decided to cancel integration of Moxie because even with successful reverse engineering and custom scripts we wouldn't be able to purchase new Moxie robots for centres.

5.3 Additional tasks done

- Migration script to convert all data from TXT files to our database schema
- Google Sheet for the Professor and RAs to fill in actual information about available robot activities categories, actions, skills, subskills, specialist grades, robot metrics and session assessment questions.
- Migration script to convert the above data from CSV format to our database schema.
- NAO internal space and naoqi library were researched to design flow to upload new robot scripts directly to robots at the centers within synchronization.
- Full tri-language support (English, Russian, Kazakh)
- Rewritten and expanded online application

5.4 Final offline application screenshots

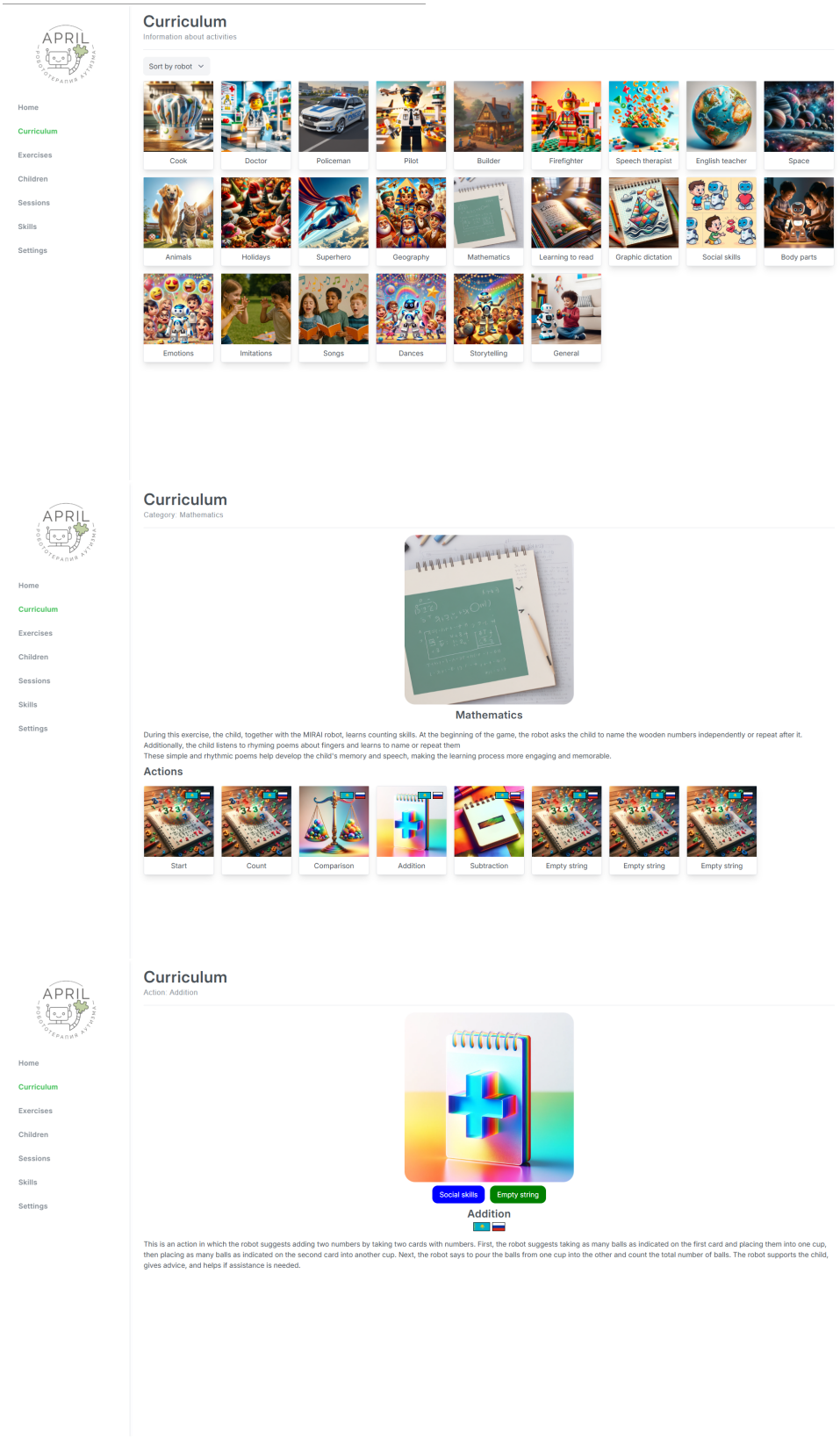

Figure 7: Curriculum module - info about all available robot activities (categories and actions). Names and descriptions of categories and actions. + Skills which actions contribute to.

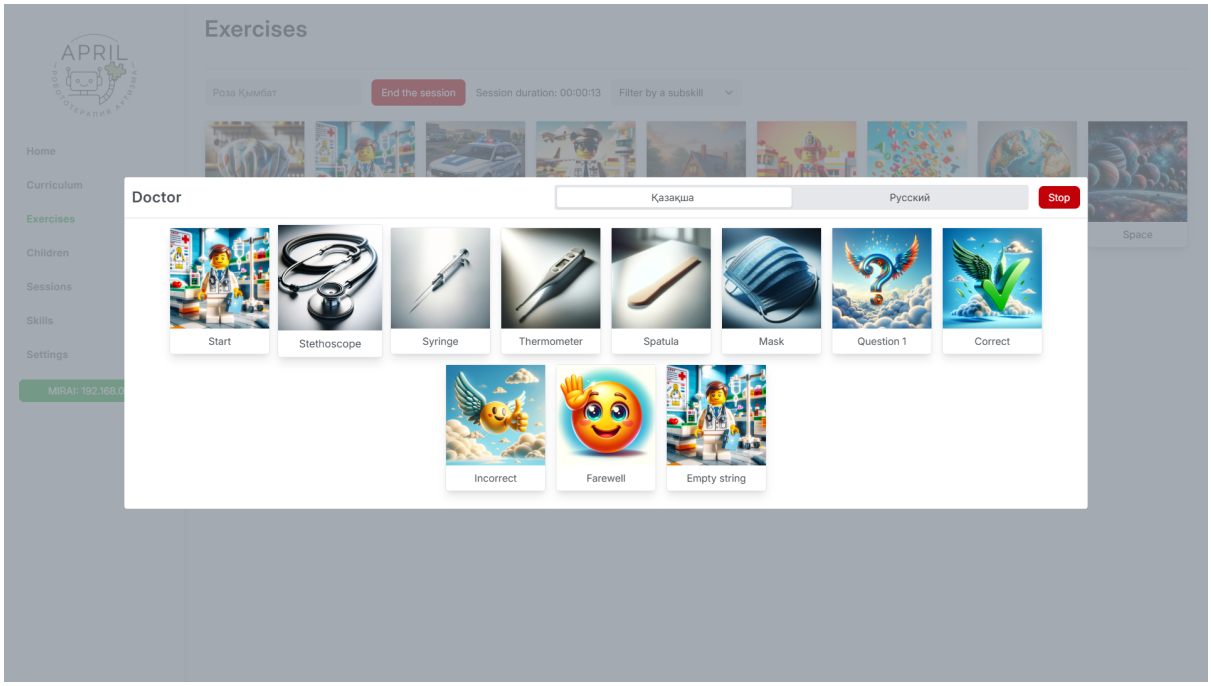

Figure 8: Exercises module. Category form with actions of the selected category (Doctor)

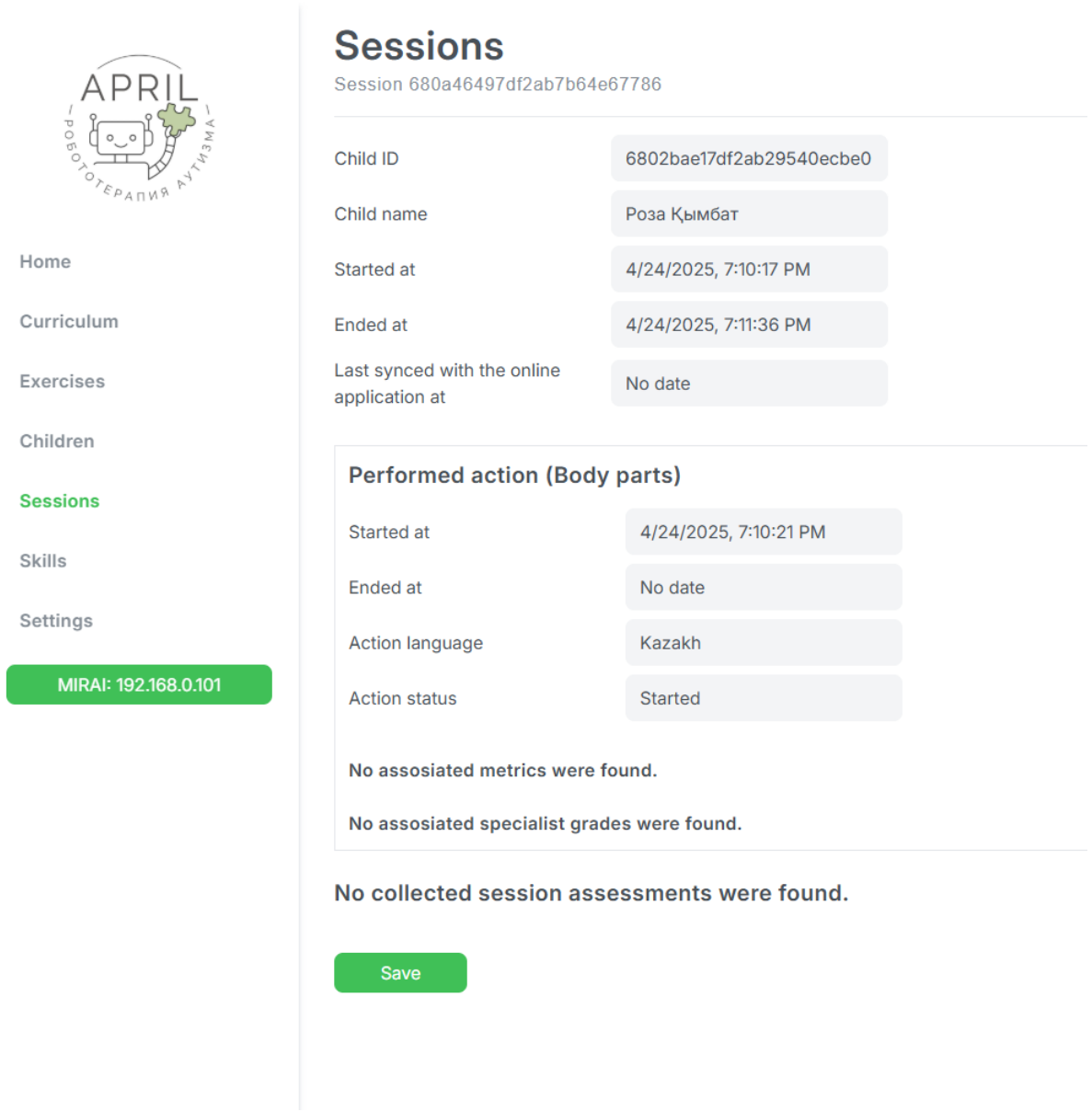

Figure 9: Sessions module. Information about children sessions, performed actions and collected evaluation (specialist grades, robot metrics, session assessments)

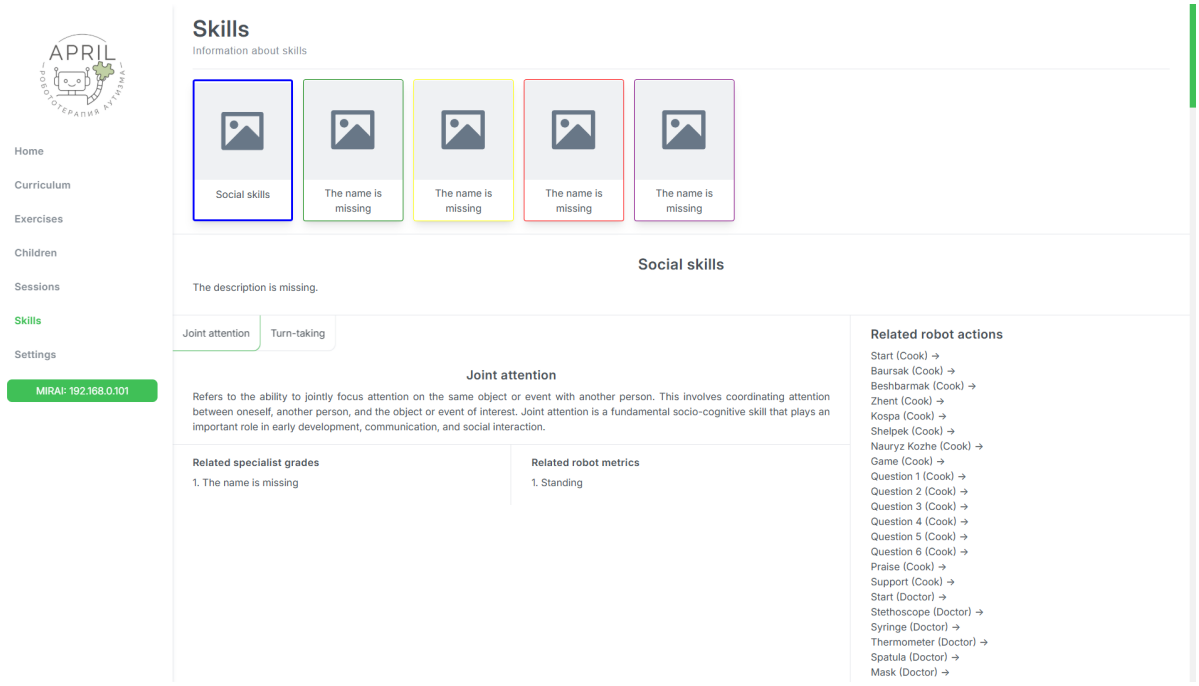

Figure 10: Skills module. information about skills and subskills, as well as evaluation for each subskill and actions that target the subskill.

5.5 Team management and teamwork

Since we are a team of 3 students, we were not significantly struggling with managing the team. We consistently had weekly meetings with our adviser to show our progress and discuss any problems together. We also extensively used Plane.so to keep track of all ongoing, scheduled, completed and canceled tasks. Having regular meetings in the laboratory and using task-management software allowed us to effectively progress through the project. Although we did not officially select a team leader, Olzhas usually took management responsibilities, supporting strong communication with the adviser and between team members to complete their tasks. Since the project turned out to be fairly complex, we mostly divided tasks per student and occasionally provided help if another team member's expertise was necessary. There was no need to have shared tasks outside of general and technical discussions during.

6 Project Evaluation

To evaluate our solution, we contacted 2 of the 7 centers that have been using the legacy system and shared our beta version of the offline application. Based on short interviews and textual feedback, we were able to confirm that our application and installation starts as intended. However, our first contact revealed that, while our application worked on in

the developing and testing environments, the application contained unexpected functional bugs that prevented NAO robots from being stopped. We have addressed this and other raised issues, including the fact that, apparently, we had been using an older version of Furhat's robot scripts. Newer version of the scripts has been provided by research assistants involved in the project, and the latest version of the application was sent for the two centers to try out again. Eventually, we were informed that most of the scripts work, but some require revision, while design and functionality seemed to be sufficient so far. Crucially, both centers did not have problems adapting to the new application after the old ones, meaning that our rewrite and redesign has been successful in ensuring smooth transition. In the near future, we are waiting for the research assistants and our adviser to fill in remaining information about skills and evaluation, which needs to be migrated to the offline application and sent for more feedback from the centers. To ensure that our software will work on a previously unknown machine with low computational power, we dedicated a laptop that has close characteristics to the ones used in centers and cleaned the SSD to ensure that software is not relying on pre-installed libraries, languages, APIs, etc. After building an execution and before sending to the centers software should be launched on that laptop, only if the laptop can run our build, we could send the installation program to the centers.

7 Conclusion and Future Work

7.1 Future work

- Implement the designed synchronization flow.
- Finish test deployment of the online application and gather feedback from the first users. (Online application has been deployed but not set up to be accessible via public internet connection).
- Expand reach of the system by contacting the remaining centers and providing installation executables.
- Complete the final system architecture with multipliable nodes for software versioning and scalability.
- Rewrite NAO and Furhat robot scripts to collect provided robot metrics
- Further expand on online application functionality to allow the administrator to download gathered information about therapeutic sessions and children progress for research purposes.

- Refine the formula for calculating children's progress to account for more complex scenarios.
- Expand on the offline application to include general and personalized children development curricula based on research data, specialist expertise and machine learning algorithms trained on collected information.

7.2 Conclusion

- The APRIL project provides a new approach to the therapy of ASD in kids. Our team's goal was to create a software based on a legacy project that would assist ASD treatment centers and parents in this therapy approach. The APRIL project currently relies on two types of robots (Furhat and NAO). By re-engineering the legacy system our team created an interface in a single application to control and interact with both types of robots. Additionally to this application, our team integrated a database system to store information about children and their performance during therapy sessions. Online website for parents and specialists to keep track of progress of their children was developed. For smooth interaction and information exchanging among three entities (online website, offline application, robots) our team developed a system architecture design to ensure that robots and offline applications will receive an update and data from offline application is synchronized. Feedback from our advisor and specialists from the center helped us to keep track of important features, ensuring the best experience for the end user. For the future developers, throughout the whole project we attempted to keep the system clean, easily maintainable and expandable to minimize the entry cost.

[1] [2] [3]

References

[1] Anara Sandygulova, Aida Amirova, Zhansaule Telisheva, Aida Zhanatkyzy, and Nazerke Rakhymbayeva. Individual differences of children with autism in robot-assisted autism therapy. *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, page 43–52, Mar 2022.

[2] Zhansaule Telisheva, Aida Amirova, Nazerke Rakhymbayeva, Aida Zhanatkyzy, and Anara Sandygulova. The quantitative case-by-case analyses of the socio-emotional outcomes of children with asd in robot-assisted autism therapy. *Multimodal Technologies and Interaction*, 6(6):46, Jun 2022.

[3] Aida Zhanatkyzy, Zhansaule Telisheva, Aida Amirova, Nazerke Rakhymbayeva, and Anara Sandygulova. Multi-purposeful activities for robot-assisted autism therapy. *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, page 34–43, Mar 2023.