EPFL

CHILI
computer-human interaction
in learning and instruction

LEARN
Center for Learning Sciences

Graphical user interface for social robotics

Author:
Julien Jordan

Supervisor:
Daniel Carnieto Tozadore

Professor:
Pierre Dillenbourg

June 9, 2023

Contents

1 Introduction

1.1 Context

The main idea behind this projects stems from a discussion we had with Daniel Carnieto Tozadore , our supervisor in the context of designing and creating a user interface to greatly increase productivity and quality of life working with the [4] QTrobot both as a researcher and a professor. The target users for such an interface are people from the iReCheck team which could greatly profit from the enhanced workflow this project could provide.

1.2 Motivation

This project grants us a great opportunity to gain experience in designing and implementing a software solution from scratch that could help the iReCheck team members by making them save time, enhance their user experience working with the robot and centralise much needed features so they do not have to do manual repetitive work in some of their tasks they are involved in. We also followed the CS213 course last year that was given by professor Pierre Dillenbourg where we learned valuable concepts that are useful in the making of a good quality user interface such as how to approach user experience, making models about the final designs before implementations as well as being very aware about the quality and usability of our interface by working closely to the target users. We will present a convincing proof of concept that there is a lot to gain from this interface in the context of working with the QTrobot from the iReCheck people point of view.

1.3 Objectives

The goal of this project is to engage ourselves into designing and providing a first implementation of the discussed concepts and features so that we can present a convincing Proof of Concept that highlights the the feasibility and interests in having this interface for working with the QTrobot as well as showing present limitations with the current approach so that there is a clear path and and interest into putting effort and resources into a future work where a full and complete implementation would be made.

2 Methodology

In this section we will detail each phase of the project and highlight our approach working through the requirements of our project.
You will be able to find encountered challenges and problems in the dedicated Challenges section.

2.1 Preliminary work

After the supervisor pitched us the project, we felt we had to do some research before the semester began. Indeed this project is about a graphical interface made from scratch which means there wasn't any strict set usage of specific languages and technologies so the initial part of the project, among the first design ideas, was to actually figure out which technologies and languages we would use for our implementation.

Nevertheless, one thing that was highlighted during the initial discussion that started this project was that a web application would be the most viable option and we also agreed about this route for multiple reasons :

1. We wanted the interface to not require users to install anything on their machine.
2. We need the interface to be independent of the user's system.
3. It was made known that the QTrobot runs on an Ubuntu system which means it can easily host a web-app.
4. Web-app development is quite flexible and has many resources.

It quickly became clear to us that we wanted to explore the option of hosting our interface inside QTrobot by making a web-app hosted on it so that when connected to the QTrobot a user would then have access to this interface and interact from there to do the work he needs to do. We then had to find out, for the most part, which would be the main language used to write such a web-app.

Luckily we had the chance to talk about it with another fellow student we know that had a great deal of experience in web-app development and actual professional experience. The discussion we had with him was very valuable in gaining suggestions about which language we could use for our end. We also did research over the web, talked to other students that have experiences in web development and even asked [7] chatGPT to give us some suggestions.

Some options quickly became clear as most suggestions we received (even from AI) formed a consensus that we felt confident about as it was based on the advice of few people with some years worth of experience behind them in this field of development.

2.2 Technical details

Based on our preliminary research work and suggestions we received we decided to chose [5] React, a powerful javascript framework made for front-end oriented development. The motivation behind this choice comes from the formed consensus we found and also that despite having no experience developing in React before, it has a lot of resources online and is somewhat easy to get into.

Component library : it did not require a lot of research to stumble upon [6] MUI components library for React and the reason we chose to use it was firstly due to how clean looking and responsive their components were, which plays a great part in UX and the fact that their library is used by companies such as Spotify, Amazon, Nasa, Netflix, Unity, ShutterStock and more. A library being used by such an impressive list of companies made us confident it was a well supported and interesting library for our project, especially since the license was fitting for this project.

Web-app hosting : to be able to make our web-app usable when connecting to the QTrobot we needed to make its Ubuntu system behave like a web server and to do so we needed to look into the requirements for a React web-app and decided to use these technologies :

1. Node.js : it is an open-source, cross-platform JavaScript runtime environment built on Chrome's V8 JavaScript engine. It allows you to execute JavaScript code on the server-side, enabling server-side scripting and building scalable network applications. Installing it on the QTrobot Ubuntu system is necessary.

2. Nginx (pronounced "engine-x") : it is a popular open-source web server that we chose to use based on the recommendations we gathered about React web-app development. This is the main process that is responsible of hosting and serving our web interface when HTTP requests are made to the host.

3. Vite : it is a build tool and development server. There are few other options, this one has been chosen based on suggestions we received and the fact that it is able to create and deploy a basic React app that proved to be very useful to get into the development and learn the basics before we could implement more complex things. It is also powerful as a build tool and allows a development mode that locally hosts the web-app with a quick-reload function on file save that makes the development experience smoother.

Package Management : Very similarly to the way we can install new software and packages on Unix systems, downloading and installing the dependencies for our web-app would require us to use a package manager. A quick research shows we can either use Yarn or NPM (Node Package Manager). Both are used in a very similar way and we chose Yarn based on recommendations and the fact it has been introduced as an alternative to NPM with a focus on performance.

IDE : we went with Jetbrain's Webstorm IDE mainly because we are already very used to working with Jetbrain's IDEs in other languages and Webstorm just so happen to be their IDE specifically made for web development which pairs well working with React.

Database : our application requires to store data to get rid of manual data handling in google sheet and instead rely on a built in database manager. In another project at the time we used one of industry standard's solution which is Google's [3] Firebase. The

reason we chose this solution is mainly due to the fact we are already working with it on another project at the time so we already had a head-start in understanding the technology and it has the advantage of real-time database which allows us to display live-updates within our application, enhancing UX, as well as having an authentication system which would allow us to create accounts to use the interface.

2.3 High-level design description

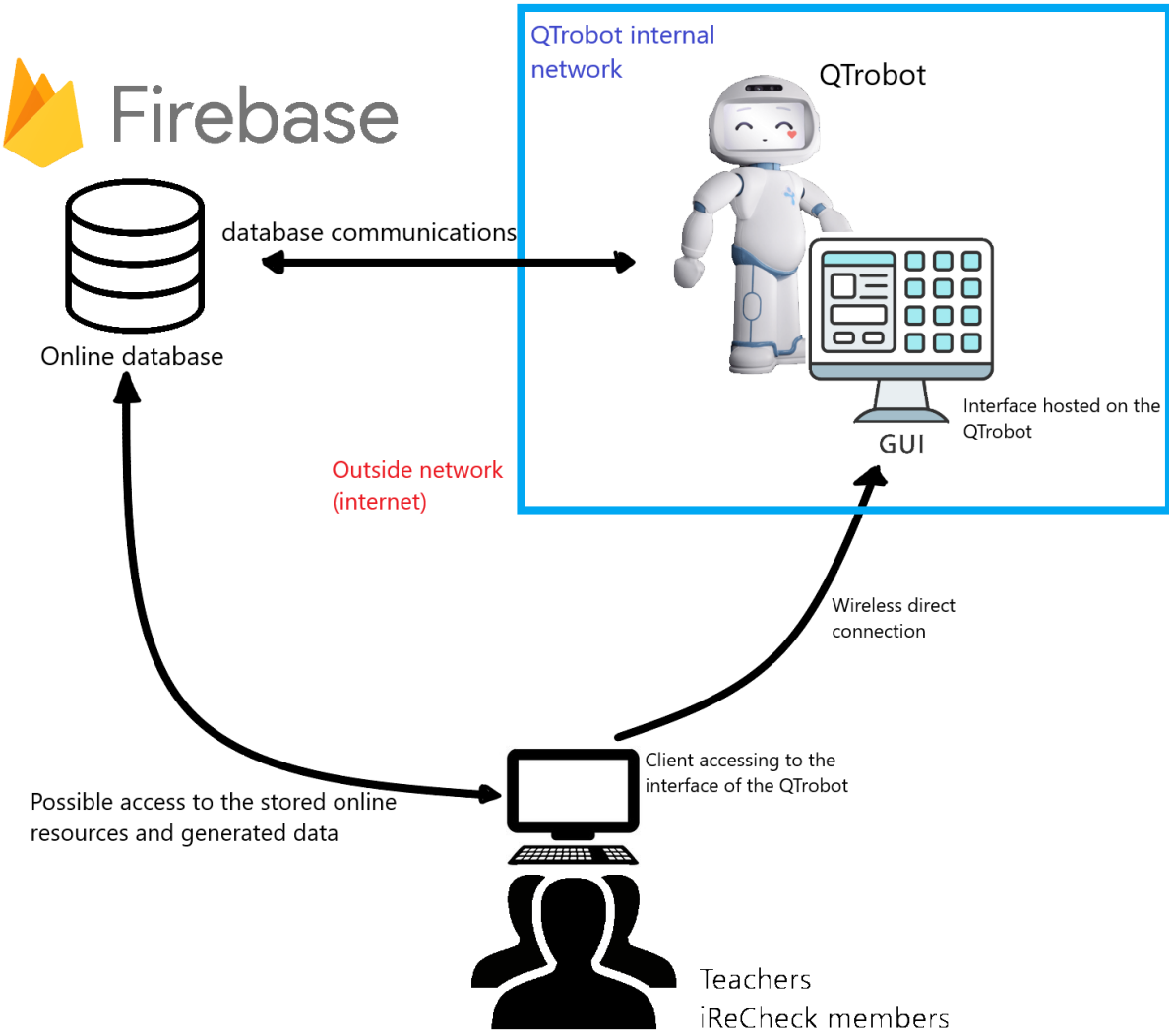

Figure 1: High level diagram of the interaction

The idea behind the implementation is to make the QTrobot function as a web-server to host the web interface. Users will connect as usual wirelessly to the QTrobot and will be able to reach the interface from there due to being on the same internal network within the robot. Using this interface the users are able to interact with the QTrobot , particularly researchers working in the iReCheck project by letting them manage data, generate and run commands to generate CSV results, process and visualise them while making all of the data available from online so users do not need to have a direct access to the robot to retrieve already existing files. The web-interface also has some back-end so that it can run system functions and communicate with the online database.

2.4 Project phases

In the next subsections we will provide details about our approach throughout the different phases of our project.

2.4.1 First phase : Main features and problem evaluation

During this phase of the project we had to engage in discussions with our supervisor but also other users to define as clearly as possible what would be the core features of the interface. During the first few meetings with him, we were able to ask about and observe his workflow when working with the QTrobot and it clearly indicated where we could gain time by streamlining the actions a typical iReCheck member would do working with QTrobot .
We noted some key details about the observed workflow :

1. Children data handling (and more) was done by hand in google sheet or similar, which is tedious.

2. To start activities and generate data, ROS commands needs to be written and launched, it is not particularly practical to have to lookup data elsewhere to call the correct command arguments.

3. The generated data from these commands, precisely .csv files, are processed manually and used through tools other members made available for data visualisation which is also something that could be avoided with an interface.

Needless to say, these tasks are somewhat laborious and can profit a lot from using a well tailored interface instead. Few features quickly rise from these observations which are first and foremost the creation of a database management system so that it is not necessary to manage a sheet with the data by hand which would already prove to be a good way to gain time.

The second main feature is about the usage of ROS commands. They are always of a similar format where only the arguments fed to the command changes based on the child we want to run the data from which means it is possible to create an interface that displays the existing children of the database and lets the user select them as well as select other options such as the context of the QTrobot activity to run and command types. Doing so would generate a command in an editable text field where the user can either edit this command if further changes are needed, copy it or simply run it. This editable command field would act like a terminal running directly on the QTrobot 's system.

The 3rd main feature is about the data visualization. Indeed it would also be nice to have the possibility to generate graphs based on user input and the generated .csv from the 2nd feature so that a user can view the data without having to play too much, if at all, with different graph / plot generation scripts and software and just go straight to

data-visualization.

These ideas and potential solutions were brainstormed through these meetings as a part of collaborative work: on one end we are trying to gather feedback and data about how users are interacting with their work to understand what are their needs and on the other hand we have to make proposals and be creative about potential solutions solving some of the worst aspects of the current workflow.

Some requirements were also made clearer by our supervisor during these first meetings. We have two type of users : the researchers (developers) and professors working with the children. There is therefore a need to separate the privilege of these users as we do not want a professor to do administrative database management or access the ROS command developer page. It was also made clear that there are data privacy concerns : indeed we do not want a professor to access the data of children that are not in their classes.

Knowing this, we proposed that the data management feature would also include user account management for the QTrobot interface and have a straightforward login system to access it. When concluding with this phase of the project, it was asked by Daniel that we make a simple demo interface about the 2nd feature, the command generation part.

2.4.2 Second phase : Demo, feedback and layout design

After this first phase we started a bit of development in React to create this small demo about the ROS command generation feature, as our supervisor asked, and we were able to present it during a meeting which was a very important moment for the project's development as it was the first we could present our vision of the discussed features and see if it corresponded to what our supervisor, part of the target users, was expecting. This is a crucial part of the design and development of an interface as it's an iterative process with lots of communications and collaboration involved. We were pleased to have a user's approval of the demo and it allowed to have a more specific idea of how the interface could look like which is valuable feedback for us to progress in this project. We were then presented with the great opportunity to assist a meeting with other iReCheck members and present them briefly our semester project and gather more feedback.

This meeting took place Friday the 3rd of March at the CHILI Lab where we were able to have a better understanding of what iReCheck research was about which could prove useful for the interface direction. We were able to present our semester project and more specifically ask for feedback, particularly about features they would like to see about the web-app we're making. The meeting room was enthusiastic and participated well in giving out ideas which was extremely valuable to us to figure out user needs for our application.

Here's a list of the relevant notes we took from the feedback the iReCheck people have given us :

1. Different views between the two user roles : A professor would see a more simple, straightforward version of the application where a user with developer privilege would see the full application with the advanced features that are specifically made for them so that we ensure ease of use for the professors.

2. Request that the data generated from the QTrobot would be automatically uploaded online and allow the user to not only download the data from using the application but also remove the necessity of accessing the robot to fetch already generated data.

3. A professor requested that it would be nice to make logging in to the application as simple as possible, we want as little steps as possible before accessing the interface.

4. Using this interface as a "hub" that lets you reach other activities the QTrobot has. E.g. the Story Telling activity could be accessed / started by clicking a button from the home page of the interface.

5. Data visualization within the interface, which was already something we discussed before with Daniel, was a popular request.

After this meeting we had multiple things we could tackle : Discussing with our supervisor which fields do we want for our database entities so that we could design a database structure and create a first draft of the app's layout now that we have a better idea of the main views it has.

For the next meetings we did a drawing of the app's layout to represent how we view how each page's layout would look like and there would be exactly 3 pages representative of the core features we defined :

1. Data and user management page.

2. main developer page to generate ROS commands and run them.

3. Data visualization page to visualize the generated data.

For the database structure, we already had some ideas and inspired ourselves as much as possible from what we've seen in the sheet we observed our supervisor using but we could not find a very specific consensus about the final structure. Instead it was asked that the solution provided would be flexible so that it is easy to add (or even remove) fields of the database entities so that it could be changed at a later time. At first this request was seen as an issue until it was clarified with our supervisor that the database should be online rather than hosted on the QTrobot and we found which solution to use which happens to be very flexible towards field changes when data already exists.

2.4.3 Third phase : design and visuals

At this point we had to work on the visual end already and a good approach to design an interface is to actually do a model of the app rather than a full implementation before getting feedback about the visual design. The reasoning we learned during the course is that it's a lot faster to create a model with high visual fidelity of the end product without the big time requirement of coding the whole application and therefore get feedback and make changes before we're too deep in the development. Otherwise these changes, especially drastic ones, would make use lose valuable time whereas with this approach we can already agree on how the application looks before implementing it.

To do so we used [2] Figma which allows us to create such a model. We used this as based after doing some research it seemed to be a very popular platform to use for this specific use case and we also noticed that the MUI also made their components available, at a fee, for Figma. The cost being somewhat low we decided it was worth buying so that we could create a model of our application on Figma by using the exact same components we are going to use during implementation, ensuring a high fidelity look.

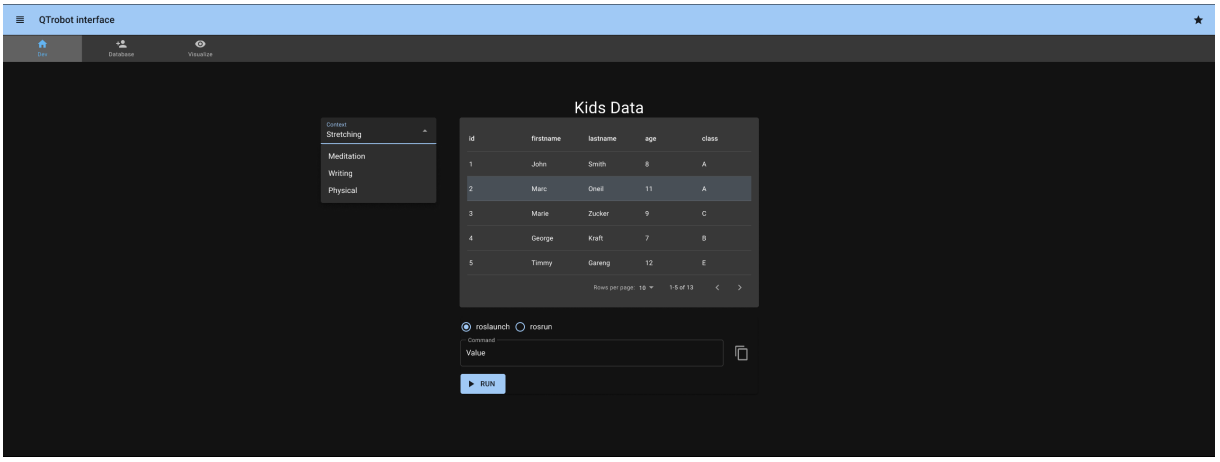

Figure 2: Figma model of the dev page

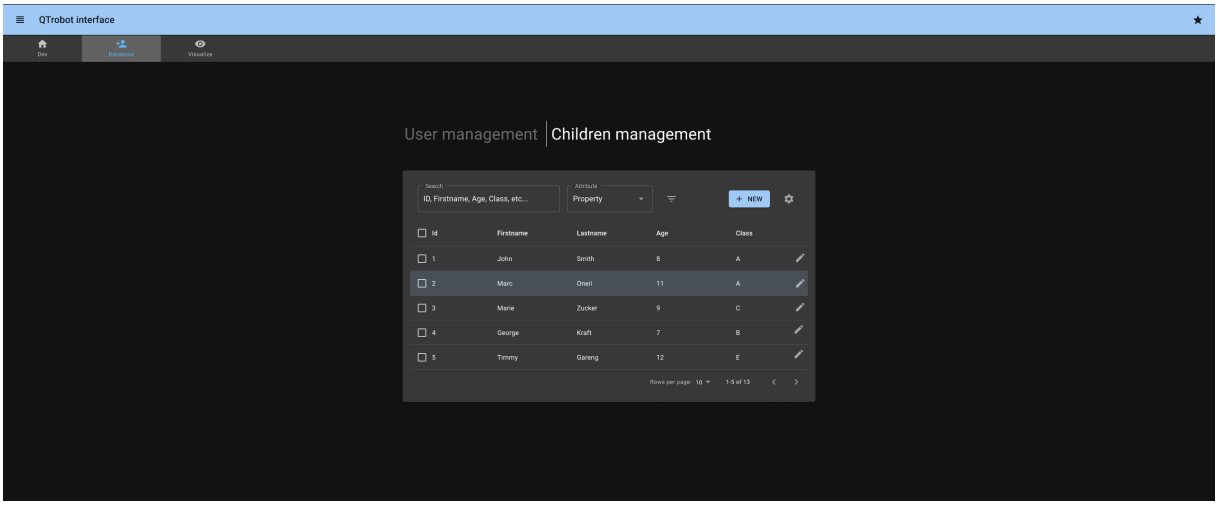

Figure 3: Figma model of the database management page

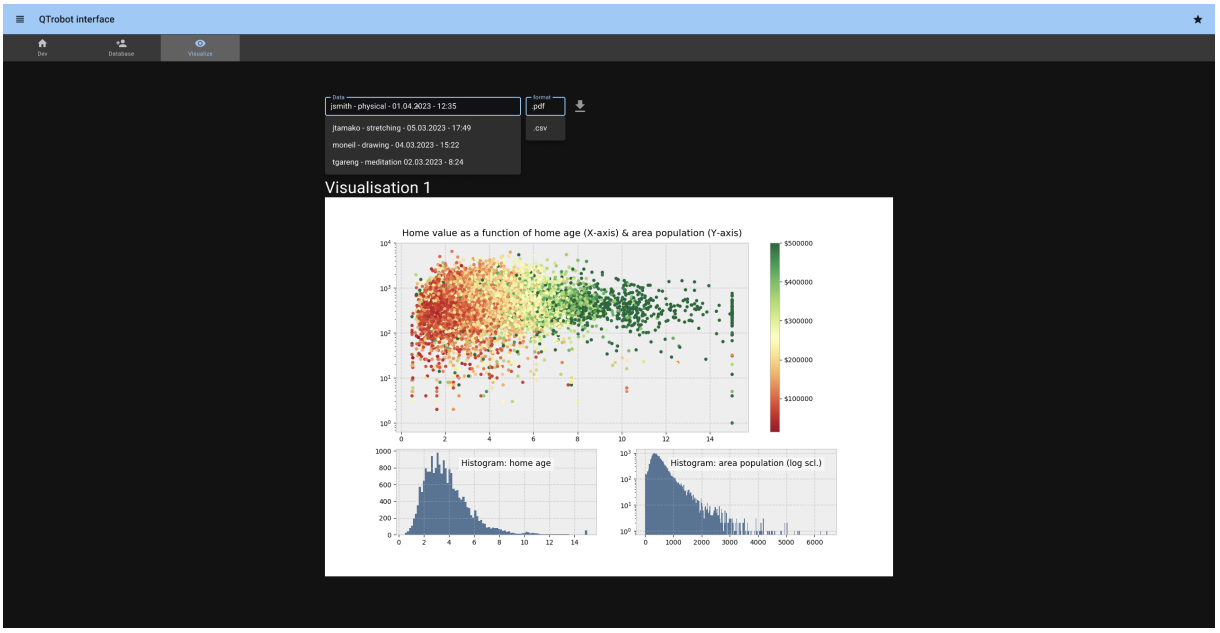

Figure 4: Figma model of the data visualisation page

Another great advantage of using Figma and also something we learned during the CS213 course was that it is also possible to create interactive models so that we could use it to get feedback of users trying out parts of the application without having anything real implemented yet. We took advantage of this during the meeting with our supervisor where we could present this model and asked him to do a few tasks on the interface so that we could observe if it was intuitive to him where certain things were and on which elements he should click to do the actions he wanted.

We were then able to reach a good satisfaction level and felt confident we could proceed with the next step : development.

2.4.4 Fourth phase : development and testing

This is the last and biggest phase of the project as it has the most workload. We now have enough ideas about the project so that we need to make progress in writing code for this interface and implement discussed features.

There are two sides of the work during this phase. One side being working on the code itself and making it follow our designs as well as implement convincing enough features that follows as best as possible the required behavior and the second side being testing our implementations on the QTrobot to make sure our implementations work as intended on the machine it will be running on long term.

The latter has proved to be more problematic unfortunately and during this phase we met many hardships we had to overcome. The reason the testing part has been an issue is due to the fact that the QTrobot had very low to no availability by the end of the semester which does not align well with the fact that actual development was the last phase of this project and therefore we could only have decent enough codebase to try on the robot at that time.

Of course this is a potential issue that we took into account before by setting up a Ubuntu virtual machine with VirtualBox to imistate the QTrobot operating system. Testing and making things work on the VM or QTrobot was more centered about setting up servers and configuration files so that the application could be hosted and be reachable which required to learn and gain knowledge on a vastly different domain than the first side : React web-app development. Needless to say, lacking experience in both sides of the work this phases asks of us, research, documentation reading and smart usage of tools to learn played a big role in the progress and the total time spent.

Our approach for this phase was to start with the custom database management system as some other core features of the application were dependent on children data being present and usable. After which we could follow with the other two main features being the dev page where we want users easily generate and run commands and the data visualisation page.

After this phase we would end with our final proof of concept, implementing a solid base for this interface and already some features able to demonstrate feasibility and interest for a future work about a complete implementation.

3 Challenges

Here is a description of the most notable problems we encountered during this project and how we adapted to them.

3.1 Learning new things from scratch

The first challenge that seemed evident was the fact we had never written code in React before and we had to learn fast if we wanted to have a chance at producing a decent enough proof of concept in time for this semester project. To learn as fast as possible while working on the code already, we read official documentations, used extensively resources available on platforms such as [1] StackOverflow but also using one of the most recent and powerful tools in the industry : AI.
AI assited learning proves to be very interesting in this context with the availability of [7] chatGPT AI models. Typically we used it as an interactive provider of documentation, examples and explanations to greatly accelerate learning and understanding new technologies and concepts. We do not rely on chatGPT to write our code, rather, we use it as an assistance tool to learn during our work. A paper titled [2] *"Exploring the use of chatGPT as a tool for learning and assessment in undergraduate computer science curriculum: Opportunities and challenges"* highlights a performance increase in using such a tool in this context but very interestingly shows the limitations when relied upon to do our work rather than simply help with information due to the AI inaccuracy. A great video titled [3] *"How to learn to code FAST using ChatGPT"* made on this subject by Tina Huang who has a computer science degree and previously worked at Meta as a data scientist explains very well how powerful of a tool chatGPT and using AI to drastically increase learning speed can be once the user understands the art of using such a tool for this purpose.

3.2 Setting up a web server and deploy a web-app to it for the first time

We had never done this before and it proved to be a problem as setting things so that the QTrobot Ubuntu system operates as a web-server for our web-app was nearly entirely about understanding which technologies were required and how to configure them to our needs. Something we did that proved to be very helpful and valuable was to get in touch with a student with prior experience in this domain that was able to make suggestions to

[1] StackOverflow's platform for questions and answers, ref: [8]

[2] Exploring the use of chatgpt as a tool for learning and assessment in undergraduate computer science curriculum: Opportunities and challenges. Ref: [1]

[3] Tina Huang on using chatGPT as a powerful learning tool in software development context. Ref: [9]

us which helped during on research reach a concensus about which technologies we chose to use and how to set them up to achieve our goal. We were able to overcome this after a lot of documentation reading, some AI suggestions and looking up configuration examples online.

3.3 Testing on the robot

Due to the fact there is only one QTrobot we can use and it's not going to be available at times, it was seen early on as a potential problem which we overcome by setting up and using a virtual machine with VirtualBox with Ubuntu to have an environment as identical as possible of the one of the robot. Accessing the web-server on the VM would only be made from the same local network as to immitate the internal network of the robot we access by connecting to it wirelessly.

3.4 QTrobot limitations and found issues

During testing we stumbled upon a consequent roadblock : after a manufacturer's update, we could not have internet through the robot when we were connecting to it by wireless. This is a problem as it indeed prevents a client connecting to the interface from making any outside requests, breaking Firebase communications for example, rendering the app not usable without tweaking the internal network of the QTrobot . We could not find a solution for this just yet but we are confident after testing that it is due to how the internal network and eventual firewall is setup and restoring the client's connection to the internet should solve this problem.

3.5 Project requirements evolving

This project heavily relies on discussion with users and the meetings with our supervisor have an iterative nature to it in a way that the further we were into development, the clearer some ideas for features and requirements became. Some changed from the initial idea which is absolutely expected in software development and our approach to this was to stay open but be clear about limitations and more importantly priorities of the key features we had to implement so that we could meet the best compromise between flexibility during development while keeping a good enough pace to implement this proof of concept on time.

3.6 Time management and the database feature

During this project we greatly underestimated how big of a feature the database management system would be in terms of time spent implementing it. This caused time management issues to meet deadlines for other parts of the project. We believed staying transparent about it and explaining well the state of our progress in the meetings were key to have a constructive discussion about the project state, the expectations we have about it and more specifically the priorities we needed to have in order to spend time working in a more efficient way.

3.7 Firebase limitations

Implementing the authentication system was somewhat tricky as we discovered that if we wanted to include advanced management of users within the interface, it required of us to set up special rules that can only be set from the Firebase Admin SDK as per their documentation. To use this SDK it has to run on an authorized server and we unfortunately failed at making a solution work. Ultimately after losing hours of work we decided that it was not worth wasting too much precious time on this as the only issues it creates is that we can create new user accounts just fine as well as update their first name, last name as well as privileges and list of classes if the user is a professor but deleting an account is not possible as it required the specific admin rules. We can delete users just fine though if we login to the Firebase project directly and do it manually from there, which is actually quite straightforward to do. We concluded with our supervisor that this was a good enough solution for the time being and that there were indeed more important things to get to in priority.

4 Results

In this section we will present results obtained from what we were able to achieve with our implementation and hope to provide convincing evidence of the feasibility of this web-app interface for the QTrobot platform.

We also make available a youtube video demonstrating our projet here :
https://www.youtube.com/watch?v=4kSWLGu49G8

Here is a link to the repository of the project :
https://github.com/NervzZ/QTrobot-interface

4.1 Authentication

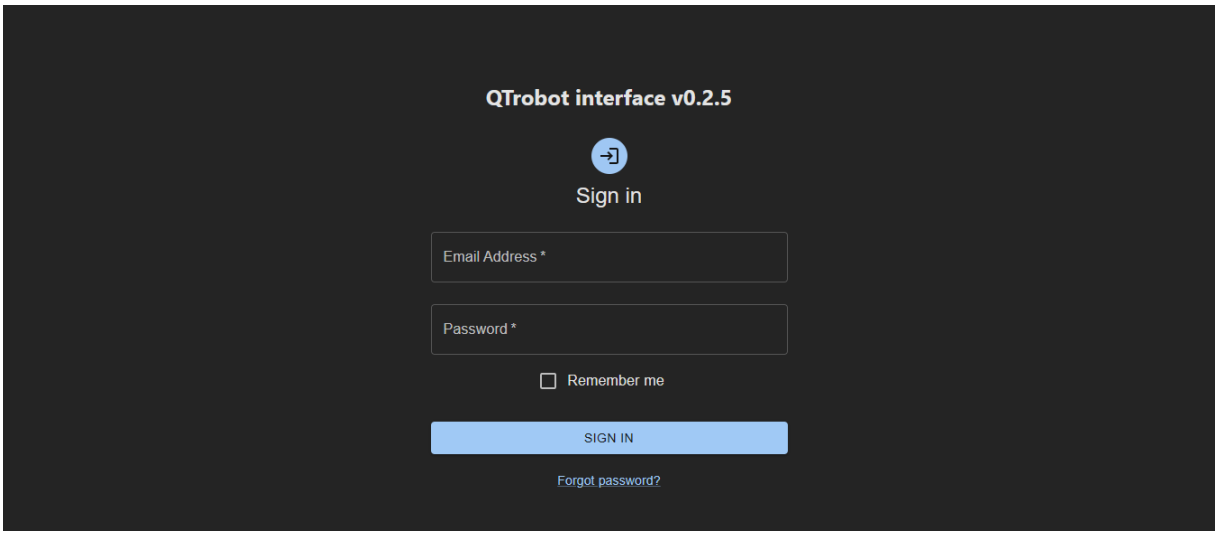

Figure 5: Authentication page

We were able to implement Firebase authentication system using accounts created from email and passwords. Creating an account in the interface can allow these newly created logins to be used here to access the interface. We prevented the app navigation routes to be rendered without the user being authenticated, thus ensure an unauthenticated person has no means to access parts of the app reserved for identified users.

4.2 Home page

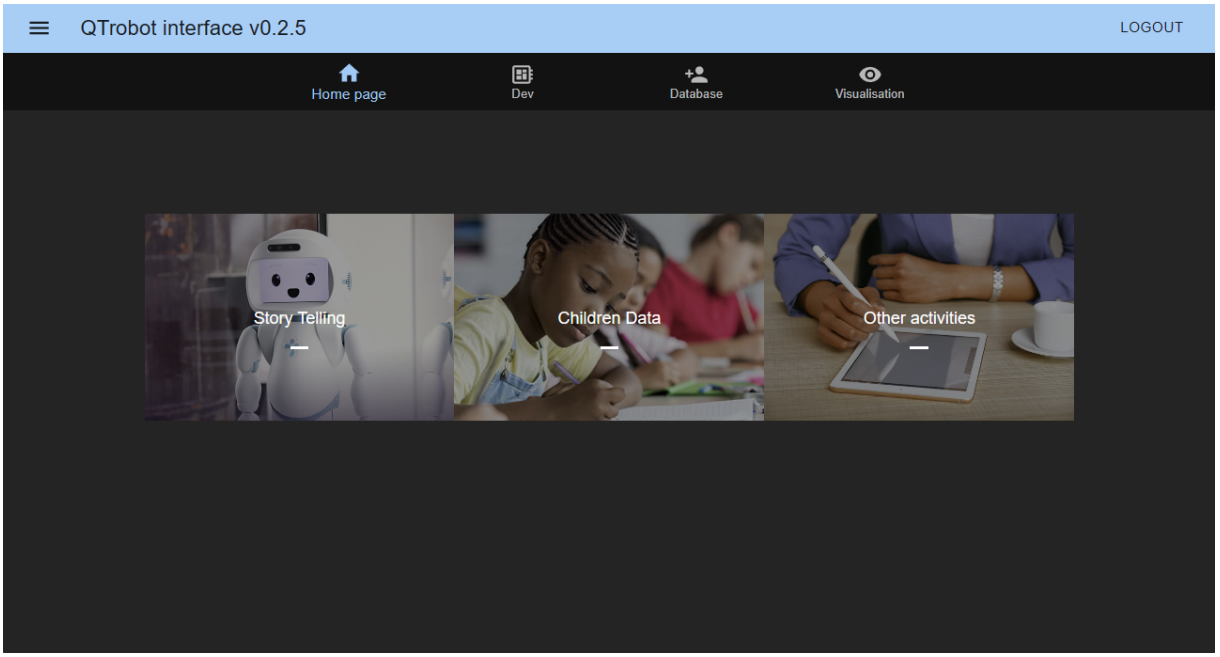

Figure 6: Homepage

Here you can see the simple homepage we implemented, it is intended for users with the "professor" role mostly. It acts as a hub where it's easy to link other applications / activities for the QTrobot, thus achieving the requirement of a simplified view for non-researcher roles. For now the big buttons are not starting anything yet as it is up to a future work to decide what we want to make accessible from this homepage. Another requirement was to give developer mode access for developer privileged accounts and this is what we see with the navigation bar on top. This navigation bar is not visible when logged in as a professor and the app will not render any of the developer-only pages of the interface, no matter if you try by url or other means so we achieved the requirement of splitting the views between developers and professors.

More specifically, the Children Data button would lead a user on a simplified database view where no admin managements actions are possible and also ensure each professor is only able to access data about children of their respective class since their account can have a list of the classes they are teachers from which was another important requirement we had about data privacy.

4.3 Database management

This is the database management system. Every data we're seeing here is from the Firebase Realtime Database which this page directly interracts with. A logged in developer can do the following :

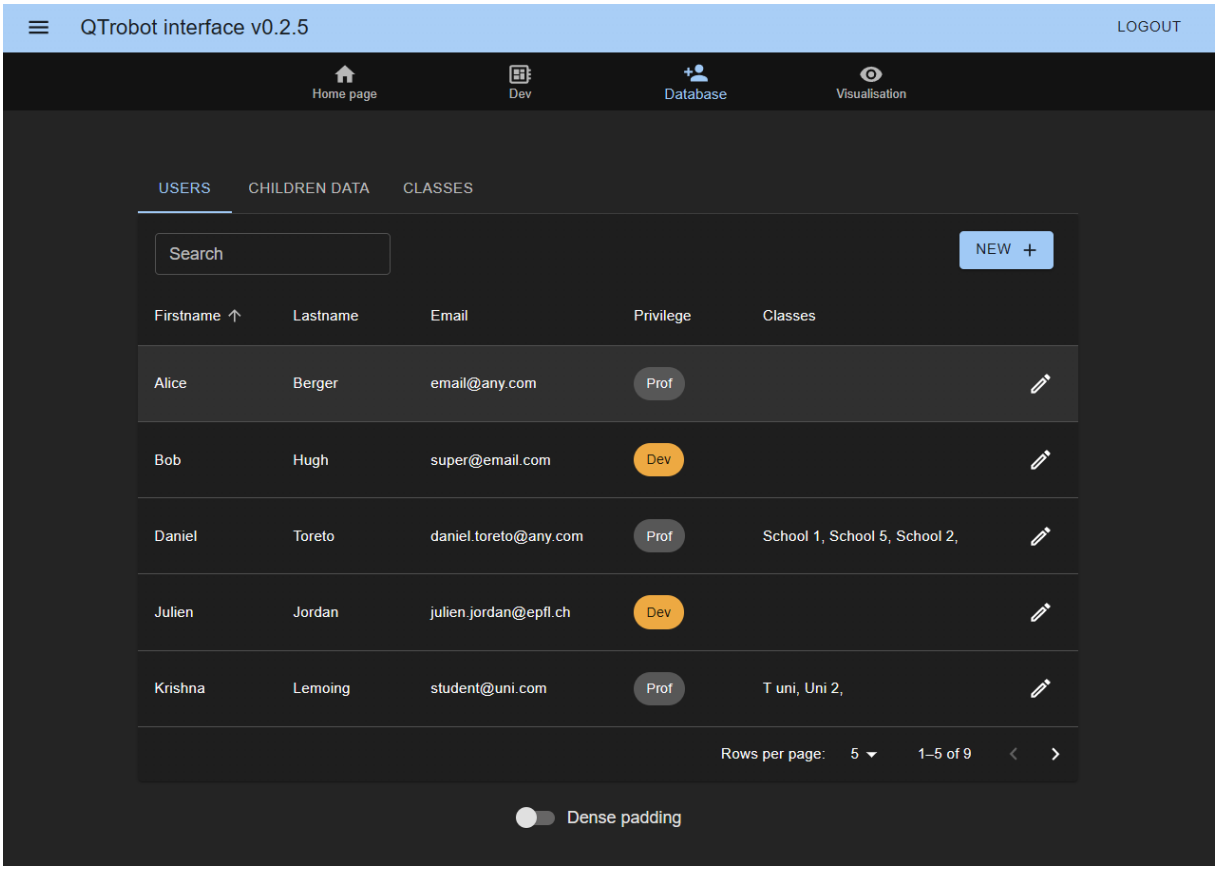

Figure 7: Database management system

1. Create and edit user accounts with professor or developer privileges
2. Set classes to the professor accounts
3. Total management of children data in the children tab
4. Total management of the classes in the class tab
5. Elegant table design and features for filtering / sorting, pagination and row density for a good user experience.

Using this system, we can completely remove the need for managing data in google sheets. This makes data management much smoother and faster while also providing the advantage of possible privilege based data access to preserve data privacy. We also take advantage of the real time nature of the database so that this page live-updates the visible data which means multiple users managing the data can see each other's actions reflect on the data in real time without having to refresh the page.

4.4 Dev page

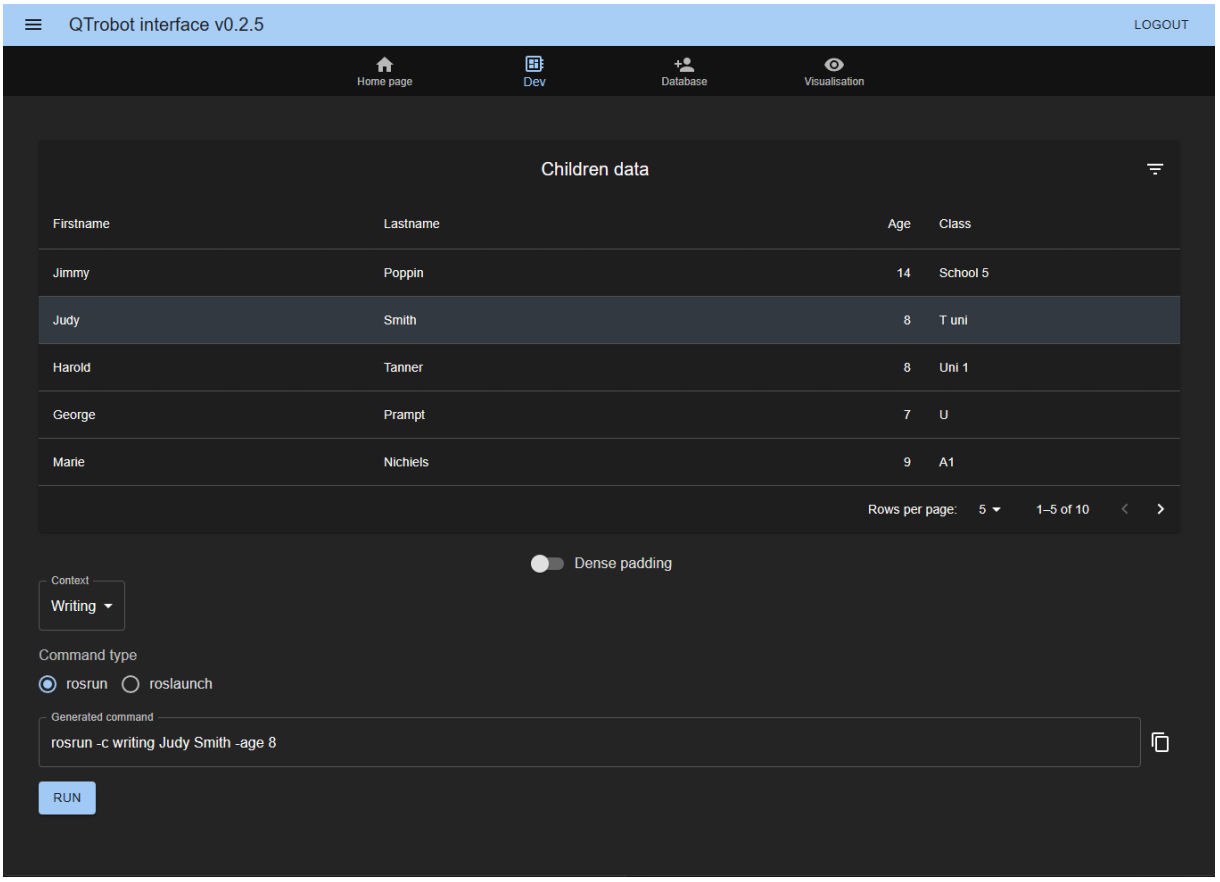

Figure 8: Developer page

This view is about one of the very first key features that was discussed. Here we can see children data and we are able to generate ROS commands simply by selecting the children we want to run our actions about, a context and the type of command we would like to run. As we could not access the QTrobot to test running real ROS commands we are still providing with an implementation that proves it's feasible with our implementation.

Currently if the user runs a ROS command, it is recognized as valid by our backend and, for the purpose of the proof of concept, runs commands from the host machine's terminal. We chose to use an "echo" command that prints the generated command we chose to run and it puts it into a dated file then automatically uploads it to online Firebase storage.

This proves we are able to make this built in terminal run similarily like the native terminal of the system, as we can chose to run any command we'd like by easily adapting the backend code. This feature allows the user to not have to look-up at different place to read and write by hand the commands he wants to run by providing him with simple yet

nonrestrictive options to generate the wanted ros command which smoothen the workflow, increase speed and ease of generating data as well as reduce user error (typos while making the command by hand).
It is also worth noting the user can edit the field at any time, this has been allowed for flexibility purpose so that if the user is unable to generate the command he wants due to current interface limitations, he is still able to edit and tweak the generated command to his liking.

4.5 Making the data available online

A key requirement was the ability to be able to retrieve already processed data. Indeed we do not want the user to need accessing the QTrobot or re-run commands if the data has already been generated before. We have made it possible and with our proof of concept implementation we show this feature as follows:
We used the file that result from the command a user runs from the dev page, if it's valid a file is indeed created and we automatically upload it to Firebase storage which is online. It is possible to go to this storage to view and download available files which we are indeed able to retrieve.

Therefore we are able to completely remove the dependency for physical access to the robot to view existing data, greatly enhancing access to existing resources.

4.6 Data visualisation

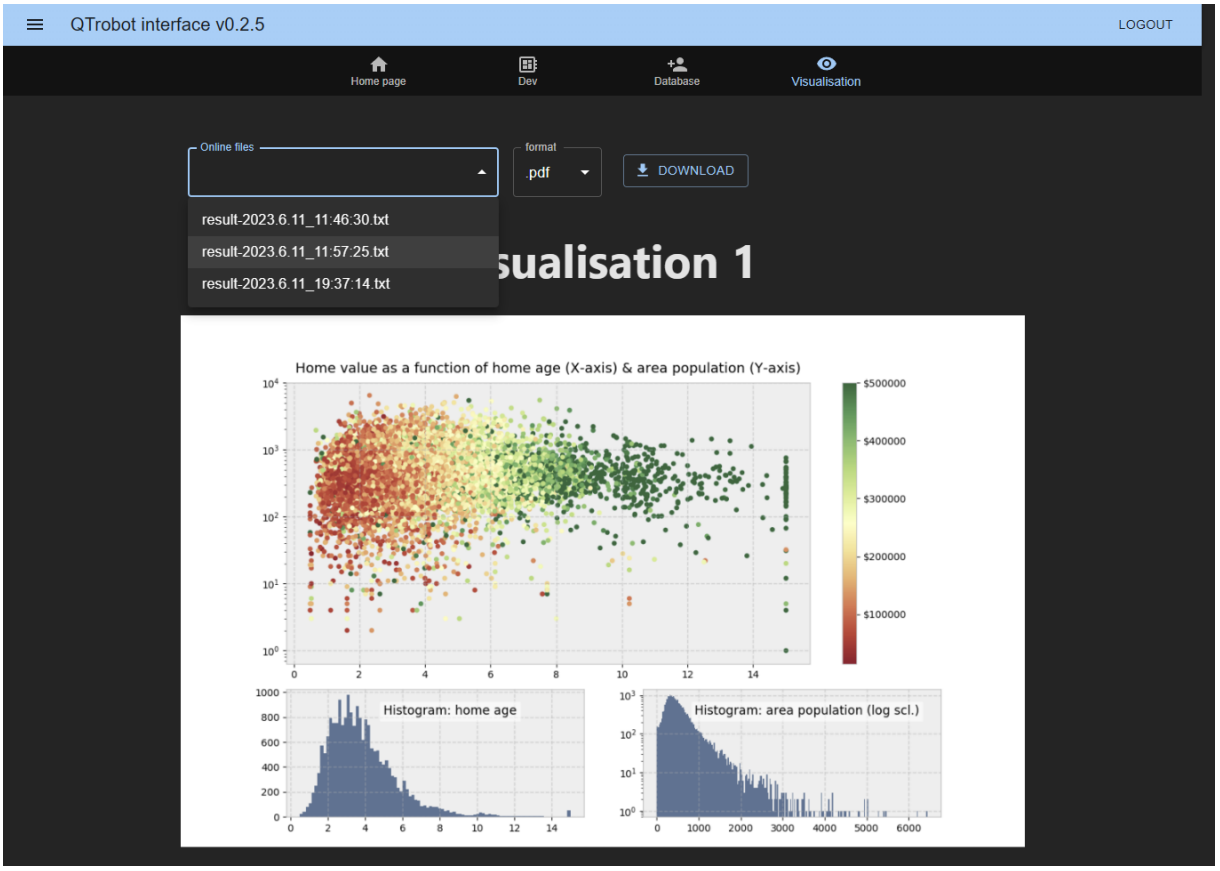

Figure 9: Data visualisation page

This page represent the data visualisation feature, it is mostly a placeholder to highlight some ideas of what it could become in the future work as we could not make a working implementation, due to time contraints, by integrating existing tools the iReCheck team members are using to generate graphs.
Nevertheless our supervisor provided us with one of the tool currently used for data visualisation which is actually displayed using a javascript build. This is very interesting as we're creating a React web-app which is also javascript so the potential to integrate such a tool is very high but still requires a lot of work to adapt it for a fully working system.

We also still wanted to include one of the features that was requested, which is to be able to download generated data. Indeed at the top of this page we implemented an example where the user could select the csv file he wants to download. The list of files comes directly from the Firebase Storage, the downloaded files does not come from the local files of the interface host. As a proof of concept, the files generated are .txt and selecting it then clicking on the download button indeed downloads the .txt file and not a .csv or .pdf yet, it would be the case on a future implementation where .pdf would

simply be a complete graph report about the selected .csv file while choosing .csv allows to retrieve the raw file itself.

This proves it is perfectly doable and the fact iReCheck team members rely on few tools written by certain members that require the user to set local servers to process their data make it all the more obvious how an in-built system such as this one would be helpful by providing a data-visualisation system packed with features, centralised in this app and allows for the files to be easily downloaded or even uploaded if needed which would be not only a much improved user experience but also a time saver as you would only need to start up this page and go on about your work. No need anymore to play around with various tools before being able to do something.

4.7 Making the application run on the QTrobot

We were able to install and configure different dependencies required to make the QTrobot operating system behave like a web server so that it can serve the interface to requesting users within the internal network. We were able to make it so when the QTrobot is booted, the server is up automatically, there is no need to start anything manually, not even open the session on the Ubuntu system. We also successfully reached the interface when connected directly to the robot which is very positive for a future long term implementation.

Our work did not imply any changes to the system's settings as it was specified avoid any changes that may alter the default behavior of the robot so we are confident our solution does not harm other activities on the robot. We did run into an issue after the manufacturers did an update on the available QTrobot which broke the internet access a user have when connected through the robot and we believe this to be the culprit as to why it rendered our interface unable to communicate with online resources sur as Firebase. Fixing this in the future would involved being more intrusive in the robot setup, specifically the network but our only goal is to restore having internet access through the Robot which we believe would resolve this issue.

5 Conclusion and future work

This project has lead to our present proof of concept that displays the many advantages of having a tailored interface to streamline the workflow working with the QTrobot for iReCheck team members. This would impact the entire team as well as other users such as professors involved by providing them with an elegant interface that implements features to tackle problems and answers to a variety of user needs in this work context. Our results also have been positive in indicating this project is more than feasible and would be for the profit of many.

As a future work we would first and foremost assess how big of an impact such an interface can make in improving work experience of the iReCheck members by running

tests with them and gather further feedback. Eventually development progress would be made towards a full implementation taking this project from a proof of concept state to an actual end product helping multiple researchers and other users in their productivity. As the portative nature of this interface, it is also interesting to note that it can be deployed to any QTrobot so that it becomes possible to work together and collaborate using different robots as the interface is the same and the resources and data are made available online.

References

[1] Basit Qureshi. "Exploring the use of chatgpt as a tool for learning and assessment in undergraduate computer science curriculum: Opportunities and challenges". In: *arXiv preprint arXiv:2304.11214* (2023).

[2] *Figma*. URL: https://www.figma.com/.

[3] *Google's Firebase*. URL: https://firebase.google.com/. (accessed: 09.06.2023).

[4] *LuxAI's QTrobot*. URL: https://luxai.com/humanoid-social-robot-for-research-and-teaching/. (accessed: 09.06.2023).

[5] *Meta Open Source - React*. URL: https://react.dev/. (accessed: 09.06.2023).

[6] *MUI: The React component library you always wanted*. URL: https://mui.com/. (accessed: 09.06.2023).

[7] *OpenAI chatGPT*. URL: https://openai.com/blog/chatgpt. (accessed: 09.06.2023).

[8] *StackOverflow*. URL: https://stackoverflow.com/questions.

[9] *Video : How to learn to code FAST using ChatGPT*. URL: https://www.youtube.com/watch?v=VznoKyh6AXs. (accessed: 12.05.2023).