NaoChallenge23 Technical Relation

NAONEXUS

FOR MORE INFO:

Visit our website
View our repository
Write an email at: socialnaonexus@gmail.com

Contents

Abstract:

In this year's *NaoChallenge* it was decided to develop two main projects oriented to sustainability and green energy. The *mission* is to bring sustainability to everyone making it more accessible and easier to understand. These projects introduce people to the world of solar energy and their aim is to prevent energy waste in big infrastructures.

1. METHODOLOGY AND TECHNOLOGIES USED

As a big group there was a necessity to keep track of the tasks and keep everyone in sync with the latest developed software, for this reasons two important technologies were used: *GitHub* and *Notion*.

1.1. DevOps with Notion and Agile methodology

To work on the project and keep track of the progress made the team used *Notion*, a collaborative note-taking software. The term *DevOps* comes from the last part of *software development* and *technology operations* and its aim is to improve the collaboration end-to-end. With the web application Notion, which offered many useful tools such as task management, project tracking and to-do lists, the team has always been organized and focused on the work to do each session. This collaboration platform also integrates wikis and databases, creating all-in-one workspaces for notetaking and data management. The projects were managed with *Agile* methodology, the problems were split into smaller tasks that were then distributed to the team members, this brought to an increased efficiency in software development and faster developing times.

1.2. GitHub

GitHub is a powerful service that helped us store and manage our code. It is a website and cloud-based Git repository hosting service that essentially helps the user keep track of the changes made to the code thanks to *version control*.
Through *branching* and *merging* each member was able to safely work on the code avoiding the direct modification of the original version. All of the changes made can eventually be reverted if they break the algorithm.

PRESENTING TWO PROJECTS

The first project uses NAO as an *assistant* in a building. NAO is connected to a phone through an HTTP server. Users can connect the phone to the server and send environmental data collected through an external sensor. When NAO receives the environmental data he then manages the building's domotics by turning on or off the *lights*, raising or lowering the *blinds* or turning on or off the *heating*.

The second project required the collaboration with an external company and the team worked with *Amperia* which aim is to make green transition easier and more efficient. By connecting NAO to *Solaredge* website (developed in order to design solar panel installation) an HTTP server it guides the user through the compilation of the website that creates the solar panel installation plan.

2. NAO AS A DOMOTIC HELPER

This project involves NAO, which has to communicate with a sensor in order to understand the environmental conditions and to control the blinds, lights and heating with the local domotics

server. It includes multiple steps: initially the environmental data are measured with the sensor, the a PDF is generated and selected with the app developed. It is later on sent to the server with the number of people present in the room and subsequently transmitted to NAO, which has to control the domotics if it goes over certain thresholds. The project contains multiple elements:

- Sensor: an off-the-shelf sensor that exports measured data to the PDF
- App: an app that reads the PDF and sends it to the HTTP server with the number of people in the room and the internal light levels
- HTTP Server: a Rest server that allows HTTP pages to visualise environmental data extracted from the PDF and communicates with the domotics server
- NAO: a humanoid robot that collects data from the server and controls the domotics based on their values. It spells out loud the received data.

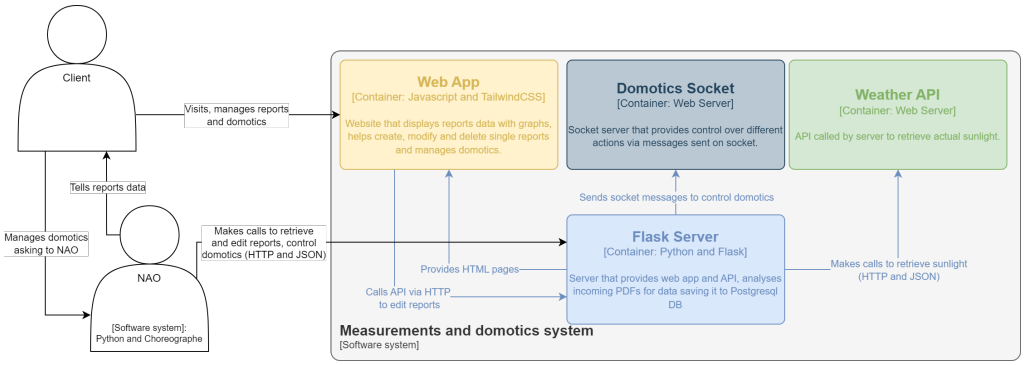

Fig. 1. NAO domotics project architecture

2.1. The sensor

A significant part of the project was being able to analyse the environment and collect data of the CO2 concentration, temperature and humidity. At first the team wanted to use an Arduino that had to connect to the HTTP server. Lots of problems were encountered so we decided to buy an *off-the-shelf* sensor. This guaranteed more precise results with the drawback that they were exported to the PDF.

In order to analyse these data a device that extracted the PDF and sent it to the server was needed. Initially the team wanted to connect the sensor to one of the USB of NAO but without root permission drives couldn't be mounted to the humanoid, making it impossible to access any external memory from NAO. It was therefore decided to use a mobile app.

2.2. The App

This mobile app is used to load the PDF and send it to the server. It was used because there was no way to connect the sensor directly to the server. This is a device that helps transfer the sensor's

data to the server adding other useful measurements that can't be made with the sensor. This app is written in *Flutter* to make it cross-platform and flexible. It includes a file picker and an input to insert the *number of people* in the room. When sending the file it adds to the request the *light* values measured with the phone sensor. The app also lists all the data stored in the server.

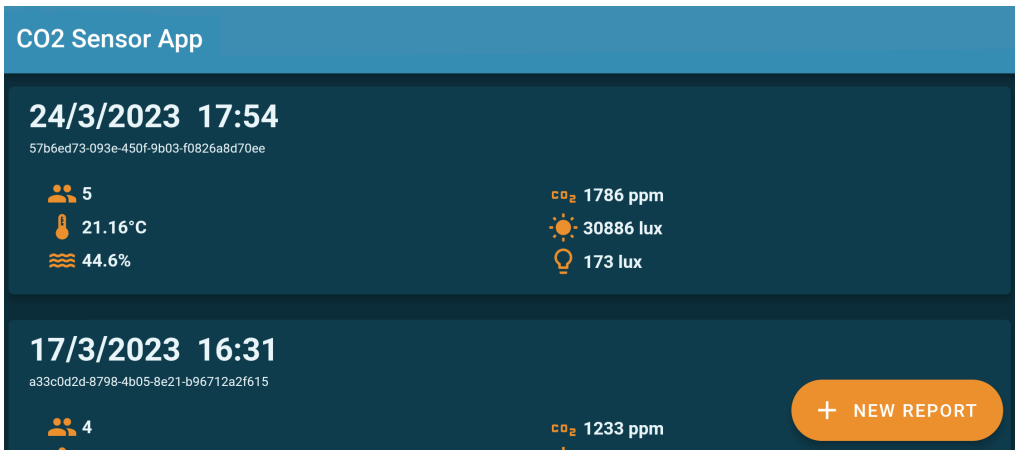

Fig. 2. Domotics application

2.3. NAO

NAO is the central point of the project. It is connected to the HTTP server and receives the collected environmental data used by NAO to understand which calls to make to the server. With control domotics NAO tells the user what it is doing. NAO has the functionality to be used as a "smart home speaker" like *"Alexa"*: the end user can ask NAO to make adjustments to the domotics controlling it directly. When receiving new data NAO communicates them to the user and based on various *thresholds* it can control the smart devices.

2.4. HTTP Server

The central point where all the data of the project is managed, this is a complex part written in Python that analyses PDFs, adds missing information, hosts a Web App that views and manage data and hosts a Rest API to connect all the clients like the mobile app and NAO. The initial idea was to run this server on NAO but the latest software versions removed the ability to *root* the robot and install Python packages, it was later decided to run this server on a *Raspberry PI* with a *fixed IP* to make it accessible to every client on the network.

2.4.1. Web App

An important part of the server is the web app that it hosts: the HTML pages are populated with Jinja and are sent using Flask. This library is responsible of running the Rest server that provides data in *JSON* format. In this web app the data of the reports is viewable with a *graph* or as a *list*, every report is *editable* and *deletable* and it is possible to *add new reports*.

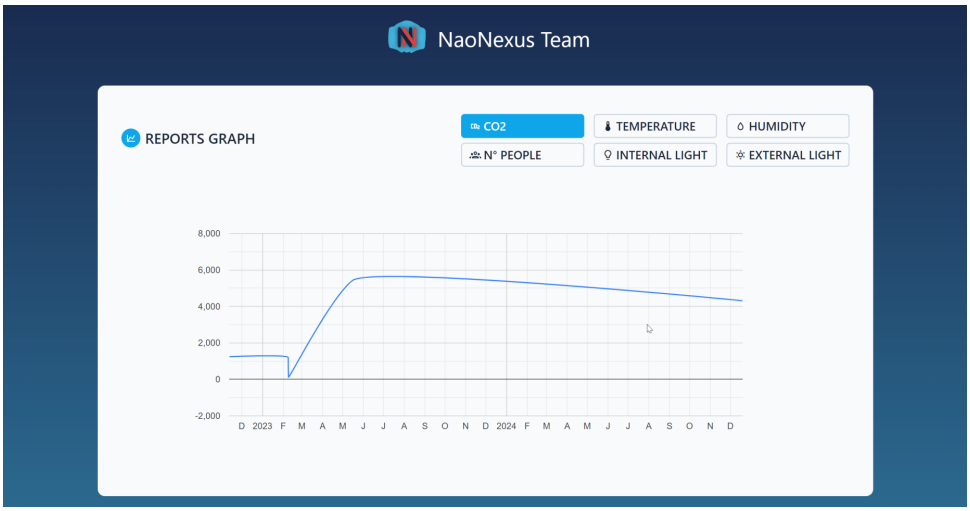

Fig. 3. Screen to see reports summary

2.4.2. Rest API

The documentation for the API can be found here.

The Rest API facilitates the exchange of data between all the connected clients, which are then sent via *JSON* using *HTTP* and are accessible through multiple endpoints. This rest server is written in Flask as the App.

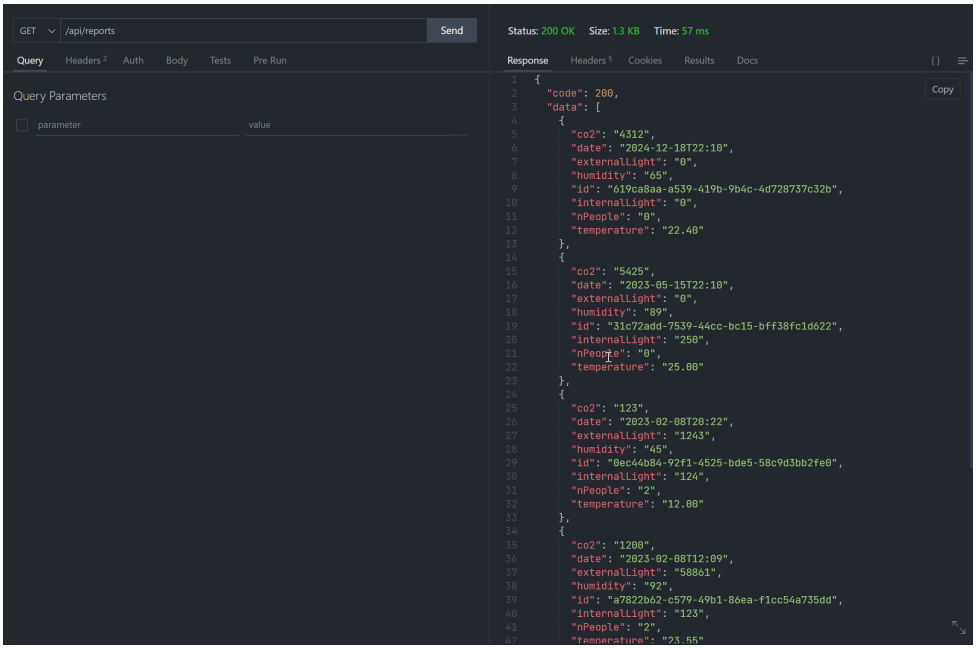

Fig. 4. Sample Rest API response

2.4.3. PDF Analysis

In order to analyse the PDF it was necessary to use the pyPDF2 library, which allowed the conversion of the file created by the CO2 sensor into a string. The parameters were difficult to individuate but after removing *spaces* and *line breaks* the program was able to find useful values by searching for *keywords* allocated in specific points of the PDF. In this way it is possible to extract various data from the PDF like: date of the measurement, CO2, temperature and humidity.

2.4.4. Domotics control

To control the lights the server was connected via *Telnet* to the domotics server of the building. After some difficulties trying to find the right commands and thanks to *Amperia*'s guidance, it was possible to understand the appropriate unique codes assigned to *lights*, *LIMs*, *blinds* and *thermostats* in a specific environment, in order to control them through the *Telnet* connection. This connection brought the HTTP Server to become a *client* for the domotics server in order for it to control the connected devices.

2.4.5. External light analysis

The measurements of the internal light is retrieved from the sensor but the external sunlight cannot be measured by the app. The first idea was to add an *Arduino* sensor externally, this couldn't be applied to every possible situation and building. The solution was to call an external API (Open meteo), this API returns solar irradiance in $\frac{W}{m^2}$ that is then converted in lux by the server.

3. NAO AS A GUIDE TO SOLAREDGE

This project involves NAO to interact with the clients of the partner *Amperia*, it asks for details of the building and generates a project for a solar transition with the website Solaredge making sustainability accessible to everyone. It includes two phases: an initial phase where NAO asks questions to the customer and inserts the answers in the website asking the customer to do it when it isn't able to control the website and a second phase where NAO tells all the results achievable from the project and the impact that it has on the Earth. The project contains multiple elements:

- HTTP Server: a Rest server that communicates with NAO and controls the website inserting requested data
- NAO: a humanoid robot that asks for details and sends the answers to the server that analyses them. It spells out loud the result from the analysed data, generated with Solaredge website.

3.1. NAO

NAO is an essential part of this project, it handles the interaction with the customer that wants to configure its solar panel installation. The initial idea was to add the libraries to control Solaredge website directly as a *package* of NAO's Python but the same problem of the
other project was encountered: there wasn't a way to install packages. The solution was to add a computer as a *server* and access the website on it making the website accessible to the user. When extracting the customer answers a problem was encountered: NAO voice recognition works only on known words but the inputs from the user could have been anything. It was then decided to register the response from NAO and send the WAV file to the server to analyse it.

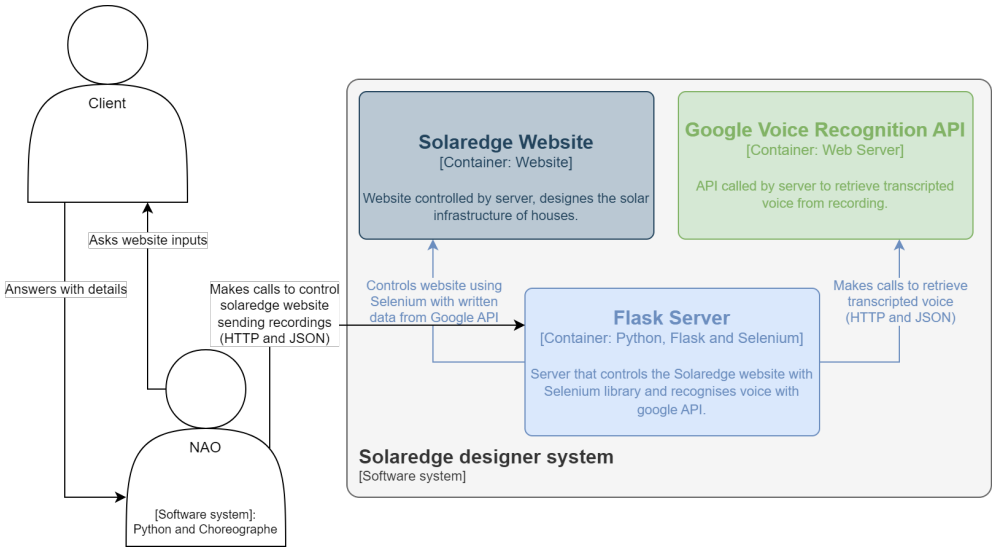

Fig. 5. Solaredge architecture

NAO makes questions and registers the answers of the customer and then procceds to send them as WAV to the server. When the project is completed the robot says aloud the results expected from the project.

3.2. HTTP Server

The central point where all the data of the project are stored, this is a complex part written in Python that communicates with NAO receiving voice registration. It then analyse these registrations by extracting the transcription that are inserted in Solaredge website. It includes various static and dynamic endpoints to manage data and is written using Flask. Similarly to the other project the server isn't run directly on NAO but on an external PC, this makes the code flexible and helps the user to adjust the inserted data and access the website.

3.2.1. Rest API

The documentation for the API can be found here.

This Rest API manages the traffic of data from NAO to the server and the other way round. It includes various endpoints and uses HTTP protocol, furthermore WAV files registered by NAO are parsed and inserted in Solaredge website.

3.2.2. Voice Recognition

The transcription of the voice is done using Python's library SpeechRecognition, it uses *Google*'s models and returns a text from an audio file. It is used to transform NAO's registrations into text to be entered in the website. The audio file given to this library presents two channels that include the *right* and *left* microphones of NAO.

3.2.3. Solaredge website Control

Selenium is a Python library containing numerous instructions that allow for a direct interaction with a local browser of choice. This tool can scour through the HTML code of a website and, in doing so, finding, selecting and interacting with specific graphic elements within the code itself (such as textboxes, lists, links etc.). It *automatizes* both the *extraction* and *importation* of the data, optimizing the process and making it more efficient. This library is fundamental in inserting the parsed questions extracted from the dialogue between NAO and the customer.

4. Team and Authors

The NaoNexus team is divided in two sections: *coding* and *social*. The components who worked on the social part of the project made sure to represent the NaoNexus experience at its best, creating and posting new content everyday in order to highlight the major progresses made by the group during these months. The social sub-team had a key role in presenting the project with videos, pictures, reels and TikToks to post on our socials and the Naonexus website.

The coding division took care of the robotics project per se, developing the code and working with the technologies and methodologies previously explained.

4.1. CODING DIVISION

Riccardo Antonelli - Team Leader
Francesco Bernardi
Elisa D'Iseppi
Filippo Buso
Edoardo Polfranceschi

4.2. SOCIAL DIVISION

Alberto Rubini - Team Leader
Davide Masini - Mascotte
Antonio Galati
Arianna Antonelli