NaoChallenge23 Technical Relation

NAONEXUS

FOR MORE INFO:

Visit our website
Write an email to: 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 help avoiding wasting energy in big buildings.

1. METHOLODGY 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 reason 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 manages the building's domotics by turning on or off the *lights*, raising or lowering the *blinds* or turning on or off *heating*.

The second project brings the collaboration with an external company: *Amperia*, this is a company that works on green transition. By connecting NAO to *Solaredge* website (a website to design solar panel installation) via an HTTP server it can guide the user to compile the website and create a solar panel installation plan.

2. NAO AS A DOMOTIC HELPER

This project involves NAO to communicate with a sensor to understand the environmental conditions and the local domotics server to control the blinds, lights and heating. It includes multiple phases: initially the enviromental data is measured with the sensor, the generated PDF

is selected via the app and sent to the server with the number of people present in the room, the server then sends this data to NAO that controls the domotics if it goes over certain thresholds. The project contains multiple elements:

- *Sensor*: an off-the-shelf sensor that exports measured data to 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
- *HTTP Server*: a Rest server that serves HTTP pages to visualise environmental data extracted from the PDF and communicates with the domotics server
- *NAO*: an humanoid robot that gets data from server and, based on this data controls the domotics. It spells out loud the received data.

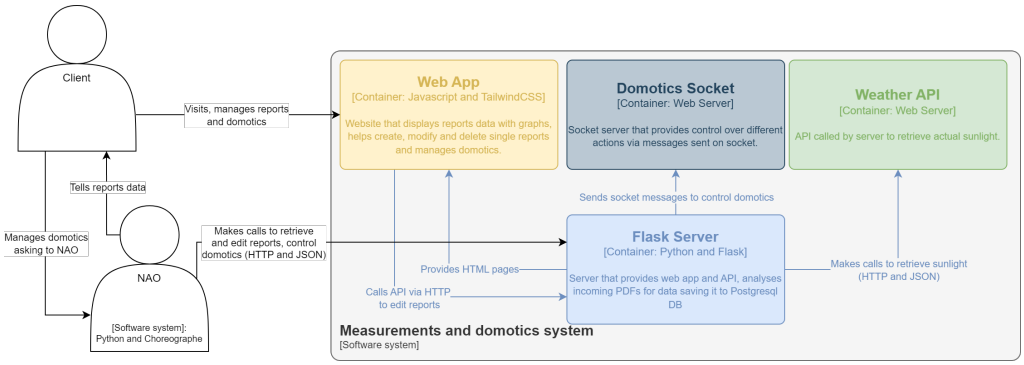

Fig. 1. NAO domotics project architecture

2.1. The sensor

A vital part of the project was being able to analyse the environment and getting data for the CO2 concentration, temperature and humidity. The first idea was to use an Arduino connecting it to the HTTP server. Lots of problems were encountered when connecting it to the server so it was decided to buy an *off-the-shelf* sensor. This guaranteed more precise result with the drawback that these results were exported to PDF.

In order to send this data to be analysed a device that extracted the PDF and sent it to the server was neeeded. The initial idea was 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 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 carry the sensor's data to the server adding also 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 current measured *light* from the phone sensor. The app also lists all the data stored in the server.

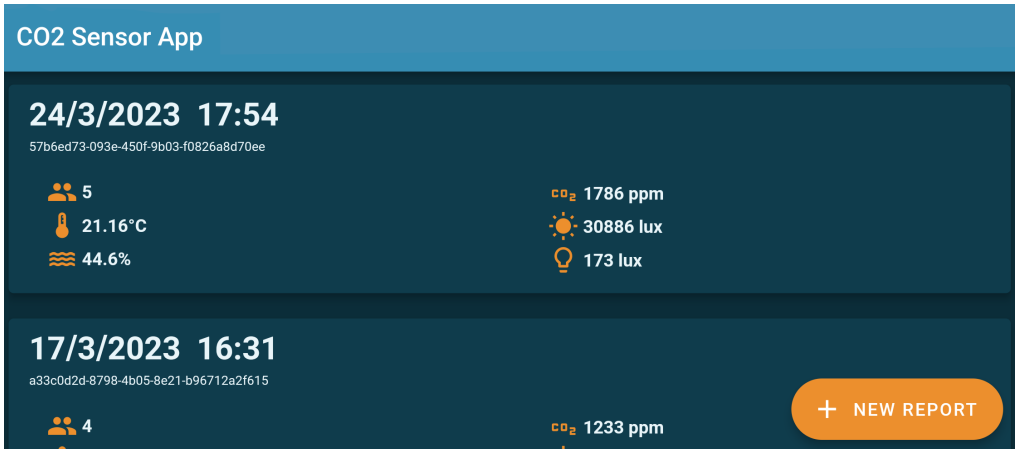

Fig. 2. Domotics application

2.3. NAO

NAO is the central point for the project. It is connected to the HTTP server and receives the collected eviromental data, based on this data NAO makes various calls to the server to control domotics telling 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 tells it 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, this is a complex part written in Python that analyses PDFs, adds missing information, hosts a Web App to view data and manage it and hosts a Rest API to connect all the clients like the mobile app and NAO. The initial idea was to run this server in 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 it hosts: the HTML pages are populated with Jinja and are served 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*.

2.4.2. Rest API

The documentation for the API can be found here.

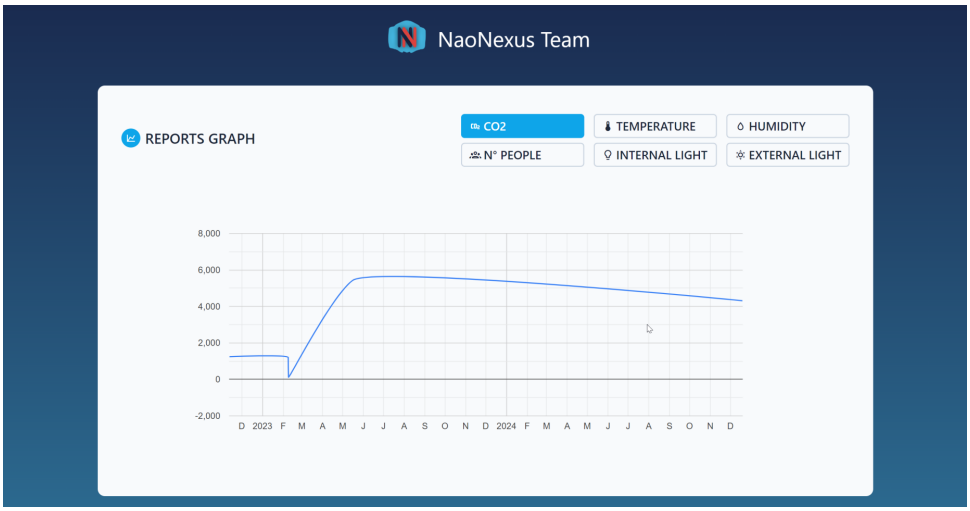

Fig. 3. Screen to see reports summary

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

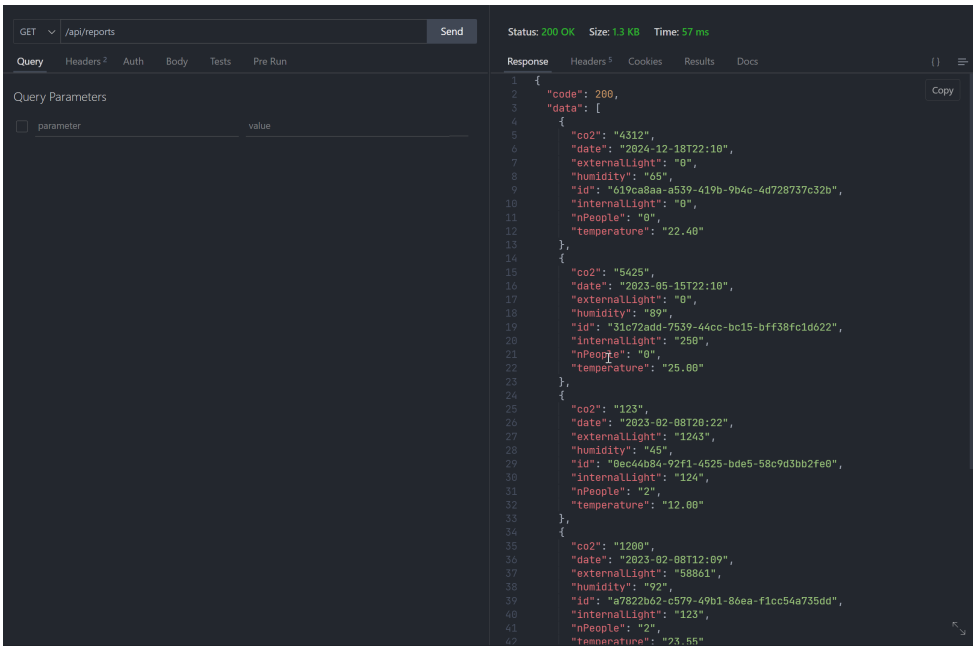

Fig. 4. Sample Rest API response

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 measurement for 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 adopted with all situations and all buildings. 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*: an humanoid robot that asks for the 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 for this project, it handles the interaction with the customer that wants to configure its solar 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 an option to install packages. The solution was to add a computer as a *server* and access the website on it making the website accessible also to the user. When extracting the costumer answers a problem was encountered: NAO voice recognition works only on known words but the inputs from the customer could have been anything, it was decided to register the response from NAO and send the WAV file to the server to analyse it. NAO makes questions and registers the answers of the customer, sends these answers as WAV to the server. When the project is completed it says aloud the results expected from the project.

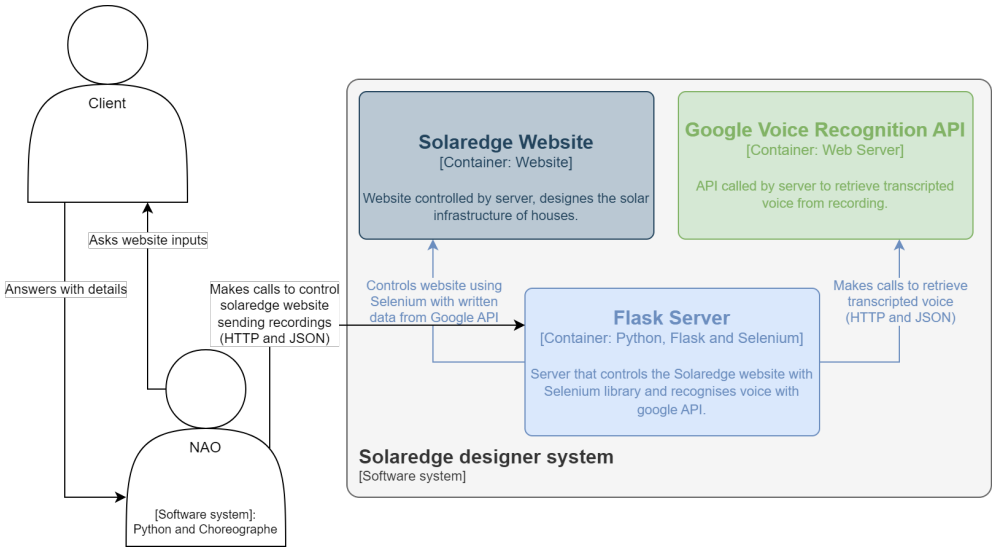

Fig. 5. Solaredge architecture

3.2. HTTP Server

The central point where all the data of the project, this is a complex part written in Python that communicates with NAO receiving voice registration, analysing these registrations by extracting the transcription and inserting these transcriptions in Solaredge website. It includes various static and dynamic endpoints to manage data and is written using Flask. As with the previous project the server isn't run directly on NAO but on an external PC, this makes the solution 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 from the server to NAO, it includes various endpoints and uses HTTP protocol, 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 a library in Python that 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 data, a

fact which becomes extremely valuable considering our task at hand. This library is fundamental in inserting the parsed questions extracted from the dialogue between NAO and the customer.