



HOLY SPIRIT UNIVERSITY OF KASLIK

SCHOOL OF ENGINEERING

Human Robot Interaction:

Final Project HR Manager

By: Georges Sarkis

Maher Halabi

Georges Saad

Christian Tauk

Presented to: Dr. Elie SAAD

USEK – *Fall 2024/2025*

Introduction:

The advent of advanced robotics has revolutionized various industries by introducing innovative solutions to enhance efficiency, accuracy, and user engagement. This report delves into the development and implementation of Pepper, a humanoid robot designed to optimize the recruitment process in an expanding electrical installation company. The primary objective of this project was to streamline the applicant assessment process by leveraging Pepper's capabilities to conduct preliminary evaluations, administer technical quizzes, and provide instant feedback. By combining intuitive interactions, automated assessments, and seamless integration of tools like Choregraphe and Python, this project demonstrates the potential of human-robot collaboration in solving complex organizational challenges. The report outlines the project's motivation, design steps, implementation, and use cases, offering insights into the technological and practical aspects of Pepper's deployment.

Project Proposal and Motivation:

Motivation:

An expanding electrical installation company is seeking skilled electrical engineers to manage an increasing number of projects effectively. To streamline the recruitment process, the company is introducing Pepper, a robot assistant designed to interact with applicants, conduct preliminary assessments, and gather essential information to support the hiring process.

Project Steps:

The process that pepper will take you through:

1. Greet you and check If you are at the right Robot (If not, guide to the right one)
2. Upload CV
3. The applicant undergoes a survey test to check how much he is familiar with some specific software.
4. The Score that the applicant obtained in this test will be displayed on the screen.
5. The applicant undergoes a technical test (quiz) of his technical skills.
6. The Score that the applicant obtained in this test will be displayed on the screen.
7. Pepper tells the User what to do in accordance with his Score.
8. Pepper's farewell to the applicant

Materials Used:

Applications:

- Choregraphe
- VS code

Languages Used:

- Python 2.7 and libraries
- Java script
- HTML
- CSS
- Qi-Chat

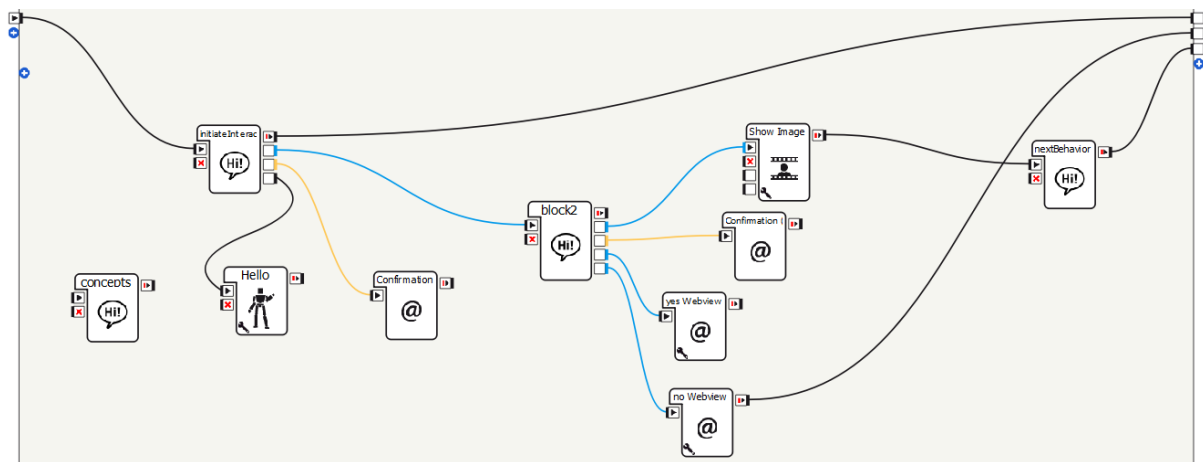
Use Cases:

UC-ID and Name	UC-01: Initiate Interaction		
Created By	Georges Sarkis Maher Halabi	Creation Date	23/9/2024
Actors	<i>Robot (Pepper) and an electrical engineer applicant seeking HR</i>		
Trigger	The applicant says a greeting word to Pepper		
Description (Objectives/Goals)	<i>Pepper greets the user, then Pepper checks if the applicant is at the right robot (he checks if he is an applicant, and then if he is here as an electrical applicant). If yes, he shows him a QR code to scan and fills his application. If not, Pepper orients the applicant to the right robot.</i>		
Preconditions	PRECOND-1. the User greets the robot companion		
Postconditions	POSTCOND-1. A QR code to scan POSTCOND-2 Pepper notify the user that his survey skill test will now start		
Action Sequence (Success Scenario)	<ol style="list-style-type: none">1. User greets Pepper2. Pepper asks the user if he is an applicant3. Pepper receives a yes from the user4. Pepper asks if he is an electrical or computer applicant5. Pepper receives electrical, he shows the User a QR code.6. The User says next when he finished filling in the info in the QR code.		

	7. Pepper notifies the user that his survey skill test will now start. (Use case ends)
Extensions	3a. Pepper receives a no: 3a1. The program ends 4a. Pepper receives computer 4a1. The user is referred to another robot. (Use case ends)
Requirements	
Storyboards	<i>Sequence of snapshots (illustrations)</i>
Priority	The priority is to initiate interaction and make sure that the user is at the right place and at the right robot
Related Use Cases	US02

Implementation:

Use case 1 (Initiate Interaction):



This is the diagram of the first use case.

Overview for the first use case:

The "onStart" event triggers the initiateInteraction dialogue box, from which we generate different outputs. The "onStopped" event is triggered if the user is not an applicant, and the use case then ends. The blue output, called "yesOutput," is

triggered if the user is an applicant at the company, taking them to Block 2. The yellow output, "showQuestion1," calls the confirmation page (a WebView) to display the question and allow the user to select inputs on the tablet. The black output, "helloAnimated," sends an order to the Hello box to perform the greeting animation.

The Block2 dialogue box also has multiple outputs. Its primary purpose is to determine if the applicant is applying for Electrical or Computer Engineering. First, a variable named "showOptions" is set to 1 and sent to the confirmation page to display the question and let the user choose an option on the laptop. If the applicant selects "Computer," a variable called "computer" is sent to the "no" WebView. From there, a picture is shown referring the user to another robot, and the use case ends with an "onStopped" event. If the applicant selects "Electrical," a variable called "electrical" is set to 1 and sent to the "yes" WebView.

Next, a variable called "showImage" is set to 1 (output from Block2) and sent to the "Show Image" WebView, which displays an image (a QR code). The signal then moves to the next dialogue box, called "next Behavior," which notifies the user that a survey will begin. Afterward, the use case ends.

concepts01 Dialogue Box:

```
topic: ~concepts01()  
language: enu  
  
concept: (greetings) ^rand [hi hello hey "hey there"]  
concept: (names) [jad john chris mark maher george]  
concept: (position) [electrical computer]  
concept: (repeat) ["i did not understand" repeat "what did you say" sorry]  
concept: (Neutral) \style=neutral\ \rspd=90\ \vct=110\  
concept: (joyful) \style=joyful\ \rspd=90\ \vct=95\
```

In this dialogue box, we implemented all the concepts required for the first use case. For example, we created a concept called "greetings," which uses the ^rand function to randomly select a word from a predefined array of variables. We also created a concept called "names" because the robot we were using has

limited sound recognition capabilities, so we preset specific names in the system.

Additionally, we defined variables such as "position," "repeat" (used when the user asks the robot to repeat something they didn't understand), and "Neutral" and "joyful" to adjust the tone Pepper uses when interacting with the user. Note that style, "rspd" (the speed at which Pepper speaks), and "vct" (the tone of Pepper's voice) are pre-existing functions in the QiChat language.

initiateInteraction Dialogue Box:

```
topic: ~initiateInteraction()
language: enu

include: concepts01_enu.top

u: (~greetings) $helloAnimated=1. \RSPD=90\ \pau=2000\ ~greetings what is
your name
  ul: ({my name is} _~names) nice to meet you $1 $hisName = $1
  ^nextProposal

proposal: are you applying for a position at this company? $showQuestion1=1
  ul: (yes) $yesOutput=1
  ul: (no) please refer to another robot $onStopped=1
  ul: (e:yesAnswer) $yesOutput=1
  ul: (e:noAnswer) please refer to another robot $onStopped=1
  ul: (~repeat) ^sameProposal
```

First, we included the concepts block in our initiateInteraction dialogue box. When Pepper hears any of the words defined in the "greetings" concept, the interaction is triggered. Pepper sets the "helloAnimated" variable to 1, which activates the animated block and makes Pepper wave. We also programmed Pepper to speak at a speed of 90, which is slightly slower than the standard speed of 100. After this, the system pauses for 2000ms (2 seconds) to allow Pepper to complete the wave. Finally, we call the "greetings" concept so Pepper greets the user before asking for their name.

Next, we proceed to *ul* (note that the user cannot access *ul* without completing *u*, as *ul* falls within the scope of *u*). The user provides their name, and the phrase {my name is} is optional due to the use of braces {}. An underscore _ is

used to inform QiChat that the name will be an input we want to store. Pepper then responds with, “Nice to meet you, {user name}.” To store the name, we write \$hisName=\$1, saving the user's name in a variable called "hisName." We then call the function ^nextProposal.

In this proposal, the user is asked if they are applying for a position at the company. The same message is displayed on the screen by setting the output of the "showQuestion1" to 1, which serves as the activation key for the confirmation (further details are explained in subsequent sections of the first use case).

If the answer is *yes* (via voice input), it is captured as (yes), while if the user presses on the tablet screen it is captured as (e:yesAnswer), which raises an event called "confirmation1" created in the HTML page. The same logic applies to *no* answers. If the response is *no*, Pepper will prompt the user to refer to another robot and send a signal to end the use case. If the response is *yes*, the "yesOutput" is set to 1, moving the flow to the second dialogue box, *block2*, to continue the algorithm.

block2 Dialogue Box:

```
|topic: ~block2()
|language: enu

|include: concepts_enu.top

u: (e:onStart) $showOptions=1 ^nextProposal

proposal: \RSPD=90\ \pau=500\ are you applying for an electrical engineering or computer
engineering position
    u1: (~position) ~joyful Ah so you are applying for $1 ~Neutral $hisPosition=$1
^first["$hisPosition==electrical $electrical=1 ^nextProposal" " 'please refer to another robot'
$computer=1 \pau=1000\ $onStopped=1 "]
    u1: (e:yesAnswer) $electrical=1 ~joyful Ah so you are applying for electrical ~Neutral
^nextProposal
    u1: (e:noAnswer) please refer to another robot $computer=1 $onStopped=1 \pau=500\

proposal: please scan the qr code on the tablet screen, to submit all the files required
$showImage="pics/qrcode.jpg" $onStopped=1

u: (~repeat) ^previousProposal
```

The working mechanism of *block2* is very similar to that of the Initiate Interaction dialogue box. In this block, after receiving an “onStart” signal (when

“yesOutput” is set to 1 in the Initiate Interaction dialogue box), the variable “showOptions” is set to 1. This triggers the confirmation page, displaying a message on the screen and allowing the user to choose between different options (e.g., "computer" or "electrical" in this case). The user is then asked if they are applying for a position as an electrical or computer engineer, and their selection is saved in a variable.

Next, we call the concept joyful, which makes Pepper speak in a joyful tone to confirm the user's chosen position. After this, the tone reverts to Neutral. We then use the ^first function to define the logic for triggering specific outputs. For example, if hisPosition==electrical evaluates to true, Pepper sets the "electrical" output to 1, activating the yes WebView (refer to the yes/no WebViews section). After displaying the message, the system moves to the next proposal.

If the user's position is not electrical, Pepper will inform the user to refer to another robot. In this case, the "computer" variable is set to 1, activating the no WebView. The system then pauses for one second to give the user time to see the displayed message before ending the use case. Note that the same procedure applies when raising events if the user selects outputs on the screen.

If the user selects "electrical," the flow proceeds to the next proposal. Here, the user is asked to scan a QR code. For this, we created a showImage variable of type string. The QR code image, named *qrcode.jpg*, is stored in the pics folder inside the html directory. By default, QiChat searches within the html folder, but we added pics/ to the path to ensure it locates the image correctly.

nextBehavior Dialogue Box:

```
topic: ~nextBehavior()
language: enu

include: concepts_enu.top

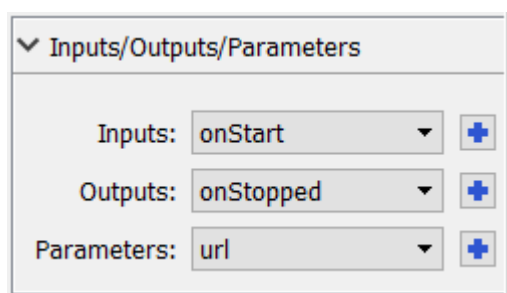
u:(e:onStart) \pau=2000\ When you fill the application on the site, please say next to begin your
quizz |
    ul:(next) ^nextProposal

proposal: now we are going to start conducting the survey... goodluck $onStopped=1
```

This dialogue is very similar to the others. Basically, once the user says next, the whole use case ends.

Show Image WebView:

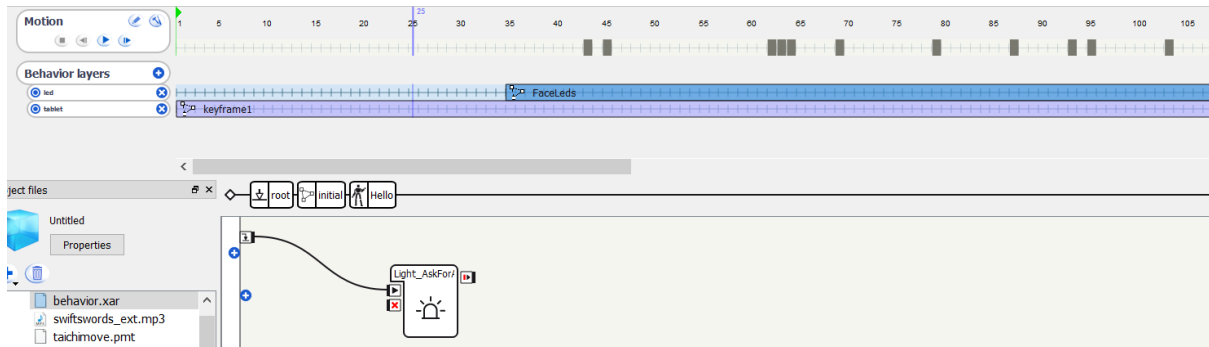
```
def onInput_onStart(self,url):
    # We create TabletService here in order to avoid
    # problems with connections and disconnections of the tablet during the life of the
    application
    tabletService = self._getTabletService()
    if tabletService:
        try:
            if url == '':
                self.logger.error("URL of the image is empty")
            if not url.startswith('http'):
                url = self._getAbsolutePath(url)
            self.logger.info(url)
            tabletService.showImage(url)
            self.onStopped()
        except Exception as err:
            self.logger.error("Error during ShowImage : %s " % err)
            self.onStopped()
    else:
        self.logger.warning("No ALTabletService, can't display the image.")
        self.onStopped()
```



The variable "showImage," which contains the name of the picture, was passed to the "onStart" of this WebView. This is necessary because the changes need to be made in the "onInput_onStart" function of the WebView. Note that the name "showImage" in the code was renamed to "url," as shown in the images above. In the inspector, a parameter called "url" was added. Finally, note that this

WebView was already implemented. All we did was delete the line `url = self.getParameter("ImageUrl")`, as the "url" was already obtained from Block2.

Hello (Animated box)



This box was created by adding behavior layers such as "led" and "tablet." However, the sound was removed since it was already implemented in the dialogue boxes. Additionally, note that in the motion, the axis represents time, and each grey rectangle corresponds to a specific motion. When all motions are executed, the robot performs the greeting (which is also similar to the farewell) motion.

Yes/no WebViews:

```

if tabletService:
    # use default robot IP address from the tablet
    robotIP = "198.18.0.1" # tabletService.getRobotIp()

    # Set page details
    pageHeading = "Ah so you are applying for electrical "
    pageText = "That is great, I am ready to help you with the proccess"
    pageImage = "electrical.jpg"

    # Set the URL
    url = "http://{ip}/apps/{appName}/pages/displayinfo.html?
pageHeading={pageHeading}&pageText={pageText}&pageImage={pageImage}".format(ip=robotIP,
appName=appName, pageHeading=pageHeading, pageText=pageText,pageImage=pageImage)

    self.logger.info(url)

    # Show the web view on the tablet
    tabletService.showWebview(url)
else:
    self.logger.warning("Couldn't find tablet service, so can't set application: %s"
% appName)

```

The picture above shows the "yes WebView." The "no WebView" is very similar and will not be elaborated on here. The important section of the code is

highlighted in the picture. First, we manually set the IP address of the robot's tablet. While we could have used the automatic method, it may cause problems during execution.

The rest is straightforward: we entered the desired values into "pageHeading," "pageImage," and "pageText." These were then passed into a variable called "url" after referencing the appropriate URL page, *displainfo.html*. Below, you can see the corresponding HTML code.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Interact with Pepper</title>
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7 <link rel="stylesheet" href="...">
8 <link rel="stylesheet" type="text/css" href="...">
9 <link rel="stylesheet" href="...">
10
11 <script src="..."></script>
12 <script src="..."></script>
13 </head>
14
15 <body>
16 <div class="w3-content" style="max-width:1100px">
17 <div class="w3-row w3-padding-64">
18 <div class="w3-col m6 w3-padding-large w3-hide-small">
19 
20 </div>
21
22 <div class="w3-col m6 w3-padding-large">
23 <div id="informationSection" class="w3-center">
24 <hr />
25 <h1 id="pageHeading" style="font-size: 45px;padding-top:50px"></h1>
26 <hr />
27 <h3 id="pageText"></h3>
28 </div>
29 </div>
30 </div>
31 <hr />
32 </div>
33 </body>
34 <script type="text/javascript">
35 displayPageInformation();
36 </script>
37 </html>
```

Note that this html was taken as is as a template and no changes were made to it.

Confirmation WebView:

```
if appName:
    if tabletService:
        # use default robot IP address from the tablet
        robotIP = "198.18.0.1" # tabletService.getRobotIp()

        # Set page details
        pageImage = "picture1.jpg"
        pageHeading = "You now have to decide"
        pageText = "Please select an option"

        # Set the URL
        url = "http://{ip}/apps/{appName}/pages/confirmation1.html?
pageHeading={pageHeading}&pageText={pageText}&pageImage={pageImage}".format(ip=robotIP,
appName=appName, pageHeading=pageHeading, pageText=pageText, pageImage=pageImage)

        self.logger.info(url)
```

This WebView is very similar to the yes/no WebViews explained earlier. The main difference lies in the HTML page we called (*confirmation1.html*), where the user can provide an input by raising an event. Below is the most important part of the HTML that we frequently modify:

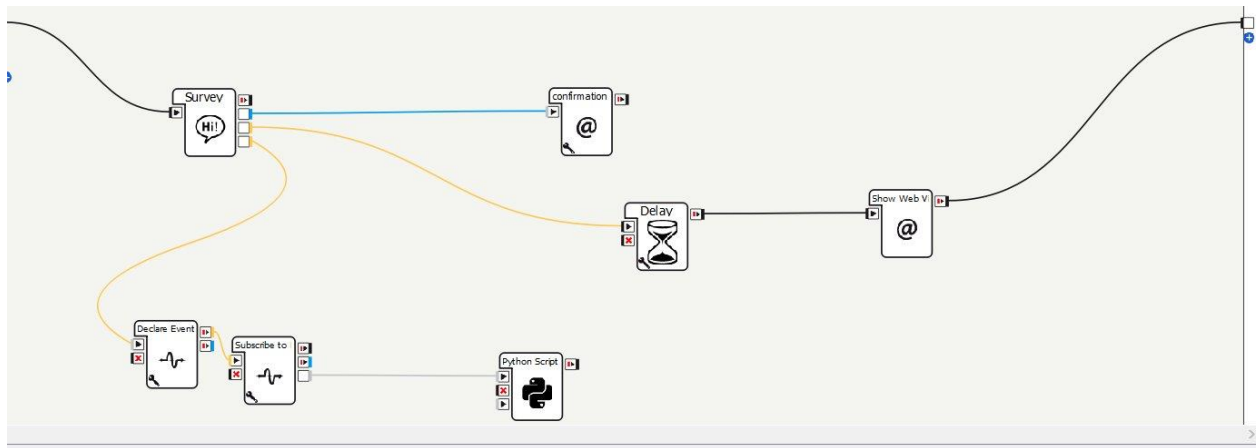
```
<button type="button" id="noButton" onclick="raiseConfirmationEvent('noAnswer')" class="fa rate-button rounded-rect-btn brown-btn" style="font-size:40px;margin:
  <span class="fa-times"></span><br>
  <span class="default-font">No</span>
</button>
<button type="button" id="yesButton" onclick="raiseConfirmationEvent('yesAnswer')" class="fa rate-button rounded-rect-btn green-btn" style="font-size:40px;">
  <span class="default-font">Yes</span><br>
  <span class="fa-check"></span>
</button>
```

Typically, what is changed is the name of the confirmation event (e.g., "noAnswer"), which is used in the dialogue box. We can also see that "No" (written in white) is what the user will see as an option to select on the screen. There are other important lines in this HTML page, such as the calls to the .js, the .css, and the padding, but these will not be discussed here.

UC-ID and Name	UC-02 [technical quiz]		
Created By	Georges saad Christian tauk	Creation Date	20/11/2024
Actors	<i>Robot(pepper) and an electrical engineer applicant seeking HR</i>		
Trigger	By the end of the initiation sequence		
Description (Objectives/Goals)	<i>Pepper starts a basic quiz asking the applicant of his skills in certain software, where the user answers the question on a scale from 1 to 5 , where we store these values to sum them up to get a final score.</i>		
Preconditions	PRECOND-1. By the end of the previous use case this use case automatically starts		
Postconditions	POSTCOND-1. The user is informed that the skill quiz is over POSTCOND-2 A webpage showing his results will appear on the screen POSTCOND-3 pepper will end this use case and send a signal to onStopped		
Action Sequence (Success Scenario)	8. Pepper informs the applicant that the skills quiz is going to start 9. Pepper starts with the first question 10. Pepper receives any answer from the applicant 11. Pepper moves to the next question 12. After the final question pepper shows the results on the tablet 13. Use case ends.		
Extensions	3a. pepper receives multiple answer ranging from 1-5: 3a1. Depending on the value answered we store the respective value		

	4a. The applicant says a press repeat 4a1. Pepper repeats the same question
Requirements	<i>Knowledge in electrical engineering concepts</i>
Storyboards	<i>Sequence of snapshots (illustrations)</i>
Priority	
Related Use Cases	Us01 and Us03
Assumptions	

Use case 02 (skills quiz and summation):



Overview of the use case

In this use case the quiz is triggered when the user says next in the previous use case, prompting the start of the quiz. Pepper will start the quiz and will wait for the user to either say the answer or press it on the WebView page seen on the tablet as seen in the fig below. After getting the answer two things will happen: one the value will be added to the previous value of the previous question of the same quiz so that at the end we would get a final score; secondly Pepper will go to the second question.

1

2

3

4

5

Repeat

Quiz dialogue box:

```
topic: ~Survey()
language: enu

concept: (scale) [1 2 3 4 5]
concept: (repeat) [sorry "i didn't understand" repeat " i didn't
hear"]
concept: (start) ^rand[start "i am ready" ready "go on" go]
concept: (Neutral) \style=neutral\ \rspd=100\ \vct=110\
concept: (joyful) \style=joyful\ \rspd=80\ \vct=95\

u: (~Start) we will start the survey now. Please answer the
following questions on a scale of 1 to 5. ^nextProposal

proposal: $softwarename="AutoCad" How familiar are you with
AutoCad?

$event=1
u1: (_~scale) $proposal1=$1 ^nextProposal $surveyAnswers=$1
u1: (e:repeat) ^sameProposal
u1: (e:oneAnswer) $proposal1=1 ^nextProposal $surveyAnswers=1
u1: (e:twoAnswer) $proposal1=2 ^nextProposal $surveyAnswers=2
u1: (e:threeAnswer) $proposal1=3 ^nextProposal $surveyAnswers=3
u1: (e:fourAnswer) $proposal1=4 ^nextProposal $surveyAnswers=4
u1: (e:fiveAnswer) $proposal1=5 ^nextProposal $surveyAnswers=5
u1: (~repeat) ^sameProposal
```

```

proposal:$softwarename="Visio"
How familiar are you with Visio?
$event=1

u1: (~scale) $proposal1=$1 ^nextProposal $surveyAnswers=$1
u1: (e:repeat) ^sameProposal
u1: (e:oneAnswer) $proposal1=1 ^goto(scoreProposal)
$surveyAnswers=1
u1: (e:twoAnswer) $proposal1=2 ^goto(scoreProposal)
$surveyAnswers=2
u1: (e:threeAnswer) $proposal1=3 ^goto(scoreProposal)
$surveyAnswers=3
u1: (e:fourAnswer) $proposal1=4 ^goto(scoreProposal)
$surveyAnswers=4
u1: (e:fiveAnswer) $proposal1=5 ^goto(scoreProposal)
$surveyAnswers=5
u1: (~repeat) ^sameProposal

proposal: ~joyful Your survey is over \pau=300\ ~Neutral lets see
how you did ^nextProposal

proposal:$score=1 $onStopped=1

```

In the dialogue box we include the concepts which are useful to tell Pepper what to wait for as input or how to say a certain speech whether in a joyful manner or neutral tone.

Pepper now will start the quiz by instructing the applicant that it started and informing them on how to answer the question. Pepper will then proceed to the next proposal which is the first question. We notice an output called software name which will be sent to the WebView page so that to create one dynamic WebView page that changes with every question. In the same proposal Pepper will ask the question that is now displayed on the tablet and the system will send another output called event which triggers the subscribe event box to start, so now the subscribe event box is waiting for a value. After that Pepper will wait for one of the inputs to be triggered whether by voice or by button and will proceed to give the correct value of the output surveyanswers that the subscribe event box is waiting for and will proceed to send this value to the python script that we will discuss later. After the steps Pepper will jump to the second question and so on, and when it finished the last question, it will say in a

joyful matter the quiz is done and that the score will be displayed in a few seconds. After that the program proceed to the onStopped output and it will trigger us03.

Html:

```
if appName:
    if tabletService:
        # Use default robot IP address from the tablet
        robotIP = "198.18.0.1"

        # Set page details
        pageImage = "picture1.jpg"
        pageHeading = "How familiar are you with
"+softwarename+" ?"
        self.logger.info(pageHeading)
        pageText = "Please select from 1-5 option"

        # Set the URL
        url = "http://{ip}/apps/{appName}/pages/
confirmation.html?pageHeading={pageHeading}&pageText={pageText}
&pageImage={pageImage}".format(
            ip=robotIP, appName=appName,
pageHeading=pageHeading, pageText=pageText, pageImage=pageImage)
```

In this figure we can see how we were able to make the WebView dynamic showing each time a different question.

In the final score WebView we had to pull the score value from the files which was different as seen in the figure below, and we had to add a new element to be inherited in the html code.

```

        return tabletService

def onInput_onStart(self):
    # Get TabletService
    tabletService = self._getTabletService()

    # Get the app name
    appName = self.packageUid()
    file = "/home/nao/answers.txt"
    score01=0
    with open(file, 'r+') as file:
        # Read the first line and store it in a variable
        score01 = file.read()

    if appName:
        if tabletService:
            # Use default robot IP address from the tablet
            robotIP = "198.18.0.1"

            # Set page details
            pageImage = "download.jpeg"
            pageHeading = "your score is "+score01+" /30 "
            self.logger.info(pageHeading)

            # Set the URL
            url = "http://{ip}/apps/{appName}/pages/
final_screen.html?pageHeading={pageHeading}&pageImage={pageImage}
&score01={score01}".format(
                ip=robotIP, appName=appName,
                pageHeading=pageHeading, pageImage=pageImage, score01=score01)

```

```

✓ function displayPageInformation() {
    document.getElementById("pageHeading").innerHTML = decodeURI(getUrlParam("pageHeading", ""));
    document.getElementById("pageText").innerHTML = decodeURI(getUrlParam("pageText", ""));
    document.getElementById("score01").innerHTML =decodeURI(getUrlParam("score01", ""));
}

```

Summation python script:

First to be able to access a file and modify it we should first create it there is two methods to do so first by logging in onto Pepper and accessing its registry via CMD or from the the Choregraph interface while connected to Pepper.

```

def onInput_onStart(self, data):
    # filename to form database
    file01 = "/home/nao/answers.txt"
    get data from file
    with open(file01, 'r+') as file:
        # Read the first line and store it in a variable
        previousSum = file.read()

        self.logger.info("old old " + previousSum)
        if(previousSum == ""):
            file.seek(0)
            file.write(str(data)+" ")
            self.logger.info("here "+ data)
            self.logger.info("log data"+ file.read())
            self.onStopped()
            return

        else: # Check if the line contains a valid integer
            file.seek(0)
            previousSum = file.read().strip()
            self.logger.info("old value "+ previousSum)
            self.logger.info(type(data))
            self.logger.info(data)
            newSum = int(previousSum) + int(data)
            self.logger.info("new sum "+ str(newSum))
            file.seek(0)
            file.write(str(newSum))

            self.onStopped()
            return

```

This code first opens the file in read plus mode which enables us to read and write on it.

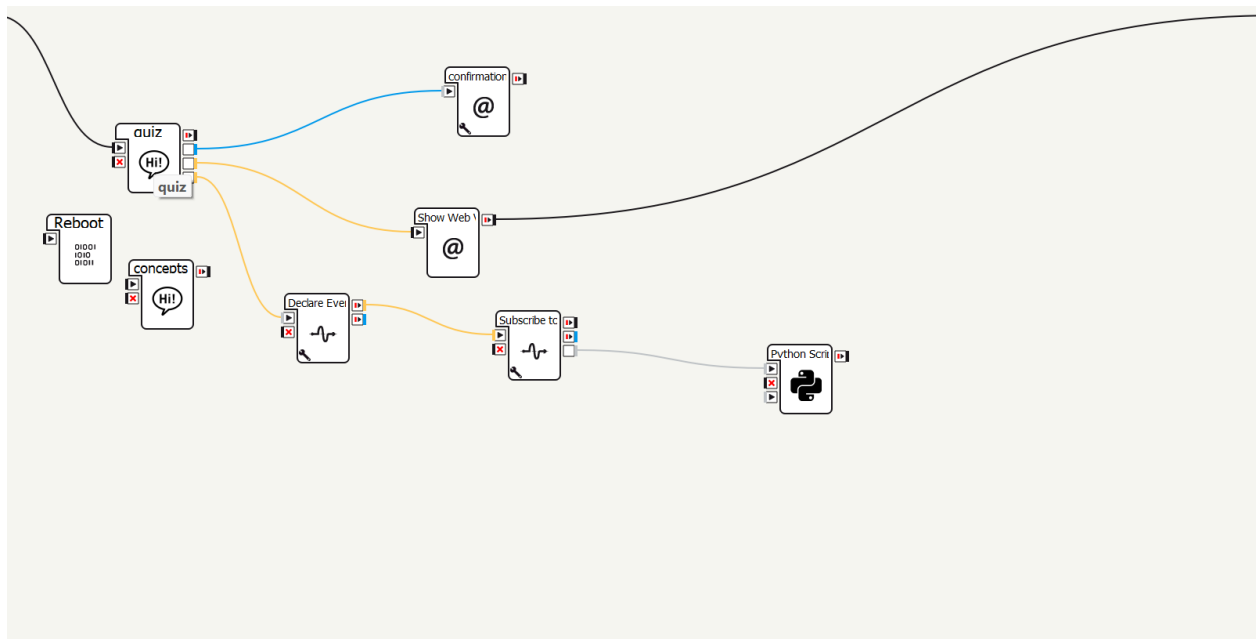
Second it checks for two conditions if the file is empty or not. Suppose at first the file is empty; we use the command `file.seek(0)` to make sure the cursor in the file is on the first position so as not to lose the value we are storing in the file.

The second time we suppose there is already a value in the file, so we set the cursor at the initial position again, pull the value, add it to the new one, return the cursor to the initial position and store the value, and so on.

UC-ID and Name	UC-03 [technical quiz]		
Created By	Maher Halabi Georges Sarkis	Creation Date	1/11/2024
Actors	<i>Robot(pepper) and an electrical engineer applicant seeking HR</i>		
Trigger	By the end of the previous quiz		
Description (Objectives/Goals)	<i>Pepper starts a technical quiz for the applicant and for each correct answer the user gets one point and 0 points on the wrong ones, this quiz is conducted to check if the applicant has the knowledge and the theoretical background that are required for this position</i>		
Preconditions	PRECOND-1. By the end of the previous use case this use case automatically starts		
Postconditions	POSTCOND-1. The user is informed that the technical quiz is over POSTCOND-2 A webpage showing his results will appear on the screen POSTCOND-3 pepper will end this use case and send a signal to onStopped		
Action Sequence (Success Scenario)	1. Pepper informs the applicant that the technical quiz is going to start 2. Pepper starts with the first question 3. Pepper receives any answer from the applicant 4. Pepper moves to the next question 5. After the final question pepper shows the results on the tablet 6. Use case ends.		
Extensions	3a. pepper receives an answer: 3a1. If the answer is right 1 point is added to the score 3a2. If the answer is wrong 0 points are added 4a. The applicant says a press repeat		

	4a1. Pepper repeats the same question
Requirements	<i>Knowledge in electrical engineering concepts</i>
Storyboards	<i>Sequence of snapshots (illustrations)</i>
Priority	
Related Use Cases	Us02 and Us04
Assumptions	

Use case 3: (technical quiz)



Use case overview:

In this use case we trigger the quiz dialog box using the “onStart” signal then in the quiz DB we start by conducting the quiz, then after each question we are sent into the confirmation WebView that shows the applicant the question written and also shows the possible answers, when the applicant chooses one answer verbally he then moves to next proposal and set a value for the variable score, or non-verbally he then raises an event which is referred at choregraph as

an event and after that he leads us to the next proposal and sets the score, and after the last question we sum up all the score values, then we move to the show WebView where this WebView shows us the total results of the applicant and then we send an onStoped signal after the result is projected on the screen.

Quiz dialog box:

```
quiz/quiz_enu.top
1 |topic: ~quiz()
2 |language: enu
3
4 |include: concepts_enu.top
5
6 |u: (e:onStart) \rspd=85\ we will start your technical quiz now. \pau=300\ It is a
  |multiple choice questions, where only one answer is right \pau=500\ ^nextProposal
7
8
9 |proposal: $question="The size of a conductor used in power cables depends on the:
  |A:operating voltage; B:power factor; C:current to be carried" The size of a
  |conductor used in power cables depends on the:\pau=300\ A:operating voltage
  |\pau=300\ B:power factor \pau=300\ C:current to be carried
10 |$event=1
11 |u1: (A) $proposal=0 ^nextProposal $surveyAnswers=0
12 |u1: (B) $proposal=0 ^nextProposal $surveyAnswers=0
13 |u1: (C) $proposal=1 ^nextProposal $surveyAnswers=1
14 |u1: (e:repeat) ^sameProposal
15 |u1: (e:aAnswer) $proposal=0 ^nextProposal $surveyAnswers=0
16 |u1: (e:bAnswer) $proposal=0 ^nextProposal $surveyAnswers=0
17 |u1: (e:cAnswer) $proposal=1 ^nextProposal $surveyAnswers=1
18 |
19 |u1: (~repeat) ^sameProposal
20
```

In the quiz dialog box, we include the concepts_enu.top file then we start first using the event “onStart” which will activate the “rspd” into 85 which the speed of the speech and we have pau which is a pause calculated in milliseconds and then we have a next proposal command which will send us to the next proposal Now in this proposal we have a variable called question this variable is then filled or set using a string this string will be holding the question that we want to ask to the applicant and the possible choices

Now the question variable is sent as an output from the quiz DB and an input to the confirmation WebView hence it activates the confirmation WebView, and it sets the page heading equal to question

After the proposal we can see that we have as u1 the following answers A,B and C these are for the verbal interface , that mean if the user answers verbally he

will activate one of them hence leading to setting the variables proposal and surveyAnswers to 0 or 1 whether the answer is correct or not and moves us to next proposal

And also, we have a repeat command that repeats the same proposal

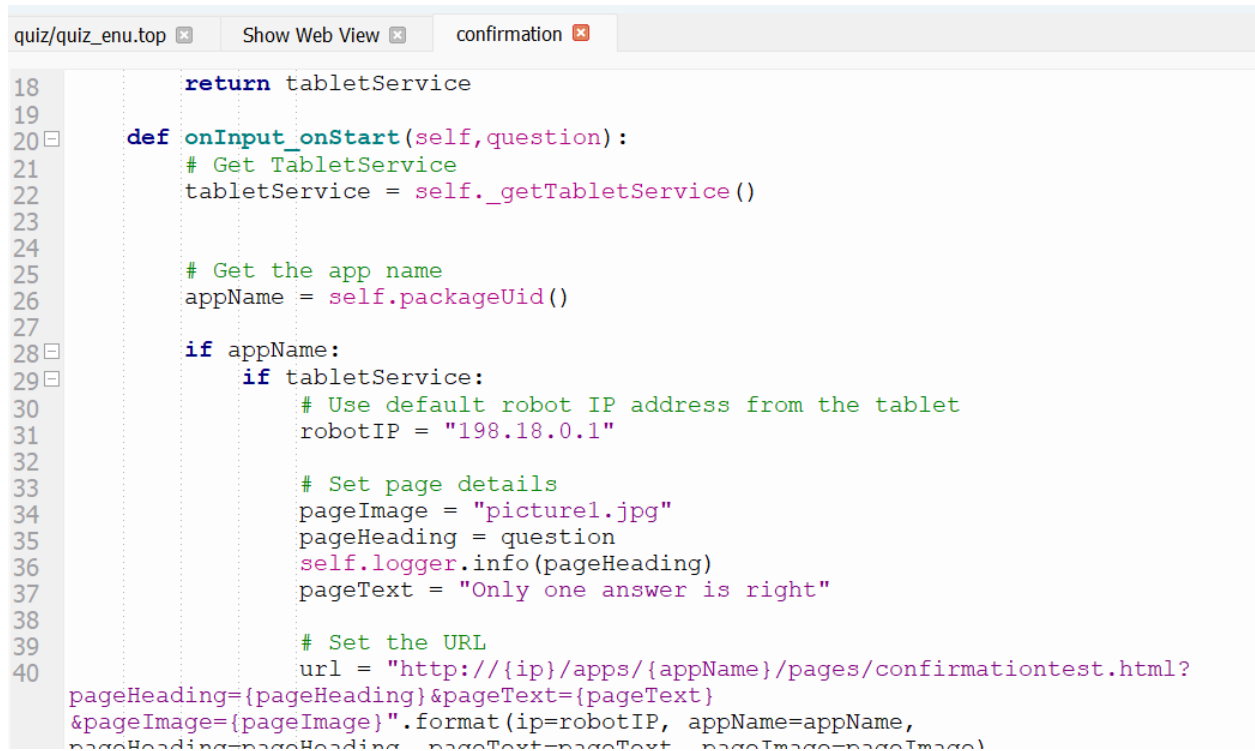
We have also u1:e:(xAnswer) this is for the non-verbal answers from the user (using the tablet) when the user presses a button on the WebView he then raises an event called xAnswer depending on which button he pressed then this event is referred to at choregraph and it is used activate u1 which will set the variables proposal and surveyAnswer to 0 or 1 and also we have the repeat command in the non-verbal answer form.

```
Script editor
quiz/quiz_enu.top  Show Web View  confirmation
64      u1:(e:bAnswer)$proposal1=1 ^nextProposal $surveyAnswers=1
65      u1:(e:cAnswer)$proposal1=0 ^nextProposal $surveyAnswers=0
66      u1: (~repeat) ^sameProposal
67
68      proposal:$question="Which type of fire extinguisher will only make the fire worse?:
    □ A:Class B; B:Class C; C:Class D" Which type of fire extinguisher will only make the
      fire worse?: \pau=300\ A:Class B \pau=300\ B:Class C \pau=300\ C:Class D
69      $event=1
70
71
72      u1: (A) $proposal1=0 ^nextProposal $surveyAnswers=0
73      u1: (B) $proposal1=0 ^nextProposal $surveyAnswers=0
74      u1: (C) $proposal1=1 ^nextProposal $surveyAnswers=1
75      u1:(e:repeat)^sameProposal
76      u1:(e:aAnswer)$proposal1=0 ^nextProposal $surveyAnswers=0
77      u1:(e:bAnswer)$proposal1=0 ^nextProposal $surveyAnswers=0
78      u1:(e:cAnswer)$proposal1=1 ^nextProposal $surveyAnswers=1
79      u1: (~repeat) ^sameProposal
80
81      proposal: ~joyful The Technical skill test is over \pau=300\ ~Neutral Let us see
    □ how you did in this whole pre-eliminatory interview ^nextProposal
82
83      proposal:$score=1 $onStopped=1
84
```

In the next proposal we set a new a string value to the variable question so it is now sent again to the confirmation WebView and now a new WebView appears at the screen with the new question and the new possible answers, after the last question we go the next proposal where we use the joyful tone to tell the applicant that the quiz is done and we have a pau of 300 MS then we move back

a neutral tone, in the last proposal we set the score variable to 1 in order to start the summation and onStopped=1

confirmation WebView:



```
18         return tabletService
19
20     def onInput_onStart(self, question):
21         # Get TabletService
22         tabletService = self._getTabletService()
23
24         # Get the app name
25         appName = self.packageUid()
26
27         if appName:
28             if tabletService:
29                 # Use default robot IP address from the tablet
30                 robotIP = "198.18.0.1"
31
32                 # Set page details
33                 pageImage = "picture1.jpg"
34                 pageHeading = question
35                 self.logger.info(pageHeading)
36                 pageText = "Only one answer is right"
37
38                 # Set the URL
39                 url = "http://{ip}/apps/{appName}/pages/confirmationtest.html?
40                 pageHeading={pageHeading}&pageText={pageText}
41                 &pageImage={pageImage}".format(ip=robotIP, appName=appName,
42                 pageHeading=pageHeading, pageText=pageText, pageImage=pageImage)
```

in the confirmation WebView we have input parameters the question variable that is set at the quiz DB and we set the pageImage to picture1.jpg that would be saved in the html/pics folder

and we set the page heading equal to the variable question so now it will be changed dynamically every time the variable questions is changed

and here we are calling confirmationtest.html file from html/pages

ConfirmationTest.html:

```
<!-- Button Row for QCM -->
<div class="button-row">
  <button type="button" id="A" onclick="raiseConfirmationEvent('aAnswer')" class="fa rate-button rounded-rect-btn green-btn" style="font-size:40px;margin-right: 60pt;">
    <span class="default-font marine-blue-text">A</span>
  </button>
  <button type="button" id="B" onclick="raiseConfirmationEvent('bAnswer')" class="fa rate-button rounded-rect-btn brown-btn" style="font-size:40px;margin-right: 50pt">
    <span class="default-font marine-blue-text">B</span>
  </button>
  <button type="button" id="C" onclick="raiseConfirmationEvent('cAnswer')" class="fa rate-button rounded-rect-btn brown-btn" style="font-size:40px;">
    <span class="default-font marine-blue-text">C</span>
  </button>
</div>

<!-- Repeat Button -->
<div class="repeat-button-container">
  <button type="button" id="repeat" onclick="raiseConfirmationEvent('repeat')" class="fa rate-button rounded-rect-btn brown-btn" style="font-size:40px;">
    <span class="default-font marine-blue-text">Repeat</span>
  </button>
</div>
</div>
</div>

<!-- Footer -->
<footer class="w3-center w3-light-grey w3-padding-32">
  <p>Powered by <a href="https://www.softbankrobotics.com/emea/en" title="PEPPER" target="_blank" class="w3-hover-text-green">PEPPER</a></p>
</footer>

<script type="text/javascript">
  // Initialize page information without pageText
  displayPageInformation();
</script>
</body>
</html>
```

in this code here we are creating the home page WebView and here we are creating 3 buttons A,B, and C each raises an event for example button A raises an event called aAnswer which is later on connected to qi-chat using java script and we are setting the style using css as if here we are using the w3.css

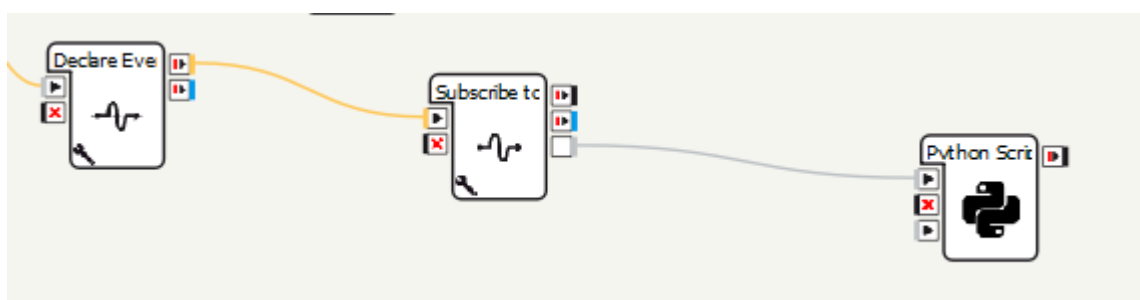
Show WebView:

```
quiz/quiz_enu.top x Show Web View x confirmation x
34 □ if tabletService:
35     # Use default robot IP address from the tablet
36     robotIP = "198.18.0.1"
37
38     # Set page details
39     pageImage = "download.jpeg"
40     pageHeading = "your score is "+score01+" /6 "
41     self.logger.info(pageHeading)
42
43
44     # Set the URL
45 □ url = "http://{ip}/apps/{appName}/pages/final_screen2.html?
pageHeading={pageHeading}&pageImage={pageImage}&score01={score01}".format(
46     ip=robotIP, appName=appName, pageHeading=pageHeading,
pageImage=pageImage, score01=score01)
47
48     self.logger.info(url)
49
50     # Show the web view on the tablet
51     tabletService.showWebview(url)
52 □ else:
53     self.logger.warning("Couldn't find tablet service, so can't set
application: %s" % appName)
54
55     self.onStopped()
```

Here in the show WebView we set the page image to “download.jpeg” and the show image to “your score is” + score01(which is the total score that we added in the summation which is a variable) + “/6” (which is the maximum score)

and we in this WebView we called final_screen.html file from html/pages folder

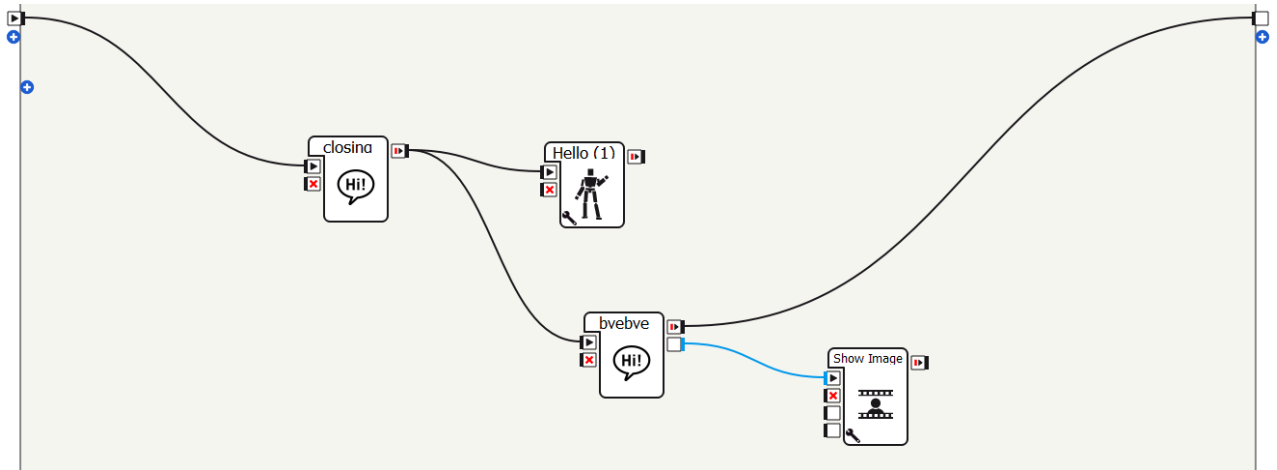
Once we are done with the show WebView we set the onStopped to one and we send it to the onStopped of the use case



This part of the use case is the python code responsible for the summation that we already spoke about and defined in the previous use case

Use case 04

In this use case pepper is just informing the user that the interaction has ended and getting feedback on the interaction and proving contact information to file any complaint and finishing by saying goodbye and waving its hand. This use case will not be explained in the report since it is similar to the other ones.



Personas:



“Motivated and Newly graduated engineer”


Demographics	Name	John El-Khoury
	Age	24
	Degree	BE in Electrical Engineering
	Years of experience	Less than 5 years of experience in the related field (can be a Fresh graduate)
	Job target	Electrical installation, focused on IPTV system
	Location	Living near the company
Motivation	Skills	Proficient in AutoCAD, REVIT, Dialux, small knowledge in Ecodial and Visio.
	Challenges	Not proficient in doing practical work due to lack of experience
	Willingness	Willing to learn new skills (technical or practical)
Social Environment	Background	Father or mother have a creative job.



“the intermediate electrical engineer”

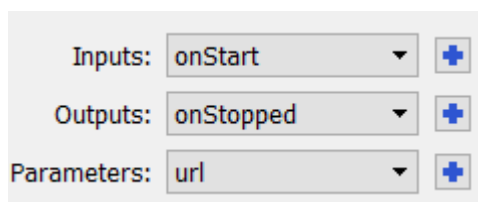
Demographics	Name	Eng. Simon Petrovich
	Age	32

	Years of experience	10
	Field	mechatronics
	Health	Good condition
Motivation	Career goals	. Build expertise in clean energy technologies to contribute to combating climate change. . Gain recognition in the field through impactful projects and innovative solutions.
	Sports	Football coach, for little league
	Experiences	Academic: Academic: <ul style="list-style-type: none"> Recently graduated from a reputable university, earning a Master's degree in Electrical Engineering with a focus on renewable energy systems. Conducted research on energy-efficient robotics during graduate studies. Industry: - Worked as a junior engineer at a mid-sized company, contributing to the development of automation solutions for industrial applications. -Part of a team that designed and implemented a small-scale solar-powered automation project. Patents: Holds multiple patents for innovative electrical engineering solutions.
	Personal challenges	Balancing Research and Practical Applications: Strives to bridge the gap between theoretical research and real-world implementation. Staying Relevant: Continuously adapts to the evolving landscape of technology and industry trends.
Social Environment	Background	Father has very long working hours while the mother stays at home to cook and care for the children.
	Parents	Retired

	“the seasoned electrical engineer”	
Demographics	Name	Dr. Eng. Anya Petrova
	Age	55
	Years of experience	20+
	Field	Power transmission
	Health	Good condition
Motivation	Career goals	Get more enhanced in the clean energy field combat climate change
	Sports	Taekwondo, Black Belt (Dan) in Taekwondo, with extensive competition experience.
	Experiences	Academic: Served as a professor at a prestigious university, specializing in electrical engineering. Industry: Held leadership positions at major corporations, overseeing large-scale projects and research initiatives. Patents: Holds multiple patents for innovative electrical engineering solutions.
	Personal challenges	Balancing Research and Practical Applications: Strives to bridge the gap between theoretical research and real-world implementation. Staying Relevant: Continuously adapts to the evolving landscape of technology and industry trends.
Social Environment	Background	Father has very long working hours while the mother stays at home to cook and care for the children.
	Parents	Retired

Challenges Faced (for the whole group):

- As an electrical engineer with a solid programming background, I was able to work efficiently with Choregraphe. However, creating completely new HTML pages was challenging for someone not familiar with HTML. I overcame this challenge by studying existing HTML templates, understanding their structure, and modifying them according to my needs.
- Another challenge I faced was transferring a picture from a dialogue box to the "show-Image" WebView. I resolved this by creating an output in the dialogue box containing the image, sending it as a string to the "onStart" function of the WebView. In the WebView, I added the variable to the existing function and updated the inspector accordingly.



Conclusion:

The development of Pepper as an HR assistant showcases the transformative potential of robotics in human-centered applications. Through this project, we explored innovative ways to enhance recruitment efficiency by integrating intuitive human-robot interaction frameworks and advanced programming techniques. By addressing challenges such as dynamic WebView integration and personalized user experiences, the project highlights the critical intersection of engineering expertise and real-world problem-solving. The successful implementation of Pepper underscores the significance of robotics in streamlining organizational processes while maintaining user-centricity. Moving forward, further refinements and broader applications of this technology could pave the way for more versatile and impactful solutions in various industries.