What we need it’s a mobile application development project for creating a hybrid Flutter app for both Android and iOS platforms.  
The primary purpose of this app is to facilitate interactions between users and a web-based SaaS (Software as a Service) workflow engine through a JSON-based API.

Key components and functionalities:

1. **Workflow Engine**: We have an existing web SaaS workflow engine that communicates with client software using a JSON-based API. This workflow engine is responsible for managing various step-by-step workflows for your clients.

2. **Hybrid Mobile App**: Our goal is to develop a hybrid mobile app using Flutter, which can be installed on Android and iOS devices. This app serves as a user interface for interacting with the workflow engine.

3. **Chat-based Interaction**: The app provides a chat-like interface where users can engage in conversations with the workflow engine. Users can initiate conversations, and the app will send requests to the server using JSON-based commands.

The user can crawl through the chat and a special character like a button, must display which button the user pressed at that point.

4. **Multilingual Support**: The app should be capable of fetching and displaying language options from the server. Users can select their preferred language, and the app will send this information to the server.

The server is responsible of sending information/questions/details in the selected language.

Flussu  
Server



request

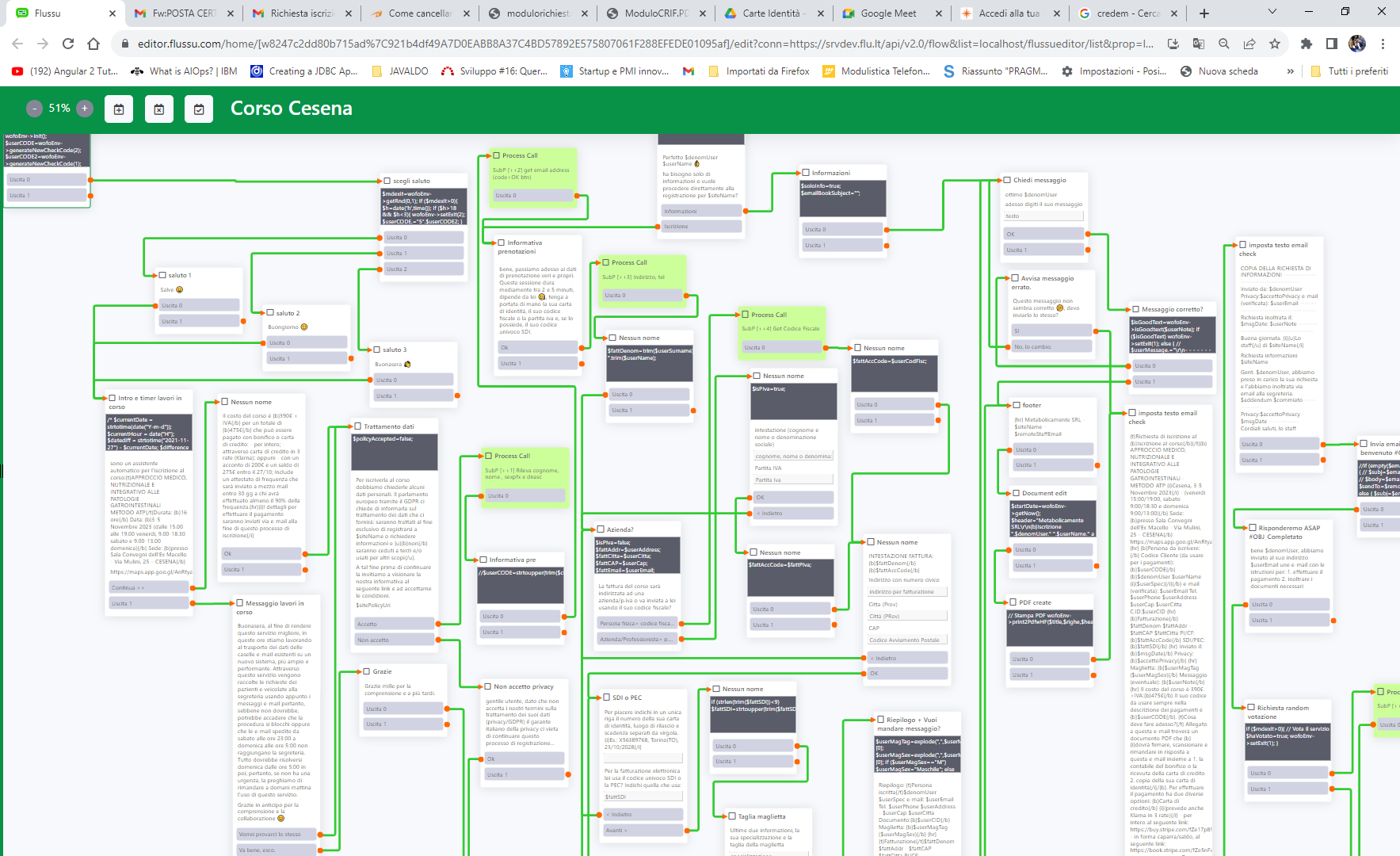
JSON

HTTP

JSON

response

5. **Step-by-Step Workflow**: The core functionality of the app involves guiding users through step-by-step workflows. These workflows can be varied and include tasks like product nutritional checks, course registration, medical appointment scheduling, etc.



6. **Camera Integration**: In scenarios like product nutritional checks, users may need to use their device's camera to capture information from product labels, such as EAN codes or nutrition facts. The app should facilitate this interaction and send captured data (such as images or scanned codes) back to the server.

7. **Data Exchange**: The app must efficiently exchange data with the workflow engine. This includes sending user inputs and receiving instructions or information from the server in response.

8. **Dynamic Content Display**: The app should dynamically display content as per the instructions received from the server. This could include showing product details, prompts for additional information, and other relevant information.

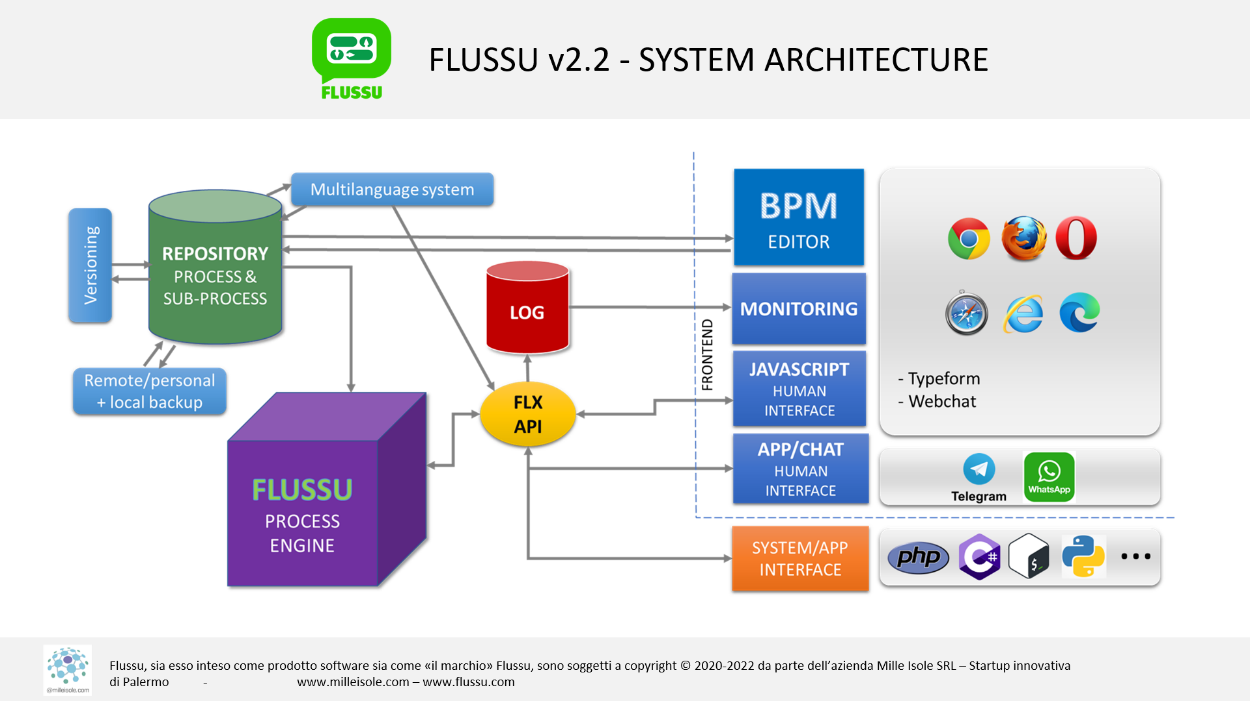
9. **Error Handling**: Proper error handling and user feedback mechanisms should be in place to handle scenarios where data cannot be processed or when there are issues with server communication.

10. **Flexibility**: The app should be designed to be adaptable to various workflow applications, not limited to just product nutritional checks.

11. **Customer identification** Sometimes information must pass an "unique client id" to put together customer informations with generic workflow informations.   
  
12. **Local storage** the app must provide a local data storage for:

1. Server address
2. workflow ID (and SessionID when working)
3. customer ID
4. entire chat display
5. User personal data (name, surnema, email, phone, etc.)

In summary, this project involves creating a Flutter-based hybrid mobile app that acts as a user interface for interacting with a web SaaS workflow engine. The app's main features include chat-based communication, language support, step-by-step workflow guidance, camera integration, and efficient data exchange with the server. It will enable users to perform various tasks and workflows, with a focus on flexibility and adaptability for different applications.

---------------------------------------------------SYSTEM ARCHITECTURE---------------------------------------------------

How the APP it must work?

1. Download and install the app
2. On the first screen the app must ask for (these will be provided to the user prior to install APP):
   1. Server name
   2. Workflow ID (WID)
3. From that moment the App will use just that server and just that WID to execute a chat from our workflow server and the user.

Sample steps:

1) User starts pressing a button for language option (says [IT] OR [EN] for italian OR english)  
2) You send to server a “START” command in JSON with the user ID and the workflow ID  
3) The server will reply a JSON MESSAGE  
4) You interpret the display need inside the JSON MESSAGE   
5) You build the display reply from the server to the user inside the app interface  
6) You get the reply form user (text, images, files, gps position, anything server asked and user response) and send back to the server in JSON  
…) goto 3

4. The App must help server/user to get photos or files or gps position if needed

5. If it is not too difficult to do at this stage (it can be a v2.0 option), in case the server needs just a code you can ask the user to shoot a photo of a code (ean code, qr-code), interpret it in the phone, extracting just the text (ex. 20-68768-267 ) and send it back to the server (just the text, not the photo). This can be needed only for EAN-Codes or QR-Codes.

**Flussu -  
Sample API Back and forth**

Let’s say we basically have two informations:   
1. Server address (servername)  
2. Workflow ID (WID)

|  |
| --- |
| Request (**GET**) |
| https://{servername}/api/v2.0/flussueng?CMD=info&WID=[w72b58f09754d44a0] |
| Response |
| {  "tit":"this is the workflow title",  "langs":"IT,EN",  "def\_lang":"IT"  } |

Pre request (and health test)

Where:

**tit**: the workflow title  
**langs**: if >1 you can choose one of this languages  
**def\_lang**: defalut language (from editor)

Start the workflow

|  |
| --- |
| Request (**POST**) |
| https://{servername}/api/v2.0/flussueng?WID=[w72b58f09754d44a0]  LNG=EN 🡨 choosen language  BID= (empty)  SID= (empty)  TRM= (empty)  APP= ***APPAND!V1.0!124574-8656756-9787987***  (App\_Id is made by: “**APP**”+{**IOS** or **AND**[roid]}+”**!**“+**version**+”**!**”+**deviceUniqueId** (see page 8)  ex.:”***APPAND!V1.0!124574-8656756-9787987***”) |
| Response |
| {  "sid":"6f056462-e437-72a7-4605-08168918b0ec",  "bid":"75a0645a-5162-2147-6561-158245d07e56",  "elms":{  "L$1":["Hi there!","class":"","display\_info":[]},null],  "ITB$0":["Go on",{"class":"","display\_info":[]}]  }  } |

where

**sid**: the session id, you need to store and use this key for every request from now on  
**bid**: the block id, is not needed at this stage/version, will be used in future  
**elms**: array of element you need to display:

* **L**abel**$1**: display “Hi there!”
* **I**nput**T**ype**B**utton**$0**: display an input button with “**Go on**” as text on the button
* **Then the user hit the “Go on” button:**

**Send action:**

Add the **sid**, the **bid**, the **wid**, and the **TRM** (terms).  
 *\* Notice, if you doesn’t need to change language, you can remove the* ***LNG*** *element from the call.*

The **TRM** is the element that contains the information needed to understand what to do in *JSON* format.

As we say, the user had hit the **$0** button (**$**ex!**0**) and the text on that button was “**Go on**”

|  |
| --- |
| Request (**POST**) |
| https://{servername}/api/v2.0/flussueng?WID=[w72b58f09754d44a0]  BID=75a0645a-5162-2147-6561-158245d07e56  SID=6f056462-e437-72a7-4605-08168918b0ec  TRM={"$ex!0":"Go on"}  LNG=EN |
| Response |
| {  "sid":"6f056462-e437-72a7-4605-08168918b0ec",  "bid":"c68d645a-5162-214a-7290-571681bf61b2",  "elms":{  "L$1":["<h1 class=\\"flussu-lbl-title\\">Privacy</h1>You must: ",{"class":"","display\_info":[]},null],  "A$2":["read and approve our policy.!|!https://www.metabolicamente.com/Privacy.html",{"class":"","display\_info":{"subtype":"button"}}],  "L$3":["Do you accept? ",{"class":"","display\_info":[]},null],  "ITB$0":["Yes",{"class":"","display\_info":[]}],  "ITB$1":["No",{"class":"","display\_info":[]}]  } } |

At this point you will know what **sid** and **bid** are. *\* Bid changes at every workflow step.*

What the meaning of that **ELMS**?

*\*Display order ($1, $2, $3, etc)*

* **L**abel **$1**: display “<h1>PRIVACY</h1>You must:”
* **A**nchor**$2**: display an anchor (you know <a href…) as an input button (display\_info":{"subtype":"button"}) with “**read and approve our policy**” as the text on the button. Notice that the two parts of information is not in json format, but “!|!” separated.
* **L**abel **$3**: display “Do you accept?”
* **I**nput**T**ype**B**utton**$0**: display an input button with “**Yes**” as text on the button
* **I**nput**T**ype**B**utton**$1**: display an input button with “**No**” as text on the button
* **User hit the “Yes” button**

**Send action:**

Add the **sid**, the **bid**, the **wid** and the **TRM** (hit the $0 button “**$**ex!**0**” and the button’s text was “**Yes**”)

|  |
| --- |
| Request (**POST**) |
| https://{servername}/api/v2.0/flussueng?WID=[w72b58f09754d44a0]  BID=75a0645a-5162-2147-6561-158245d07e56  SID=6f056462-e437-72a7-4605-08168918b0ec  TRM={"$ex!0":"Yes"} |
| Response |
| {  "sid":"6f056462-e437-72a7-4605-08168918b0ec",  "bid":"e9f8645a-5162-2158-6402-268719594ef7",  "elms":{  "L$1":["Ok I must talk at you as a male or a female? ",{"class":"","display\_info":[]},null],  "ITS$**sexTalk**":["{\"M\":\"Male\",\"F\":\"Female\"}",  {"class":"","display\_info":{"subtype":"exclusive","mandatory":true}},"[val]:[\"F\",\"Female\"]"],  "ITB$1":["Back",{"class":"","display\_info":[]}],  "ITB$0":["Forth",{"class":"","display\_info":[]}]  }  } |

What the meaning of this **ELMS**?

* **L**abel (you know)
* **I**nput**T**ype**S**election**$sexTalk**: display an input of a **selection type** (say you have a finished array of elements, from where the user must choose one or more)  
  **[“M”:”Male”,”F”:”Female”]** is the array of elements of the selection  
  **display\_info**: is HOW we want that data be displayed, it can be:
* *selection* (display a select/option dropdown, usr can choose a single element)
* *multiple* (display options as radio buttons, usr can choose a single element)
* *exclusive* (display options as check buttons, usr can choose more than one elements)

“mandatory”:”true” 🡪this input element is mandatory.

**“[val]”**: let’s say you have already used that field, and already choosed some option. So in the “[val]” array you can find the value you must display as selected (is the same for text input. In this case there is no array, but a single value)

* **I**nput**T**ype**B**utton**$\_\_**: you know
* **So the user choose M (male) and press the “Forth” button**

**Send action:**

the new data to be sent to the server will be: **sid**, **bid**, **wid**, and the **TRM**.

|  |
| --- |
| Request (**POST**) |
| https://{servername}/api/v2.0/flussueng?WID=[w72b58f09754d44a0]  BID=75a0645a-5162-2147-6561-158245d07e56  SID=6f056462-e437-72a7-4605-08168918b0ec TRM={"$ex!0":"Forth", "$sexTalk":"@OPT["M","Male"]"} |

|  |  |
| --- | --- |
| TRM | "$ex!0":"Forth" is the button pressed |
| "$sexTalk"🡨(this is the variable name) :"@OPT["M","Male"]"🡨(this is the variable value)  *(user choose the* ***OPT****ion for ”Male”)* |

Let’s talk about the values needed back: the workflow engine need to receive every part of the input, like the Option code and the Option text as well.

**GET USER INFO FUNCTION**

When needed the server will need user data.

The APP must register user information such as Name, Surname, Phone Num, Email and so on.

When these data are needed by the server, it send a “GUI” (Get User Info) request, as following:

|  |
| --- |
| Request (**POST**) |
| *(any)* |
| Response |
| "sid": "769e6545-13e4-4a81-6511-823262678691",  "bid": "640a6543-f384-4eeb-f160-185246e1aee0",  "elms": {  "L$2" : ["user info required ",{"class": "","display\_info": []},null],  "ITB$0": ["Ok", {"class": "","display\_info": []}],  "**GUI**$**userinfo**": ["","","[val]:"]  }  } |

If you receive this type of request, you must ask user if he want to send their data (privacy matter), and if yes, send it as json like the following post request. If the user does not want to send their data, you must packet as a denied post.

**Accepted Post**

|  |
| --- |
| *Server, BID, SID, (etc..)*  TRM={"$userinfo":["name":"john","surname":"doe","email":"johndoe@gmail.com","consent":"(date yyyy/mm/dd hh:mm:ss)"]} |

**Denied Post**

|  |
| --- |
| *Server, BID, SID, (etc..)*  TRM={"$userinfo":["consent":"none"]} |

**SAMPLES UI/UX**

|  |  |
| --- | --- |
| **AFTER INSTALL**  After first installation the APP needs a “startup code”.  It can be a QR-Code or a code sent by the company who ask the user to use his app.  If it is available, the first call must report the default language code used by the user, so this help us to choose the default language without asking to the user, if it is one of the languages defined by the workflow designer…  The APP must have a “device ID”, please build a UUID like this <https://www.uuidgenerator.net/> | Phone template Vectors & Illustrations for Free Download ...  Shoot photo  Get from File  CODE  OK |
| *First Installation* | |

|  |
| --- |
| Request (**GET**) |
| https://v20.flussu.com/api/appcode?{theGivenCode} |
| Response |
| {  api\_code: AL432AFX,  server: srvdev.flu.lt, (this is SERVER NAME)  wid:[wd2e0a737dfbab1a3], (this is the WORKFLOW ID [WID])  languages:{  IT:Italiano, (Workflow’s available  EN:English languages)  },  title:{ (workflow’s title)  IT:Questionario Medigenium  EN:Medigenium questionaire  },  customer:{ (customer’s data section)  logo:https://www.medigenium.com/wp-content/uploads/2023/10/medigenium\_logo.png,  title:Medigenium,  website\_url:  {  EN:https://www.medigenium.com/,  IT:https://www.medigenium.com/  },  text\_whoweare:  {  EN:Medigenium is an italian start-up who works in the PACS market,  IT:Medigenium \u00e8 una startup italiana che lavora nel campo dei PACS  },  privacy\_url:{  EN:https:\\www.medigenium.com/privacy-policy/,  IT:https:\\www.medigenium.com/privacy-policy/  }  menu\_labels:{  EN:{RESTART,PRIVACY,WHO WE ARE,USER INFO}  IT:{RICOMINCIA,PRIVACY,CHI SIAMO,I TUOI DATI}  }  },  update\_date:2023/10/01,  valid:  {  from:2023/01/01,  until:2023/12/31  }  } |
| *Very first application code request* |

**Available codes for testing: AL432AFX** or **FE162CR3** (multiple fields test)

The code will provide the following data:

1. ***server***, is the server address where you need to send the http workflow requests
2. ***wid***, it is the workflow ID you need to send at each request to the server
3. ***customer[logo]***, the company or workflow logo for the App
4. ***customer[privacy\_url[LANG]]***, the privacy contract in the available language

When APP receives the above data, and must register in it’s own cache and populate the display using the labels from choosen language.

|  |  |  |
| --- | --- | --- |
| **AFTER INITIALIZED**  After the first call the user choose language and can see the privacy contract and accept (or not)  If accepted the Workflow can start | | Phone template Vectors & Illustrations for Free Download ...  ENGLISH  See privacy  **MEDIGENIUM QUESTIONAIRE**  Accept privacy |
| *Privacy acceptation* | | |
| There will be a **OPTIONS DISPLAY** where the app will show the customer’s logo and name, and there will be the following options:   1. Restart a workflow (deletes Session ID and ask the server again to start, see page 9 of this document) 2. Go to the privacy page (open a browser and show it) 3. Go to the “who we are” page on the customer’s website (on ext. Browser) 4. Open a local storage where the info are recorded, like name, surname, email, user avatar/photo These datas are optional, if they want a user can stay in a complete privacy state   *These info and these labels can be sent from the server at the first call (see page 8 and 9 of this document) but we need to have a default english text in case of any malfunction*   1. A “technical assistance info” that can help us to solve techincal issues, must report the following data: 2. Server address 3. Workflow ID 4. Session ID (if any) 5. Device ID  * *two {red/green} led:*  1. “L” If line is active 2. “S” If the server (a.) is reachable 3. App version 4. Operating system 5. A “factory reset” button that put the application to the initial/first installation state as you see in page 8 of this document (delete the cache **but do not delete the device ID**). | Phone Menu Options  RESTART  Do not use exactly these icons, there are just as example. They must be the correct icons like a user icon, a restart icon, a factory icon, etc.  The tech one can be three dots  Factory reset  TECH INFO  MEDIGENIUM SRL - ITALY  WHO YOU ARE  WHO WE ARE  PRIVACY | |
| *Options menu* | | |

**Language choice considerations:**

Upon first use, you request a CODE (*page 8 and 9* - *Very first application code request*) to obtain from the server all the information about the workflow to be used from now on.

At this first level, you will receive a list of languages and labels to use to populate text, links, information, etc.

From this point forward, the user will use the APP connected to a given workflow (WID).

Upon the first execution of these two steps, the available languages will be exactly the same, so if a user chooses English at the very first step, English will also be used in the workflow.

Over time, it may happen that the workflow increases the number of available languages. If this happens and the user asks to start over, and the number of languages has changed, you should ask the user to choose one, with the option to choose the new language now available.

In any case, when the user selects a language, the command on *page 8 and 9* (*Very first application code request*) must be executed again because it is necessary to modify the app interface using the language chosen by the user.

SINGLE ELEMENTS SAMPLES

**BUTTON (S)**

To have a single choice of some options, we can use more than one button

**Choose me**

**Hit me**

**Click me**

|  |
| --- |
| **Selection** (dropdown) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "To have a single choice of some options, we can use more than one button",  {class: "",display\_info: []},null  ],  **"ITB$0**": [  Display order  "Choose me",  {class: "",display\_info:[]}  ],  **"ITB$1**": [  "Hit me",  {class: "",display\_info:[]}  ],  **"ITB$2**": [  "Click me",  {class: "",display\_info:[]}  ],  }  } |

**(AFTER)**

To have a single choice of some options, we can use more than one button



Hit me

**TEXT INPUT**

Please write your name



Hi there!

|  |
| --- |
| **Text input** (default, not mandatory) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Please write your name",  **$info** is the variable name  {class: "",display\_info: []},null  ],  **ITT$info**: [  "Hi there!",  it’s mandatory?  {  class: "",  display\_info: {  mandatory: false,  Input types:  **default** (one row)  **textarea**  **email**  **date**  **gps position**  subtype: "default"  }  },  [val]:""  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**(AFTER)**



Hi there!

Please write your name

**SELECTION (Dropdown)**

Normal selection (with dropdown)

Test

Toast

Taste

Toast

|  |
| --- |
| **Selection** (dropdown) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Normal selection (with dropdown)",  Default / selected  If VALUE is null  {"class": "","display\_info": []},null  ],  **ITS$sel**: [  {  "Z3,0":"Test",  "B5,**1**":"Toast",  "4H,0":"Taste"  },  {class: "",display\_info: []},  [val]:[]  ],  ITB$0: [  "Ok",  {class: "",display\_info: []}  ]  }  } |

**(AFTER)**



Toast

Normal selection (with dropdown)

**SELECTION (Exclusive)**

DO EXACTLY WHAT ALREADY DO FOR SINGLE BUTTONS (see prev. Page 13)

|  |
| --- |
| **Selection** (dropdown) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Exclusive selection (radiobuttons)",  Default / selected  If VAL is null  {class: "",display\_info: []},null  ],  **ITS$sel**: [  {  "1Z,0":"Test",  Display mode  "20TR,**1**":"Toast",  "BT1,0":"Taste"  },  {  class: "",  display\_info: {subtype: "exclusive"}  VAL=Actual value  },  **[val]:["20TR","Toast"]**  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**SELECTION (Multiple)**

Multiple selection (checkbuttons)

☑



**Test**

**Toast**

☑

**Taste**

|  |
| --- |
| **Selection** (dropdown) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Multiple selection (checkbuttons)",  Default / selected  If VAL is null  {class: "",display\_info: []},null  ],  **ITS$sel**: [  {  "CA,0":"Test",  Display mode  "CB,**1**":"Toast",  "CT,0":"Taste"  },  {  class: "",  display\_info: {subtype: "multiple"}  },  **[val]:[**  VAL=Actual value(s) code(s)  **"CA","CT"**  **]**  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**(AFTER)**

☑ Test

☑ Taste

Multiple selection (checkbuttons)



**MEDIA (image display)**

This text is on top

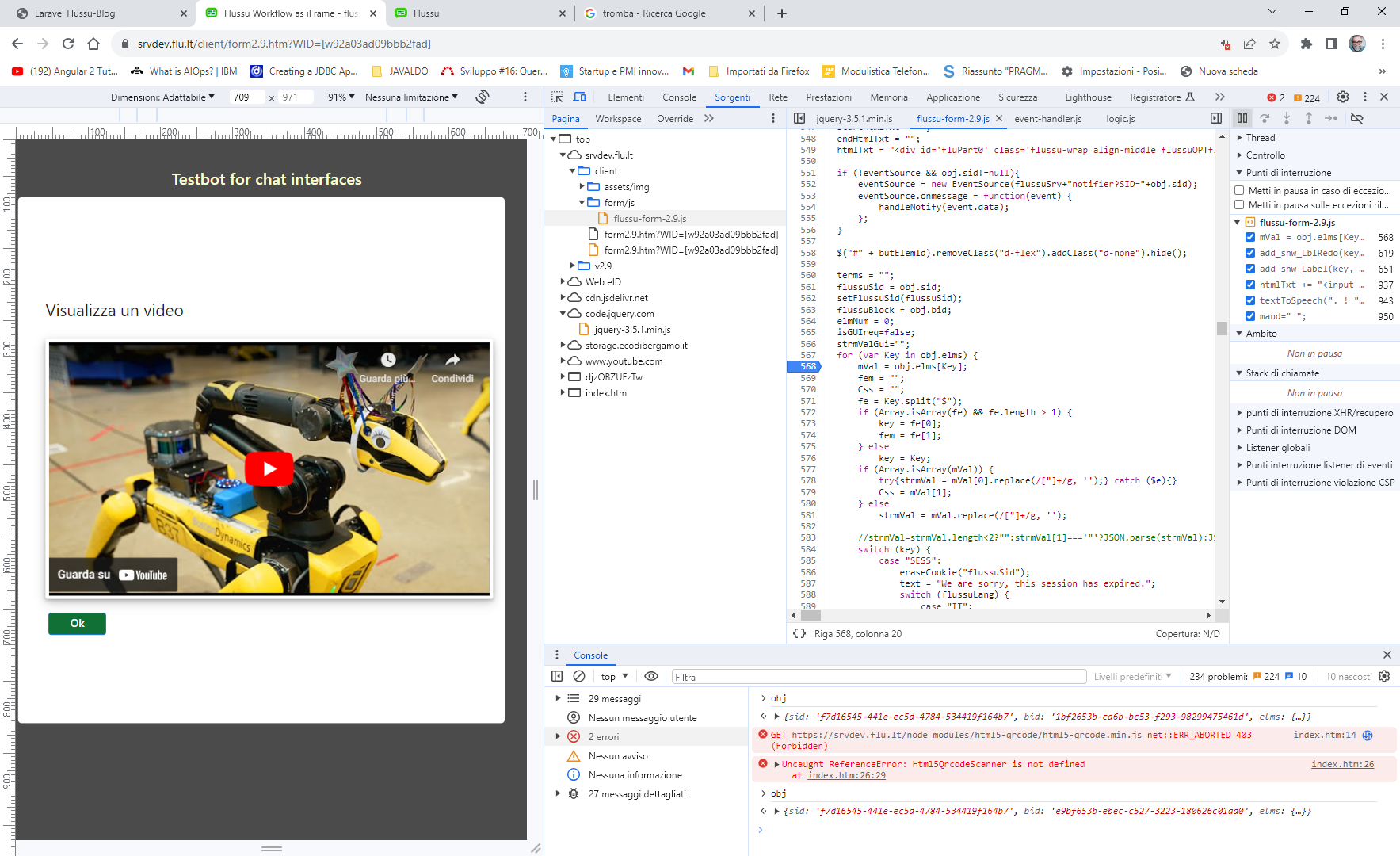


This text is under the photo



|  |
| --- |
| **Media display** (image) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "This text is on top",  {class: "",display\_info: []},null  ],  **M$2:** [  "https://wwwmywebsite.con/media/photo\_1006.jpg",  {  Media display type  class: "",  display\_info: {  type: "image"  }  }  ],  L$3: [  "This text is under the photo",  {class: "",display\_info: []},null  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**MEDIA (video display)**



In this case the routine read that is a youtube video then display a preview iframe.

In case of link, we can’t know in advance what media type is that.

This text is on top

This text is under the video



|  |
| --- |
| **Media display** (media) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "This text is on top",  {class: "",display\_info: []},null  ],  **M$2:** [  " https://youtu.be/djzOBZUFzTw",  {  Media display type  class: "",  display\_info: {  type: "file"  }  }  ],  L$3: [  "This text is under the video",  {class: "",display\_info: []},null  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**LINK (S)**

**"ITB$0**": [

"Choose me",

],

Want to see a special website? Come to:

# [**https://www.milleisole.com**](https://www.milleisole.com)

Please visit

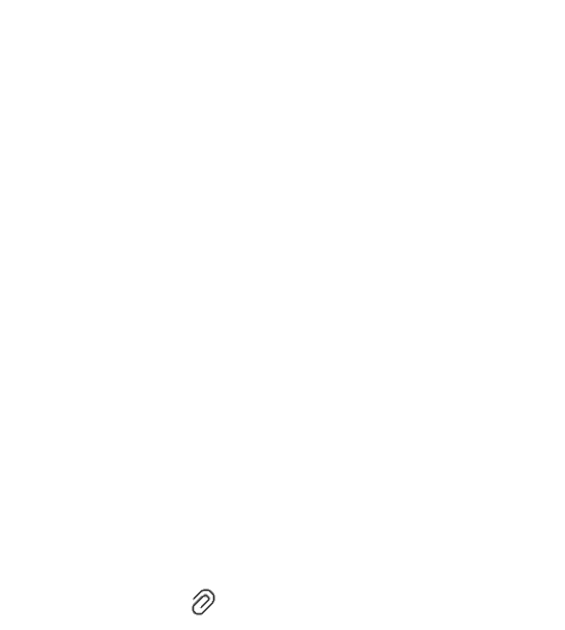
**MY WEBSITE**



|  |
| --- |
| **Anchor** (link) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Want to see a special website? Come to:",  {class: "",display\_info: []},null  ],  Button text  Separator  Link  A$2: [  "https://www.milleisole.com",  {class:"",display\_info:[]}  ],  L$3: [  "Please visit",  {class:"","display\_info":[]},null  ],  A$4: [  "MY WEBSITE**!|!**https://www.milleisole.com",  Link display type  {class: "",  display\_info: {subtype: "button"}  }  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**FILE INPUT (standard)**

Please select a FILE



|  |
| --- |
| **Text input** (default, not mandatory) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Please select a FILE",  **$myfile** is the variable name  {class: "",display\_info: []},null  ],  **ITM$myfile**: [  "Hi there!",  it’s mandatory?  {  class: "",  display\_info: {  File types:  **(none)** (any)  **read-code** get EAN/QR-CODE  (see next page)  **file-image**  **file-document**  mandatory: false,  subtype: ""  }  },  null  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**(AFTER)**

In case of MEDIA 🡪 In case of DOCUMENT





Please select a FILE

Photo 12687398.jpg

Document\_JGHTAFYT\_.pdf

Please select a FILE

If the user clicks on that photo or document, ask to open it with internal app used to open that type of file.

How to prepare TRM for file upload:

1. ***Create the input control and the event***

|  |
| --- |
| <input id="img\_" + inputId + " name=$" + inputId + " type='file' accept=images/\* class="flussu-inpFile I" + flussuBlock + " onchange="getPhoto(this,'img" + flussuBlock + "',true);" > |

That transforms as (example)

|  |
| --- |
| <input  id="img\_myFile"  name="$myFile"  type="file"  accept="images/\*"  class="flussu-inpFile I4356-23456-33442-344545"  onchange="**getPhoto**(this,'img4356-23456-33442-344545',true);"  >  </input> |

1. ***Read the file***

For example in javascript we can do:

|  |
| --- |
| function **getPhoto**(inp, prew, adj) {  if (inp.files && inp.files[0]) {  var reader = new FileReader();  reader.readAsDataURL(inp.files[0]);  }  } |

This read transfer the bytes of the image on the “src” value of the ***<img*** tag and encode it as a base64 encoding byte stream.

1. ***Prepare TRM***

The file parameter in TRM specs are the following:  
a. the filename  
b. the data (get the SRC part of the ***<img*** tag that is base64 encoded)

|  |
| --- |
| "$myFile\_name":"**myPhoto123.jpg**",  "$myFile\_data":"data:**image**/**jpeg**;**base64**,/9j/4AAQSkZJRAAD/2wCEAAU […] wBK/9k=" |

1. ***Complete TRM sample as JSon:***

|  |
| --- |
| '{  "$ex!0":"Ok",  "$**myFile**\_name":"myPhoto123.jpg",  "$**myFile**\_data":"data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAUDBA  […]  IBwcHBw8LCwkMEQ8SEhEPERETFhw RERGCEYGh0dHx8fExciJCIeJBwe  Hx4BBQUFBwYHDggIDh4UERQeHh4e h4eHh4eHh4eHh4eHh4eHh4eHh4eH  h4eHh4eHh4eHh4eHh4eHh4eHh4eHv5Ad/wBK/9k="  }', |

1. **FILE INPUT (read code)**

If it is requested to get an **image file**, open the phone folder for the photo /images file type.

If it is requested to get a **document file**, open the phone folder for the document file type.

If the server needs a **code** (subtype: "**read-code**"), you can open a frame window to display the camera aim the user to identify the code area and get a code photo.

If you can try to read the code in the phone. If you does not retrieve code or you can’t get a code as text, send the whole photo back to the server.

Please read a code





|  |
| --- |
| **Text input** (default, not mandatory) |
| {  sid: *(session\_id UUID),* bid: *(block\_id UUID),*  elms: {  L$1: [  "Please read a code",  This is the variable name  {class: "",display\_info: []},null  ],  **ITM$codeTextOrPhoto**: [  "Hi there!",  it’s mandatory?  {  class: "",  display\_info: {  mandatory: false,  The server ask for a code  subtype: "read-code"  }  },  null  ],  ITB$0: ["Ok",{class: "",display\_info: []}]  }  } |

**(AFTER)**

In case send code image 🡪 In case read CODE and send text



Please read a code

8718692059603

Please read a code

