

International Project Management

ITCPS2 Group 2



Aninda Maulik
Mohamadnour Badr
Morad Benkaraache
Dhayananth Dharmalingam

Acknowledgement

We would like to thank Mr, Piere Marrer and Olivier Boissier who guided throughout this project in order to complete this project. We also giving our thanks to the team member who worked hard to achieve our milestone.

Aninda Maulik

Mohamadnour Badr

Morad Benkaraache

Dharmalingam Dhayananth

Abstract

Projects play key roles in most modern industries and firms. The management of these economic activities, project management, is continuously developed and today considered to be at the center of competitive advantage. Much classic research on project management has, however, focused on the planning and scheduling activities of project management. Traditional writings within the area even seem to treat project management as a discipline of planning or an application of systems analysis. Based on the empirical observations, we elaborate on a framework for the analysis of project management work where two perspectives are put at the fore; knowledge perspective and time perspective. From these perspectives, we discuss different roles that project management has in a product development context.

We have been demonstrated Scrum project management methodology in ITCPS2 Workshop2 for the purpose of managing the project. All the properties and approaches of Scrum methodology has been followed and presented. In this report we explain detail information about the architecture and the functionalities of the Workshops that has been assigned to our group.

This document can be referred in future for further enhancement and improvement of the application. We explained all the information that is necessary to improve the application in future by another team.

Table of Contents

Acknowledgement	1
Abstract	2
Introduction	5
Our Stack	7
Front End Technologies	7
Server API Stack	7
Sequence Diagram	8
Filling workshop sequence diagram	8
GUI – 1 and Filling workshop Communication	8
GUI – 2 and Filling workshop Communication	9
Potting Workshop sequence diagram.....	10
GUI – 1 and Potting workshop Communication.....	10
GUI – 2 and Potting workshop Communication.....	11
Overview of Filling Workshop	12
Technical overview of Filling Workshop	12
Code sample of filling workshop.....	13
GUI 1 for Filling Workshop	16
GUI – 2 filling workshop.....	17
GUI 1 – Screen.....	18
Overview of Potting Workshop	19
Technical overview of Potting Workshop	19
GUI – Potting workshop	22
GUI – 2 potting workshop	24
Conclusion	25

Table of Figures

Figure 1: Global Use case	5
Figure 2: Architecture of the workshop	6
Figure 3: Stack of the application	7
Figure 4: GUI-1 and Filling workshop communication sequence	8
Figure 5: GUI 2 and Filling workshop communication sequence	9
Figure 6: GUI-1 and potting workshop communication sequence	10
Figure 5: GUI 2 and potting workshop communication sequence	11
Figure 7: API Socket connect code - Filling GUI 1.....	16
Figure 8: Handle message function - filling interface	17
Figure 9: User interaction form - filling workshop.....	17
Figure 10: GUI-1-Filling workshop screen.....	18
Figure 11: API Socket connect code - Potting GUI 1.....	23
Figure 12: Potting workshop GUI.....	24

Introduction

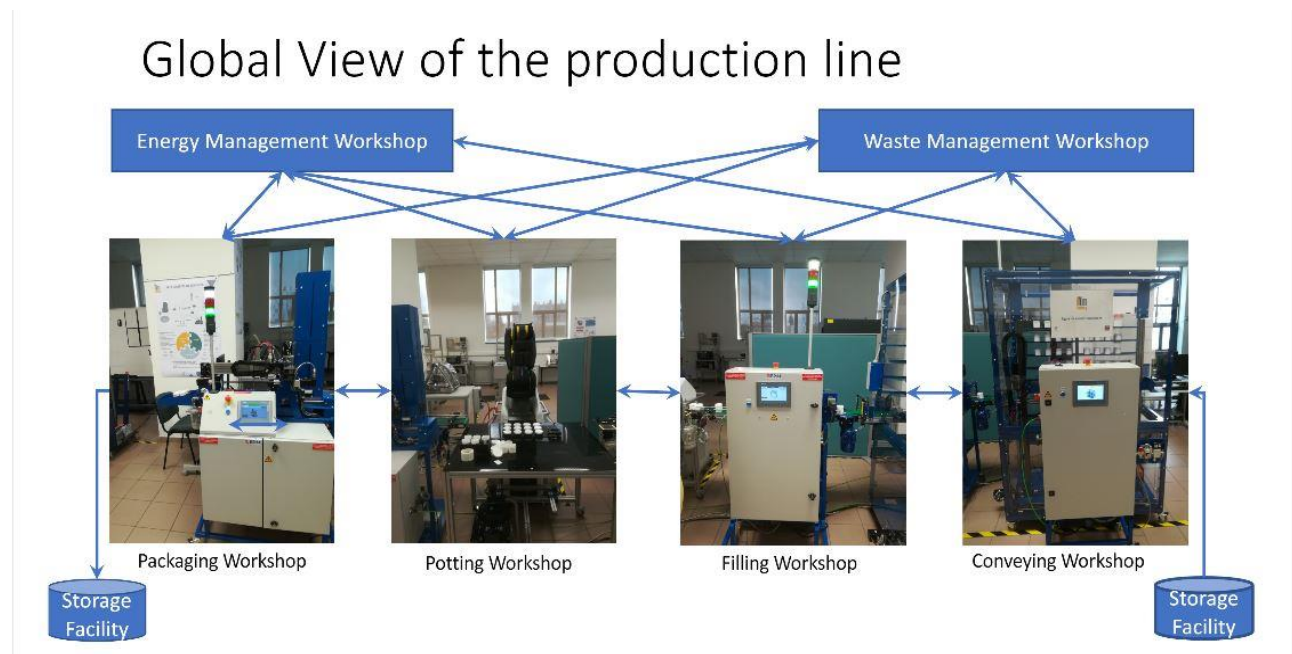


Figure 1: Global Use case

Production workshop is comprised of four different workshops which are Conveying Workshop, Filling Workshop, Potting Workshop and Packaging Workshop. Also, it connected to two other supporting service workshops, which are waste transformation and energy production system.

Conveying workshop collects empty contains from storage and send it to the filling workshop. Filling workshop fills the empty containers with material and it to the potting workshop. Potting workshop will close the containers and send to the packaging workshop. Packaging workshop will package the containers in to boxes.

Each workshop consumes energy and produces waste. These part will managed by supporting services which are energy and waste management service.

This flow will done automatically and continuously one after the other. The conveying workshop will continuously picks containers one after the other and keep send it to the next workshop and so on.

Our goal is to simulate the same process with our application by developing API application and make interconnection between them.

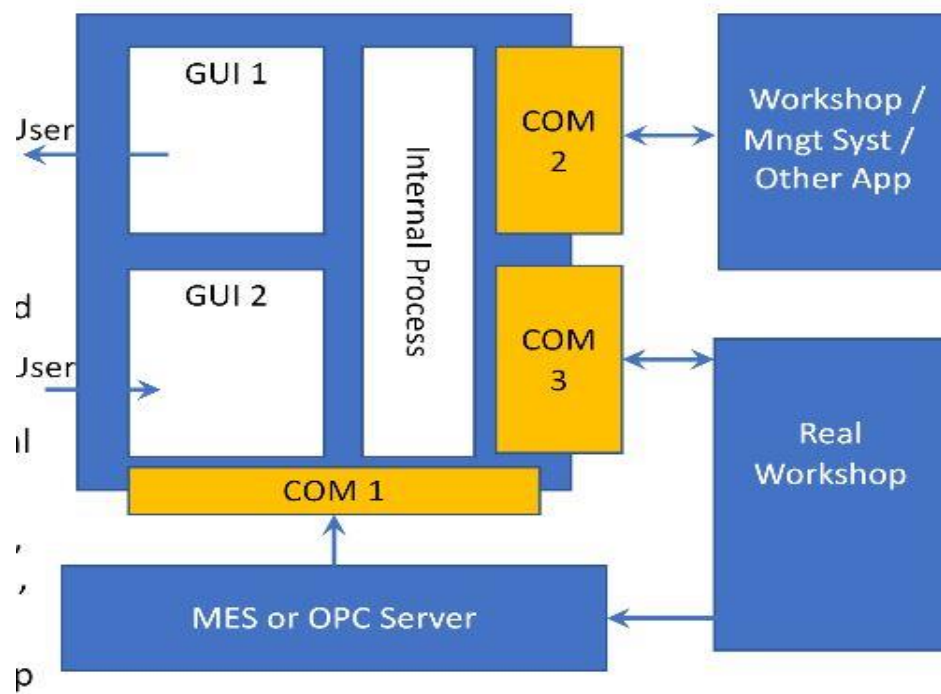


Figure 2: Architecture of the workshop

- GUI1 will be used in order to visualize the parameter that has been delivered by each workshop. Each workshop will use this interface to display the status of the application and to display inter function process with different output parameters. Ex: information sent to other workshop, information received from other workshop
- GUI2 will be used to perform user interaction to the internal function of each workshop. Allows to send values to the internal process and to perform some function.
- Internal process is the simulation of process that is corresponding to the workshop. •It will process some function and output new parameter values.
- COM1 is communication interface with the MES or OPC server that gather data (parameter values) from the real workshops.
- COM2: communication interface with other workshops, the management systems (energy, waste management), other App. COM2 is a Web API.
- COM3: communication interface with the real workshop

Our Stack

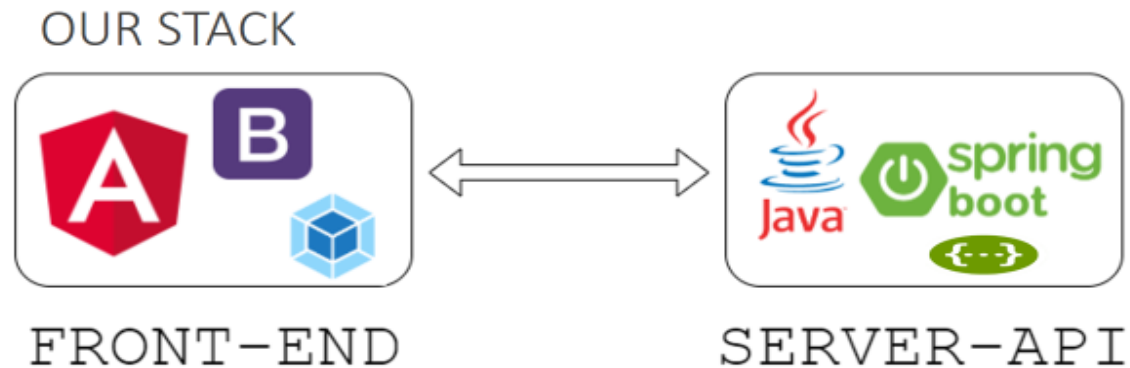


Figure 3: Stack of the application

Front End Technologies

- Angular 8
 - We are using Angular framework to develop our client side web interface. This framework fully support TypeScript.
- Bootstrap
 - Also we use Bootstrap for design our interface with good looking components.
- Webpack
 - Finally we use webpack to bundle our JavaScript application. This tool will do the job of “uglifying” of “minifying” process.

Server API Stack

- Java – Spring Boot
 - We use Spring boot for developing server application with Rest API. This is a framework that developed using Java programming language.
 - We use Swagger for API Documentation.

Sequence Diagram

Filling workshop sequence diagram

GUI – 1 and Filling workshop Communication

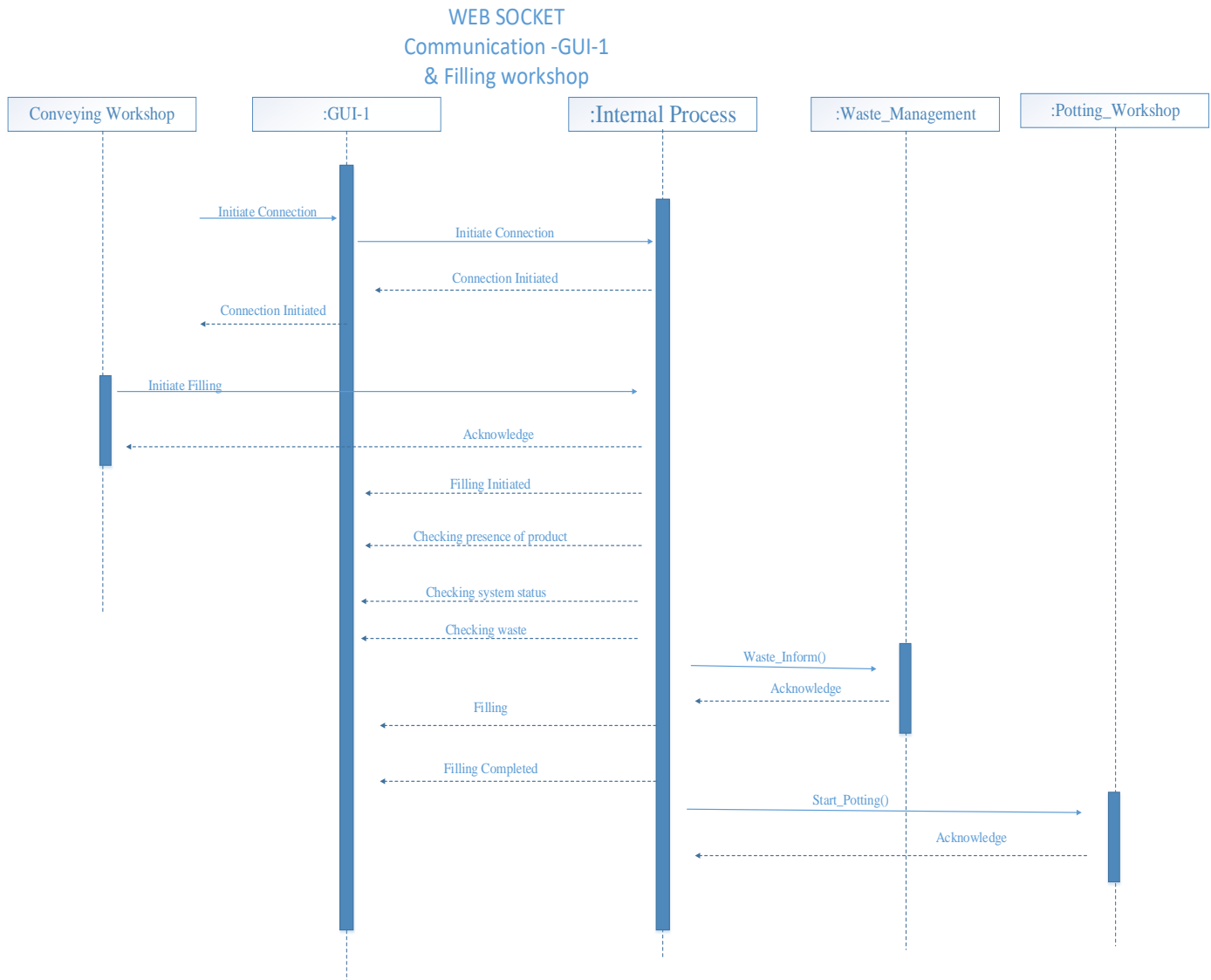


Figure 4: GUI-1 and Filing workshop communication sequence

Above diagram shows communication between GUI 1 and Filling workshop. The communication initiated using web socket and it will live constantly through the process. The filling initiate function will be start soon after conveying workshop sends a request. Waste management will be informed if any waste produced. At the end the potting workshop will be informed to initiate potting process.

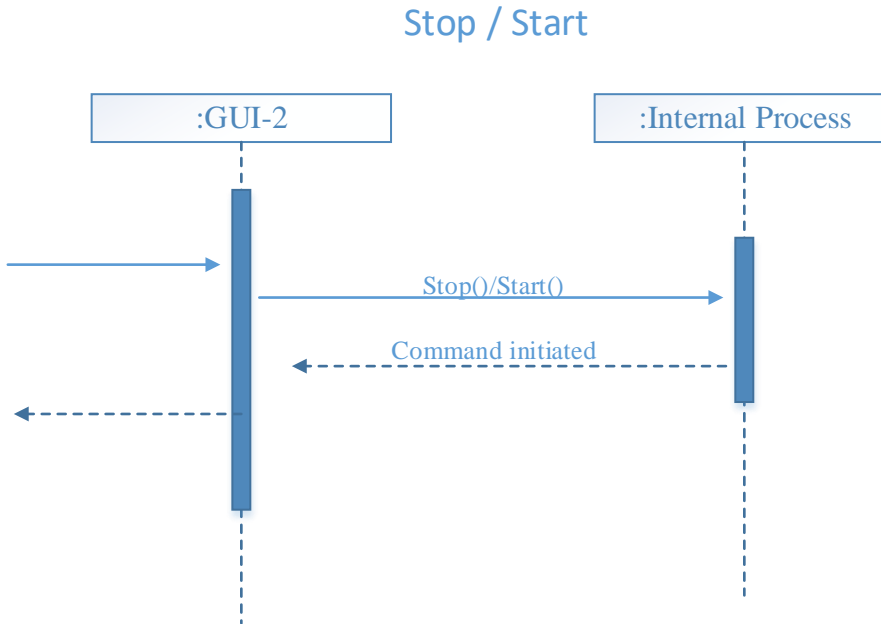
GUI – 2 and Filling workshop Communication

Figure 5: GUI 2 and Filling workshop communication sequence

Above sequence diagram shows communication between GUI-2 and internal process of filling workshop. User can trigger stop and start commands and system will react to those commands accordingly.

Potting Workshop sequence diagram

GUI – 1 and Potting workshop Communication

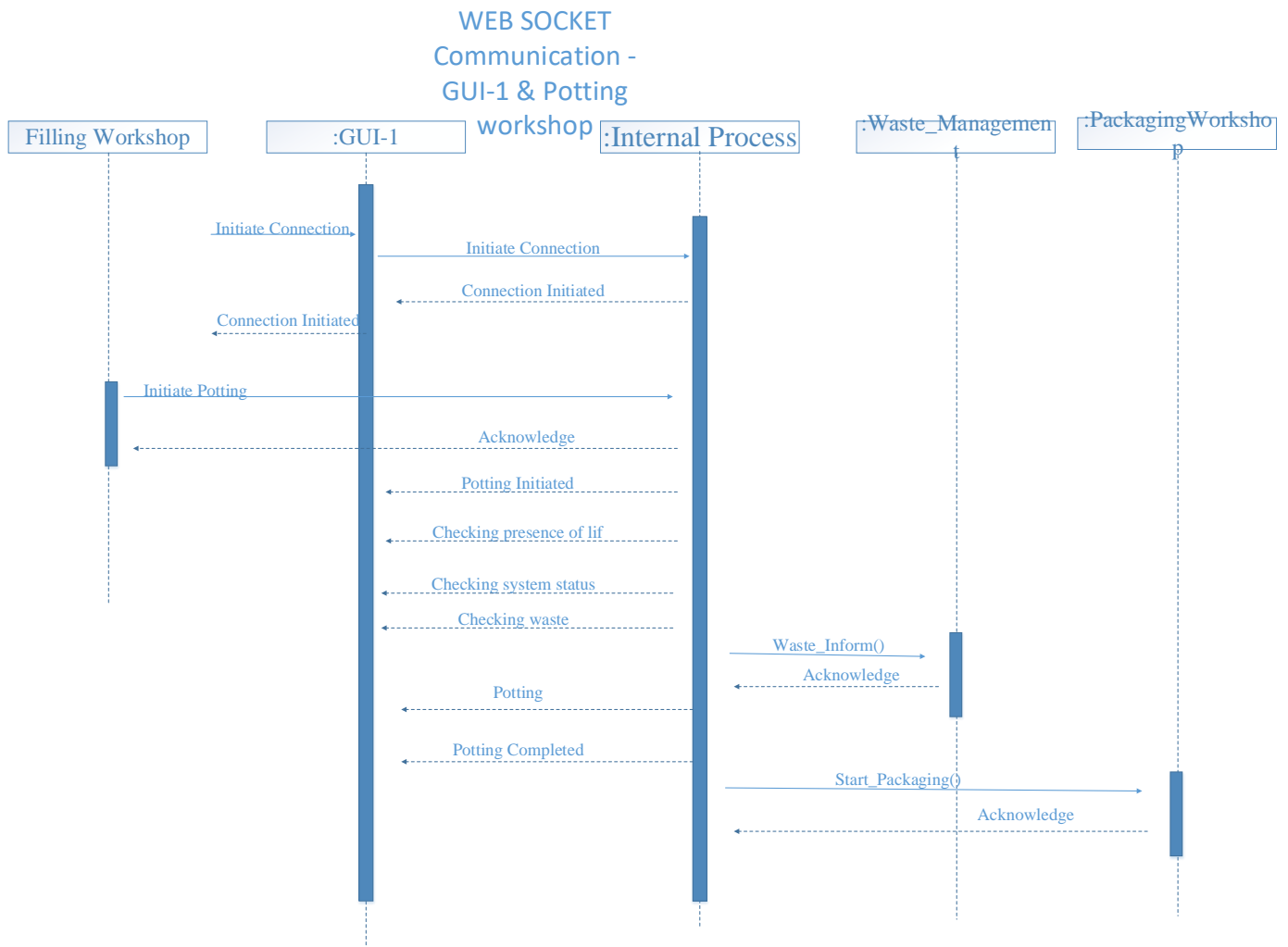


Figure 6: GUI-1 and potting workshop communication sequence

Above diagram shows communication between GUI 1 and Potting workshop. The communication initiated using web socket and it will live constantly through the process. The potting initiate function will be start soon after filling workshop sends a request. Waste management will be informed if any waste produced. At the end the packaging workshop will be informed.

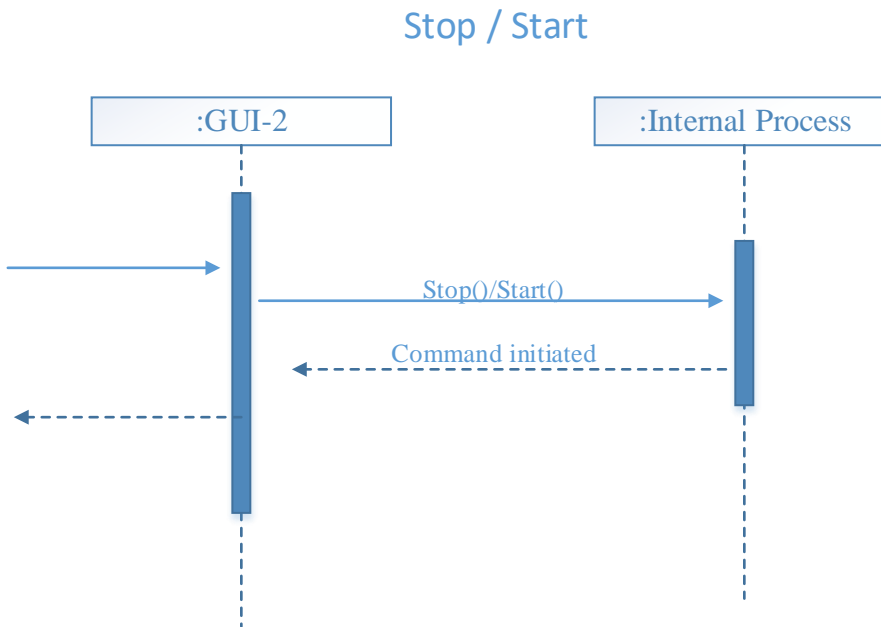
GUI – 2 and Potting workshop Communication

Figure 7: GUI 2 and potting workshop communication sequence

Above sequence diagram shows communication between GUI-2 and internal process of potting workshop. User can trigger stop and start commands and system will react to those commands accordingly.

Overview of Filling Workshop

Initially we receive data in regards to empty containers from Conveying Workshop. Filling workshop has to produce material for the cup which has been sent by conveying workshop.

Technical overview of Filling Workshop

Filling workshop is corresponding to receive the container from conveying workshop and to start filling the container with the product.

The main goal of this workshop are as follows. Checking the cup status, checking the machine status, checking availability of material, checking other connected machine status, checking power consumption, informing the waste management regarding waste products.

We expect to receive json data from the Conveying Workshop which we use in our internal process for Filling Workshop. The internal process of Filling Workshop receives the json data from the conveying workshop through a HTTP POST request and uses that data to initialise the filling process.

In the filling process, we get the empty containers, discriminate thee containers in terms of size such as small, medium and large, to finally fill these containers with small, medium and large amount of product. The size of the cup will be retrieved from json data which has been passed by conveying workshop. Following this, we check the availability of the material that would be used for filling. Each time, after filling, we update the availability amount of this material. We also check the connected machines such as the Potting mechanism and the packaging workshop availability. During the process of filling, we need to calculate the amount of waste generated in terms of broken pots during filling process. This is being done in order to generate json data to be sent to the waste management workshop, which contains information about broken pots. In this workshop, we also calculate the amount of energy expended during the process of filling and in this way, we make sure to finish the work within the stipulated energy that has been allocated to us by the Energy Management Workshop. In this regard, we would like to mention that the energy consumption capping information is being sent to us by the Energy Management Workshop.

Code sample of filling workshop

```

@RequestMapping(method = RequestMethod.POST, path = "/start_fill")
public ResponseEntity start_fill(@RequestBody Input_Prop obj)
{
    int result=fillingLogic.initiate(obj);
    ResponseCustom res=new ResponseCustom();
    if(result==1)
    {
        res.status="success";
        res.processID=10124;
    }
    else if(result==2)
    {
        res.status="System offline initiated";
        res.processID=12245;
    }
    else if(result==0)
    {
        res.status="failed";
        res.processID=12145;
    }

    return ResponseEntity.ok(res);
}

```

Following code match with post request from conveying workshop and will get the post body data from the request. The data contains information about container size and product type.

Example Json data that has been passed by conveying workshop.

```

{
  "public": false,
  "axot_under_power;": {true},
  "v90_alarm;": { true },
  "presence_pot;": { true },
  "homing_cam;": { true },
  "presence_product;": { true },
  "cup_size;": {1244},
  "downstream_b5_accumulation;": {},
  "product_name;": {"material"}
}

```

As soon as it received following request from the conveying workshop, filling internal process will be initiated and it will start to execute. Internal function will initially check for system status. If the system set for sleep mode, then it will return response code 2 which means the system is not active. This state will be initiated, only if the product has been finished or next connected machines are not active, or in case if user purposefully initiated this mode.

Sample code for sleep mode check.

//this code can be found in FillingLogic class.

```
public int initiate(Input_Prop obj)
{
    try {
        if(systemStatus.getStop())
        {
            stop();
            return 2;
        }
        //checking power
        if(powerCheck())
        {
            stop();
            return 2;
        }
    }
}
```

SystemStatus.getStop() will return true, if the system status is offline, then stop function will complete terminate the current internal process. Similarly powerCheck() will return true, if allocated power limit has been reached. In this case also the complete process will be terminated and sleep mode will be initiated automatically.

If any waste generated during the internal process execution, it will be automatically informed to the waste management service. Filling workshop will initiate a HTTP Post request to the waste management service with corresponding object that will be passed to the waste management service. That object will be used by waste management service to get information about the waste.

Sample code for post request to the waste management

//this code can be found in RestClient class.

```
public Object informWasteManage(WasteManagement obj)
{
    final String uri = "https://192.168.1.12/api/fillingWasteInform?workshop=filling";

    restTemplate.getForObject(uri,String.class);
    ResponseEntity<Object> responseEntity = restTemplate.postForEntity(uri,obj, Object.class);
    Object objects = responseEntity.getBody();
    System.out.println(objects);
    MediaType contentType = responseEntity.getHeaders().getContentType();
    return objects;
}
```

Sample Json

```
{
  "containers": "true",
  "falt_pillage": "false",
  "limit": "12.5",
  "exceeding_limit": "true"
}
```

The product data will be get updated each time of filling by the internal function. It will produce a request to the database server in order to update the product availability.

Sample code for updating product.

```
//this code can be found in fillingLogic class
public boolean updateAvailability(Cupconsume cup, Product prod)
{
    int avail=Integer.valueOf(prod.getAvailability());
    int updatedAvail=avail-Integer.valueOf(cup.getConsume());
    prod.setAvailability(String.valueOf(updatedAvail));
    Product result=productRepo.save(prod);

    if(result!=null)
        return true;
    else
        return false;
}
```

Power management data will be retrieved using get request by filling workshop. The allocated power consumption will be received from power management team and will be stored in the database of the application. This power consumption will be updated each time of filling process (reduced by 25 each time of filling).

Code sample for power management request

```
//this code can be found in restService class
public Object getPowerAllocation()
{
    final String uri = "https://192.168.1.12/api/opc/powerManagement?workshop=filling";
    restTemplate.getForObject(uri,String.class);
    ResponseEntity<Object> responseEntity = restTemplate.getForEntity(uri, Object.class);
    Object objects = responseEntity.getBody();
    MediaType contentType = responseEntity.getHeaders().getContentType();
    return objects;
}
```

Code sample for update power allocation in each time of filling

```
public double updatePower()
{
    Power power=powerRepo.findById(1).get();
    double value=power.getValue();
    power.setValue(value-25);
    Power result=powerRepo.save(power);
    double LeftPer=(power.getValue()/500)*100;
    double roundOff = Math.round(LeftPer * 100.0) / 100.0;
    return roundOff;
}
```


GUI 1 for Filling Workshop

GUI 1 has been developed using angular framework. The main objective of this interface is to fetch data from filling workshop and display it to the users. This part has been implemented using Socket programming. Initially a socket connect request will be made from GUI 1 to Filling Workshop.

Code sample for initial Request

//you can find thos code in filling-api-socket.ts

```
12 |      websocketEndPoint: string = 'ws://localhost:8080/initiate_connect';
13 |      topic: string = "/topic/greetings";
14 |      stompClient: any;
15 |      appComponent: AppComponent;
16 |      ws: any;
17 |      resp: response_;
18 |      connect() {
19 |          debugger
20 |          this.ws = new WebSocket(this.websocketEndPoint);
21 |          this.ws.onmessage = (data) => {
22 |              console.log(data);
23 |              debugger
24 |              this.onMessageReceived(data);
25 |          }
26 |      }
```

Figure 8: API Socket connect code - Filling GUI 1

Above code will initiate the connection to the web socket which has been hosted in following server location: `ws://localhost:8080/initiate_connect`. So, the Internal process function will keep this socket alive. Hence, it can keep update and sends all changing values and current functionalities to the interface. The interface application will capture those incoming data and render it to the user.

Following code sample shows how the incoming data is been arranged and rendered with a if else statement. All the incoming message will contains function id and page id attributes. These attributes will identified by angular interface application, and the data message will be rendered in relevant location based on these attributes.

Sample code for message handle function in interface – GUI 1*//you can find this code in main-page-component.ts*

```

handleMessage(message:response_){
    if(message.code==201 &&
message.page_id==1 )
    {
        if(message.status=="connected")
            this.is_connected=true;
    }
    if(message.page_id==2 &&message.func_id==101)
    {
        this.is_connected=true;
    }
    if(message.page_id==2 &&message.func_id==102)
    {
        this.process_status="Started";
        this.cup_size=message.arg;
    }
    if(message.page_id==2 &&message.func_id==103)
    {
        this.process_status="in-progress";
        this.cup_size_letr=message.arg;
        this.is_started=true;
    }
}

```

*Figure 9: Handle message function - filling interface***GUI – 2 filling workshop**

Gui-2 allows user to perform some function that will influence internal process of the workshop. At the moment, only user can able to initiate sleep mode of the application and can restart the system. In future it can be improved to perform some other interactive function that will influence internal process of the system. Below code sample explains how stop and restart commands from user will be send to the filling workshop internal function. (*//user-interaction.component.ts*)

```

Stop_()
{
    this.closeModal.emit();
    this.userIntService.stopCommand()
    .subscribe(res=>{
        debugger;
    },err=>{
        console.log(err);
    });
}
Start_()
{
    this.closeModal.emit();
    this.userIntService.startCommand()
    .subscribe(res=>{
        debugger;
    },err=>{
        console.log(err);
    });
}
}

```

Figure 10: User interaction form - filling workshop

GUI 1 – Screen

```
private static Message_Handler_Singleton obj=null;
```

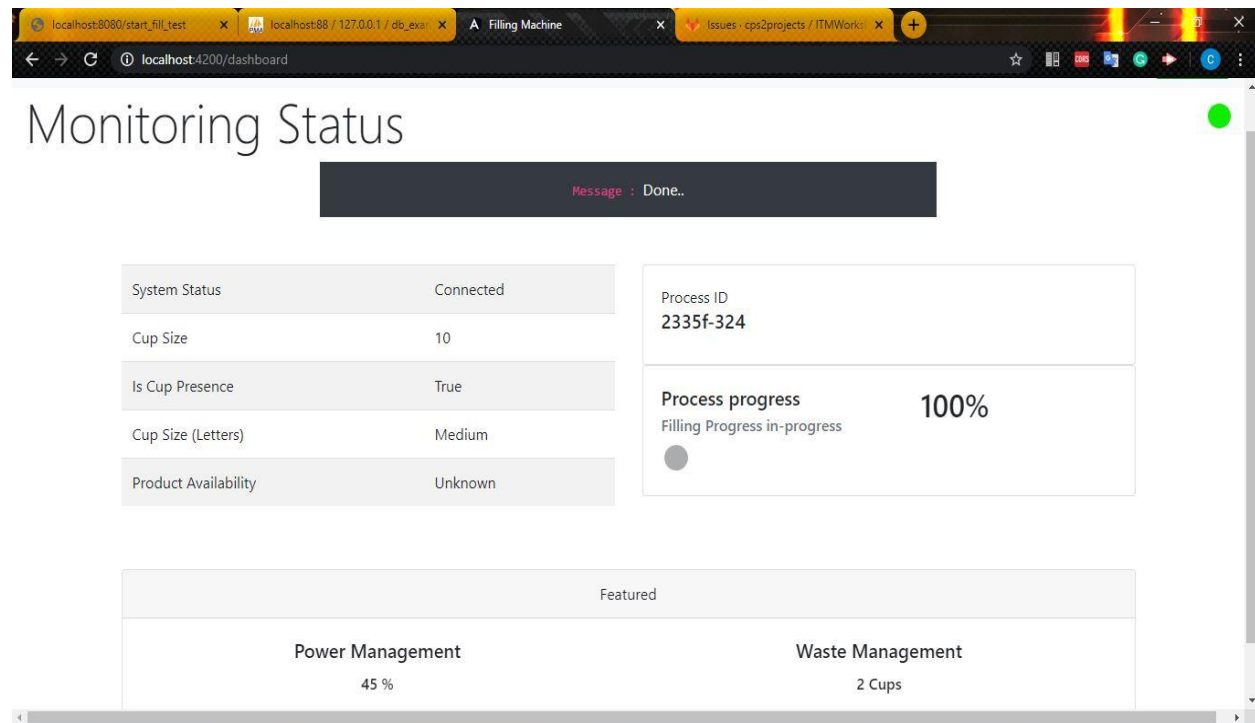


Figure 11: GUI-1-Filling workshop screen

Sample code for message handler singleton for session object in Java

```
//you can find this code in Message_Handler_Singleton.java class
private Object session;
private void Message_Handler_Singleton()
{
}
public static Message_Handler_Singleton getInstance()
{
    if (obj == null)
        obj = new Message_Handler_Singleton();
    return obj;
}
public boolean initiate(Object object)
{
    try {
        this.session=object;
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

Above code shows how client web socket will be stored in a singleton static property in order to make available constantly to send continuous message to the GUI-1 for status display. This java class has been implemented in internal process API application.

Overview of Potting Workshop

Initially we receive data in regards to filled containers from Filling Workshop. Potting workshop has to close the filled containers which has been sent by filling workshop and should further send the closed container to packaging workshop.

Technical overview of Potting Workshop

The main goal of this workshop are as follows. Checking the cup filled status, checking the machine status, checking availability of material, checking other connected machine status, and checking power consumption, informing the waste management regarding waste products.

We receive information from the filling workshop to initialise the potting process. The internal process of Potting Workshop closes the incoming pots in terms of small, medium, large with lids. During this process, any potting failure and hence waste material generation information is being sent to the waste management workshop. In this workshop, we also calculate the amount of energy expended during the process of potting and in this way, we make sure to finish the work within the stipulated energy that has been allocated to potting workshop by the Energy Management Workshop.

Initially a post request from filling workshop will be received to the potting workshop using HTTP Post method. Following Json data will be received.

Sample Json data

```
{
  "kml": "true;",
  "axot_under_power": "true;",
  "v90_alarm": "true;",
  "presence_pot": "true;",
  "homing_cam": "true;",
  "presence_product": "true;",
  "cup_size": "\"10\"",
  "downstream_b5_accumulation": "true;",
  "product_name": "\"salt\"",
  "weight": "200;",
  "speed": "5;"
}
```

Sample Start Potting request mapping function

```

@RequestMapping(method=RequestMethod.POST, path= "/start_potting")
public ResponseEntity start_potting(@RequestBody Input_Prop obj)
{
    int result=pottingLogic.initiate(obj);
    //      logic.initiate(obj);
    ResponseCustom res=new ResponseCustom();
    if(result==1)
    {
        res.status="success";
        res.processID=10124;
    }
    else if(result==2)
    {
        res.status="System offline initiated";
        res.processID=12245;
    }
    return ResponseEntity.ok(res);
}

```

Filling workshop will sent a post request to with above mentioned Json data to the potting workshop. That request will get mapped to this function and potting logic initiate function will start potting internal process.

From the above mentioned Json data, the cup size, turning speed, presence of pot, and the weight will be identified. These data will be used in internal process of Potting workshop. Initially the availability of lids will be checked in the system based on the container size. Also, the system will check next connected machine live status. Then if everything success the internal process of closing lid will be started.

```

//you can find this code in pottinLogic class
msg.message="success_ checking presence of Cup";
msg.page_id=2;
msg.func_id=106;
msg.message_type="message";
instance.sendMsh(msg);
if(userinter.getStop())
{
    stop();
    return 2;
}

Thread.sleep(3000);
msg.message="Checking Pot Size";
msg.page_id=2;
msg.func_id=0;
msg.message_type="message";
instance.sendMsh(msg);
int potSize= check_pot_size(obj);
System.out.println(potSize);
Thread.sleep(3000);
}

```

Above code will check for system status initially and also it will inform to the user of its status. That message will be handled by GUI-1 of potting workshop where user can see each status and current function of the system. You can see a variable named “instance” which is instance of client socket object. This object is a singleton object which will be available in all other classes that will be used to send display messages to the GUI-1.

The lid availability will be checked initially. The function code were given below.

//you can find this code in PottingLogic class

```
public Pot checkAvailability(Input_Prop obj)
{
    Pot pot=null;
    Iterable<Cupconsume> cupList=cupRepo.findAll();
    Iterable<Pot> potlist=potRepo.findAll();
    Cupconsume cup=null;
    for(Cupconsume item : cupList)
    {
        if(item.getSize().equals(obj.getCup_size()))
        {
            cup=item;
        }
    }
    for(Pot item : potlist)
    {
        if(item.getName().equals(cup.getName()))
        {
            pot=item;
        }
    }
    if(pot!=null)
        return pot;
    else
        return null;
}
```

Initially container will be retrieved to get exact information about the container. Then that container data will be used to get exact lid that matching the container size and type. Retrieved information will helps to identify the availability of the lids.

Similar like filling application, potting application also will be check for power consumption and inform waste management service regard of any waste.

```
//you can find this code in potting logic class
public double updatePower()
{
    Power power=powerRepo.findById(1).get();
    double value=power.getValue();
    power.setValue(value-25);
    Power result=powerRepo.save(power);
    double leftPer=(power.getValue()/500)*100;
    double roundOff = Math.round(leftPer * 100.0) / 100.0;
    return roundOff;
}
```

Above code shows how power consumption will be updated in each time of internal process. 25 Unit of power will be reduced in each time of potting process. Then it will send the power information to the user.

Waste information also will be informed to the waste management service whenever waste produced by potting workshop. Below code shows how waste management service will be get informed about waste from potting workshop.

```
//you can find this code in RestService class
public Object pottingWasteInform(WasteManagement obj)
{
    final String uri = "https://192.168.1.12/api/pottingWasteInform?status=";

    restTemplate.getForObject(uri,String.class);
    ResponseEntity<Object> responseEntity = restTemplate.postForEntity(uri,obj, Object.class);
    Object objects = responseEntity.getBody();
    System.out.println(objects);
    MediaType contentType = responseEntity.getHeaders().getContentType();
    return objects;
}
```

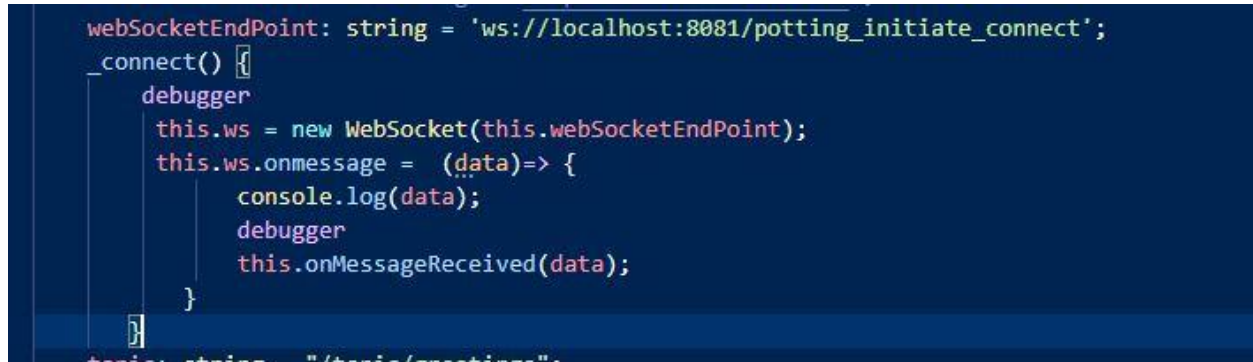
Above code shows how waste management post request will be sent from potting workshop. Waste Management model will be contain properties that has been expected request body in waste management service.

GUI – Potting workshop

GUI 1 has been developed using angular framework. The main objective of this interface is to fetch data from potting workshop and display it to the users. This part has been implemented using Socket programming (Similar to filling workshop). Initially a socket connect request will be made from GUI 1 to potting workshop.

Code sample for initial Request

//you can find thos code in filling-api-socket.ts

A screenshot of a code editor showing TypeScript code for a WebSocket connection. The code is as follows:

```
webSocketEndPoint: string = 'ws://localhost:8081/potting_initiate_connect';  
connect() {  
  debugger  
  this.ws = new WebSocket(this.webSocketEndPoint);  
  this.ws.onmessage = (data) => {  
    console.log(data);  
    debugger  
    this.onMessageReceived(data);  
  }  
}
```

Figure 12: API Socket connect code - Potting GUI 1

Above code will initiate the connection to the web socket which has been hosted in following server location: *ws://localhost:8080/potting_initiate_connect* . So, the internal process function will keep this socket alive. Hence, it can keep update and sends all changing values and current functionalities to the interface. The interface application will capture those incoming data and render it to the user.

The message that has been passed by potting internal process will be handled similar like filling workshop GUI-1 application. The incoming data is been arranged and rendered with an “if else” statement. All the incoming message will contains function id and page id attributes. These attributes will identified by angular interface application, and the data message will be rendered in relevant location based on these attributes.

Similar to filling workshop, potting workshop also maintains a singleton web socket client object in Java application. The client web socket will be stored in a singleton static property in order to make available constantly to send continuous message to the GUI-1 for status display. This java class has been implemented in internal process API application.

Sample code for message handler singleton for session object in Java

```
//you can find this code in Message_Handler_Singleton.java class
private Object session;
private void Message_Handler_Singleton()
{
}
public static Message_Handler_Singleton getInstance()
{
    if (obj == null)
        obj = new Message_Handler_Singleton();
    return obj;
}
public boolean initiate(Object object)
{
    try {
        this.session=object;
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

GUI – 2 potting workshop

Gui-2 allows user to perform some function that will influence internal process of the workshop. At the moment, only user can able to initiate sleep mode of the application and can restart the system. In future it can be improved to perform some other interactive function that will influence internal process of the system. you can find the code for this user interaction function in ‘user.interaction.component.ts’ file in ‘src’ folder.GUI 1 and 2 –Potting Workshop

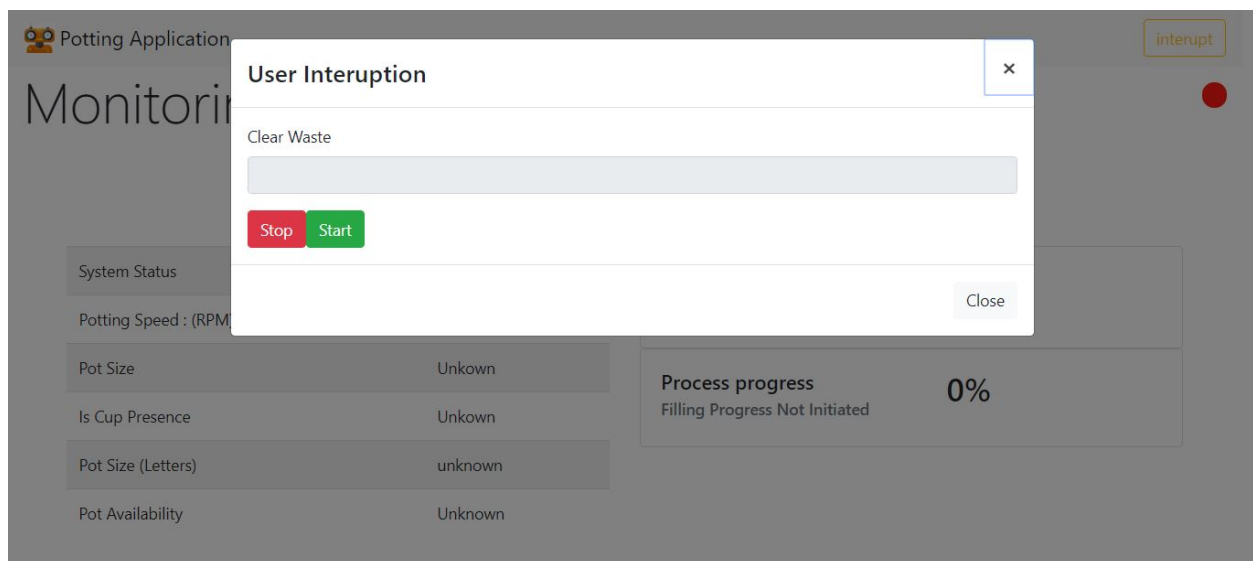


Figure 13: Potting workshop GUI

Conclusion

The potting and filling workshop were designed b considering interoperability features with other team. The application were developed using API and web socket technologies. User interfaces also completely decoupled with internal process, hence it is easy to upgrade and maintain the application. The application can be further upgraded with different features and modules in future.