Contents:

1. Objective
2. System Description
3. Methodology and tools
4. Iterations
    First Iteration
    Second Iteration
    Third Iteration
    Fourth Iteration
    Final Iteration
5. Conclusion – final Iteration
6. Problems faced and learning outcomes
7. References.

1. Objective

To create a client user interface, which would help us to monitor and control a workstation of the FASTory simulator by using web services.

2. System Description

The FASTory line is capable of assembling mobile phones. But in the TUT lab it is used as a research environment and it only draws different parts of the mobile phones like screen, keyboard and frame with different color and shape options. There are pallets which move through the line on conveyors and these contain white papers on which the drawings are made. These papers with drawings are considered as final products. The total line consists of twelve workstations, of which two of them are for loading and unloading purpose. The transportation is done with help of conveyors, two input conveyors and one output conveyors. Pallets and papers are also loaded and unloaded by conveyors. There are pen attached with the robot's end effector. The robots that are used are Sony's SCARA robots. Each workstation consists of a robot, a conveyor line with a bypass line. There are five zones on each line indicated as Z1 to Z5. Each zone has a sensor for sensing the pallet and a stopper to stop the pallet on that position. On the entrance of each zone there is a RFID reader for pallet identification.

For this entire system we are building the user interface where we would be able to monitor and control workstation 8 of the simulator.

3. Methodology & Tools

For modelling the project we approached the Unified Process. **ProjectLibre** software was used at the primary phase to divide the work in a time frame and cost analysis was done with some assumptions. We used **Visual Paradigm** for modelling purposes. We modeled our system using use case diagram, class diagram, activity diagram, state chart etc. Various platforms were used for making the User Interface. We designed our visuals using **Scalable Vector Graphics (SVG)** and presented in **HTML** format as a web page. For coding, creating the server and animation we used Node.js which has become a famous platform for its non-blocking I/O method and also for its huge library. We also used some **JavaScript** and **JSON**. For web services **REST** method was used and **Postman** REST client was used. All the coding was written using **Komodo edit 9**. The pre made SVG was modified using Google SVG creator.

4. Iterations

The work of the project was done in five successive iterations. At the primary level we were just introduced to the Unified process and learned the use of ProjectLibre Software to understand the four phases of Unified process. In our first iteration we made our first use case diagram using Visual Paradigm. In our use case we tried to keep various scope for our project. Figure 1 shows our first use case diagram. Also we described briefly our system vision and what we want to achieve or can provide our clients.

First Iteration:

When we started, we wanted to implement a monitoring system that would inform the user about the complete health of the work station. This would include the start and stop of the process, name and position of pallet, load/unload materials, alarm responses and sensor status which would be useful to an operator. For the projection manager, details about order placed, the production record, monitoring the production, System efficiency, Raw material requirement and maintenance requirements are made available. Also a maintenance engineer can have access to the raw material requirement and the scope of maintenance as shown in the first iteration of the Use Case Diagram.

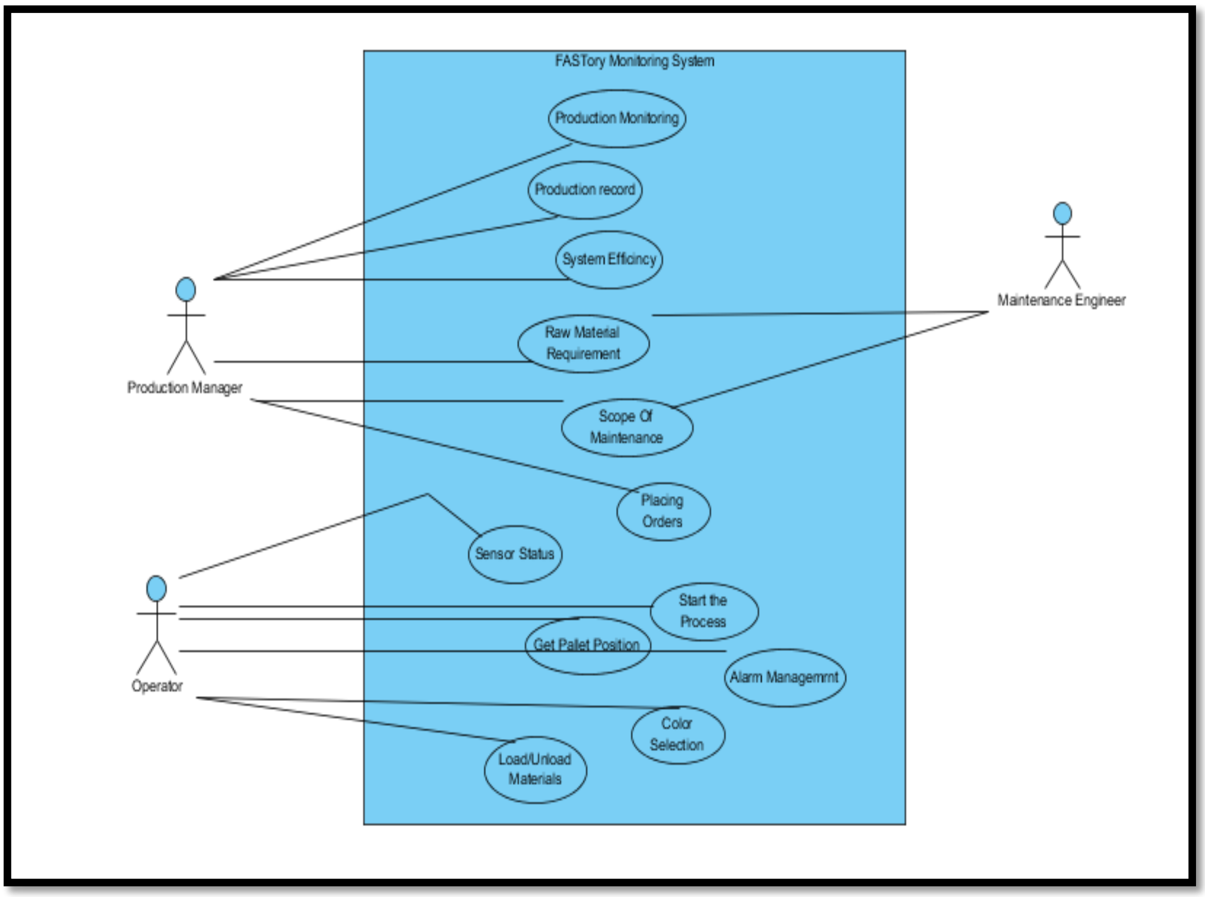


Fig 1: Use Case Diagram

Second iteration:

In this iteration, we made the first cut of the class diagram showing the various data types that would be considered in each class and their interrelations. However, we reduced the extent of the project and considered mainly the machines/ workstations (robot and conveyer both inherited from it) and the reports and orders placed in relation to the work station. The various attributes required to define each of the classes were listed in it.

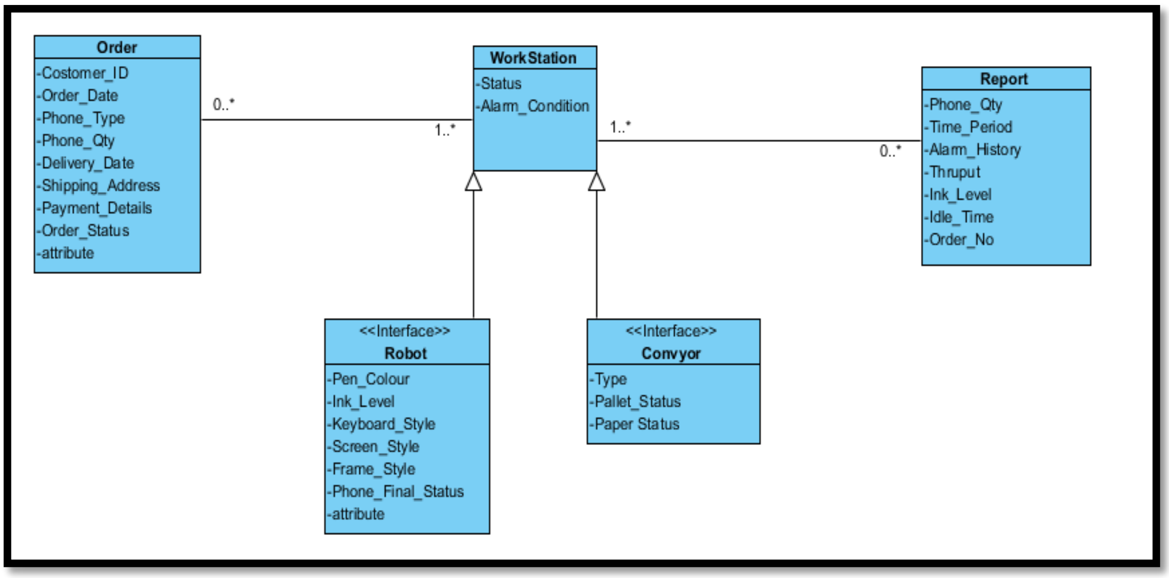
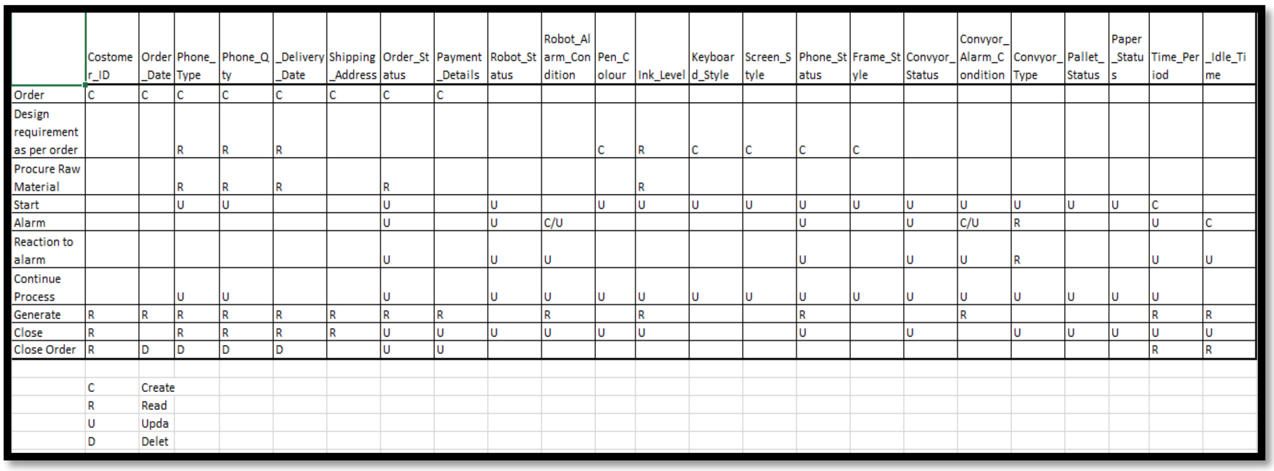


Fig 2 Class Diagram

Also, CRUD analysis were performed to understand which variables get created and updated by whom and in which stage of the program flow.



	Costomer_ID	Order_Date	Phone_Type	Phone_Qty	_Delivery_Date	Shipping_Address	Order_Status	Payment_Details	Robot_Status	Robot_Alarm_Condition	Pen_Colour	Ink_Level	Keyboard_Style	Screen_Style	Phone_Status	Frame_Style	Convyor_Status	Convyor_Alarm_Condition	Convyor_Type	Pallet_Status	Paper_Status	Time_Period	_Idle_Time
Order	C	C	C	C	C	C	C	C															
Design requirement as per order			R	R	R						C	R	C	C	C	C							
Procure Raw Material			R	R	R		R					R											
Start			U	U			U		U		U	U	U	U	U	U	U	U	U	U	U	C	
Alarm							U		U	C/U					U		U	C/U	R			U	C
Reaction to alarm							U		U	U					U		U	U	R			U	U
Continue Process			U	U			U		U	U	U	U	U	U	U	U	U	U	U	U	U	U	
Generate	R	R	R	R	R	R	R	R		R		R			R			R				R	R
Close	R		R	R	R	R	U	U	U	U	U	U			U		U		U	U	U	U	U
Close Order	R	D	D	D	D		U	U														R	R

C	Create
R	Read
U	Upda
D	Delet

Fig 3 Class Diagram

Third Iteration:

In this iteration, we made a sequence diagram to show the sequence of events that are carried out when different actions such as login and are carried out. To ensure different users at different levels can log in to the system, a log in method is required. This will give a particular user, access to only some level of data and commands depending on whether the personnel is an operator or manager. The sequence diagram for the same would be as follows:

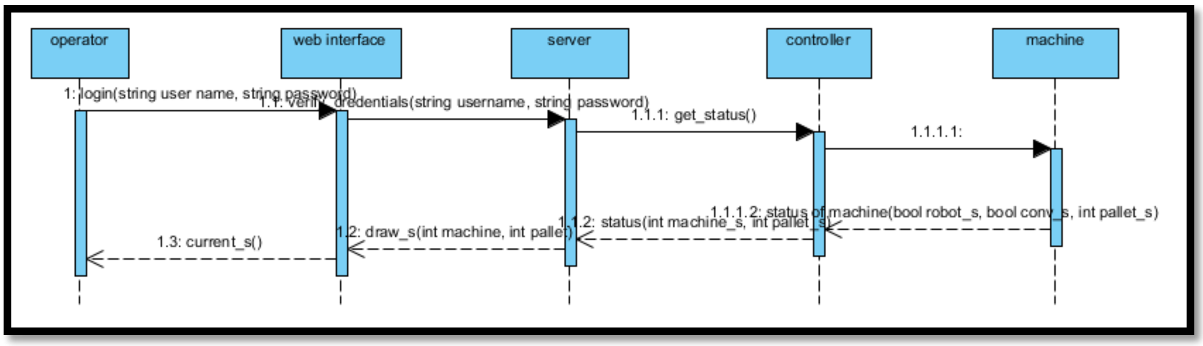

operator
web interface
server
controller
machine
1: login(string user name, string password)
1.1: verify_credentials(string username, string password)
1.1.1: get_status()
1.1.1.1:
1.1.1.2: status of machine(bool robot_s, bool conv_s, int pallet_s)
1.1.2: status(int machine_s, int pallet_s)
1.2: draw_s(int machine, int pallet)
1.3: current_s()

Fig 4: sequence chart for carrying actions like login

Similarly, to start production, the sequence of actions to ensure the operator can start the machine and the same is reflected on the UI can be seen in the diagram below.

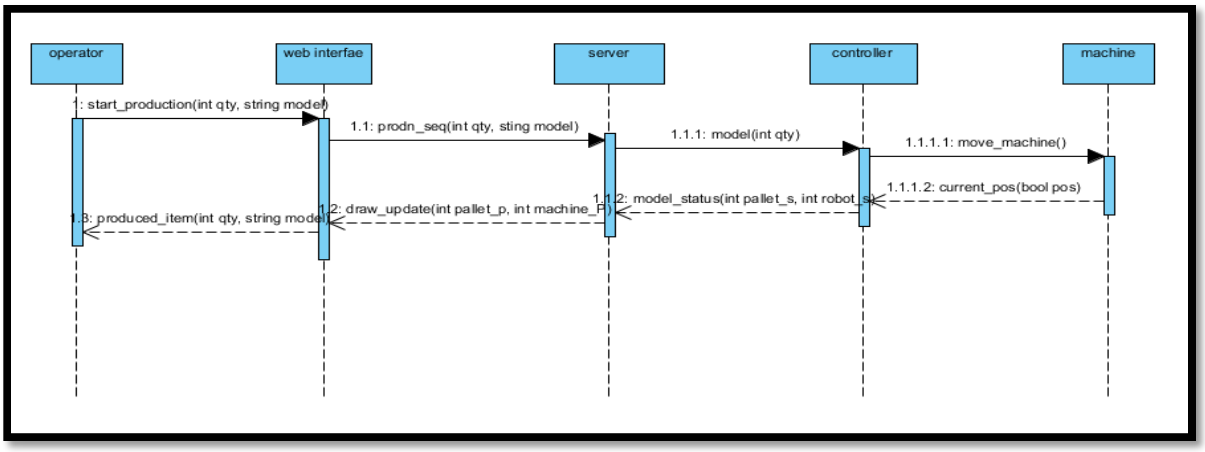

operator
web interfae
server
controller
machine
1: start_production(int qty, string model)
1.1: prodn_seq(int qty, sting model)
1.1.1: model(int qty)
1.1.1.1: move_machine()
1.1.1.2: current_pos(bool pos)
1.1.2: model_status(int pallet_s, int robot_s)
1.2: draw_update(int pallet_p, int machine_P)
1.3: produced_item(int qty, string model)

Fig 5: sequence chart for carrying actions like start machine

Also a state diagram was developed to show the different stages the machine would go through as the sequence of event gets executed. For example, the state diagram of the robot is shown here, where in the robot goes from idle to busy where in it can be drawing the frame, keyboard or screen. Once the three are done, it checks if the required number of phones are drawn and if not the cycle is repeated. Once the number is reached, the robot turns off/ goes back to idle.

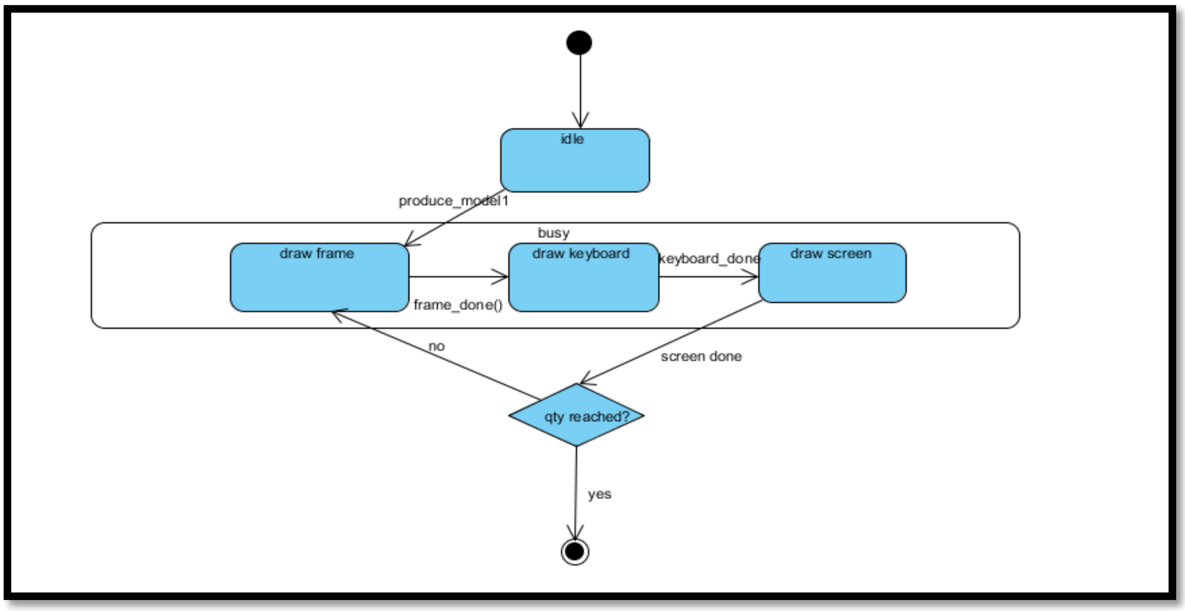


Fig 6: State diagram

Fourth Iteration:

In the fourth iteration, we developed an activity diagram. This diagram gave us detail about how information would pass between the various entities. It shows what happens when a user logs in to the system, followed by the user choosing the model of the phone to be produced and the quantity and finally requesting for a production report at the end of the production.

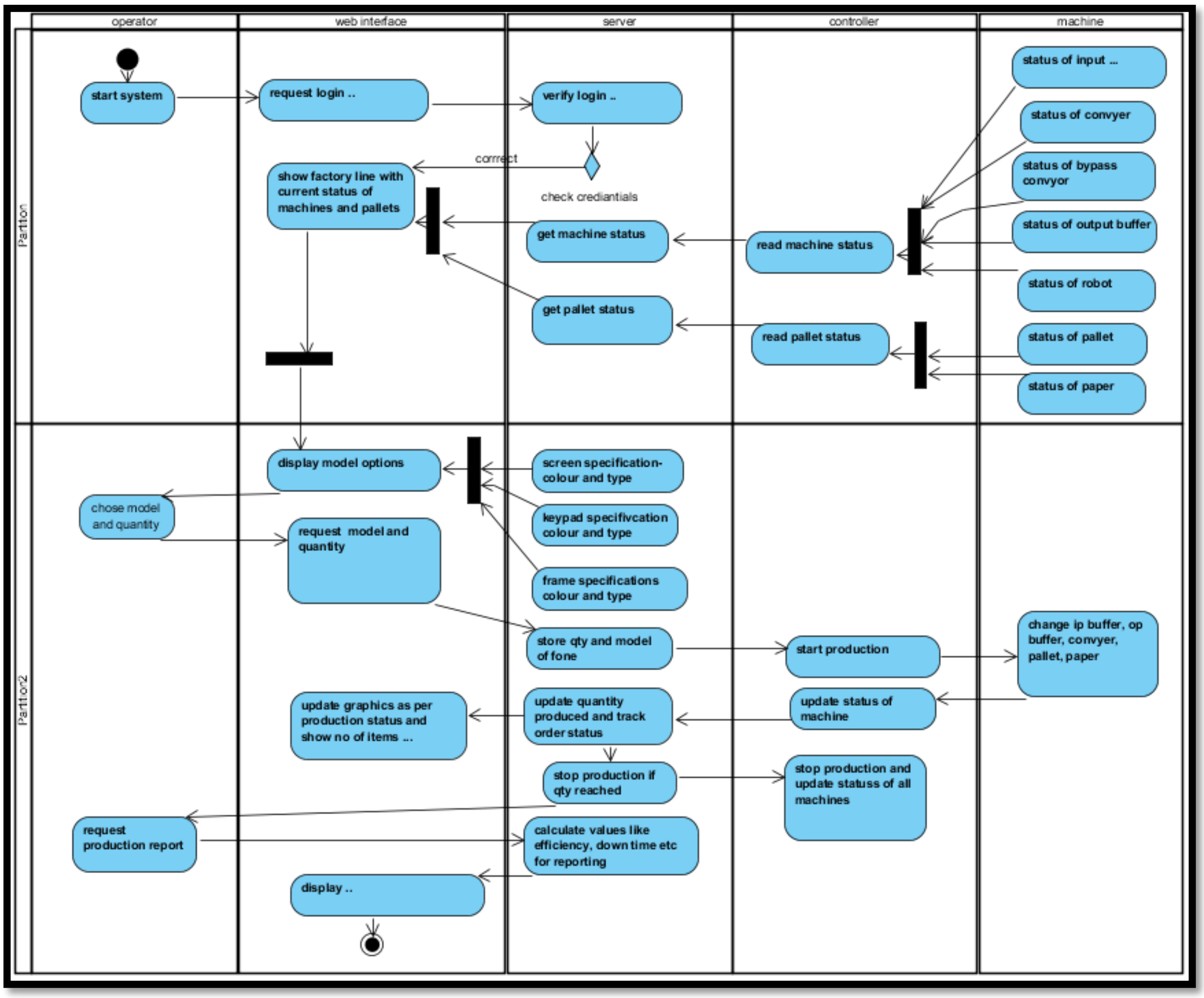

operator
web interface
server
controller
machine
Partion
Partion2
start system
request login ..
verify login ..
correct
check crediantials
show factory line with current status of machines and pallets
get machine status
read machine status
get pallet status
read pallet status
status of input ...
status of convyer
status of bypass convyor
status of output buffer
status of robot
status of pallet
status of paper
display model options
chose model and quantity
request model and quantity
screen specification-colour and type
keypad specifivcation colour and type
frame specifications colour and type
store qty and model of fone
start production
change ip buffer, op buffer, convyer, pallet, paper
update status of machine
update quantity produced and track order status
update graphics as per production status and show no of items ...
stop production if qty reached
stop production and update statuss of all machines
request production report
calculate values like efficiency, down time etc for reporting
display ..

Fig7: Activity Diagram

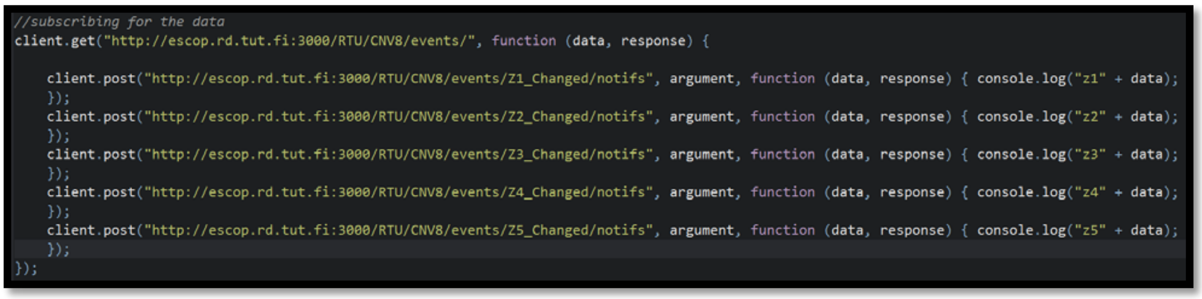
Final Iteration:

In this iteration, a control and monitoring system was established, on which the pallet information like its position and id were controlled and monitored. This was achieved by first, getting the status of each zone from the simulator and populating it on the UI when a user logs on to our UI. Further on if any change happens to the position of a pallet in the simulator, notifications are received by our server, which emits them to our html, which in turn reacts to these notifications by changing the position of the pallet and also showing the change in status of each zone, thus establishing monitoring of the pallet position and pallet id.

To achieve control, buttons are present on our UI which can be used to load or unload a pallet or move it between the 5 zones of a work cell. When the user hits any of these buttons, an event gets emitted to our server, which further posts a request to the FASTory Simulator to execute the movement. On execution of this movement, similar to the monitoring case, notifications are generated as the pallet moves, and these notifications are used to trigger change in animations on our UI.

The information exchange between index.html and the server is achieved using socket.io. socket.emit command is used to emit data whereas io.on is used to receive the message. The information exchange between the server and the simulator is achieved using the rest client, where get and post are used to receive and send data.

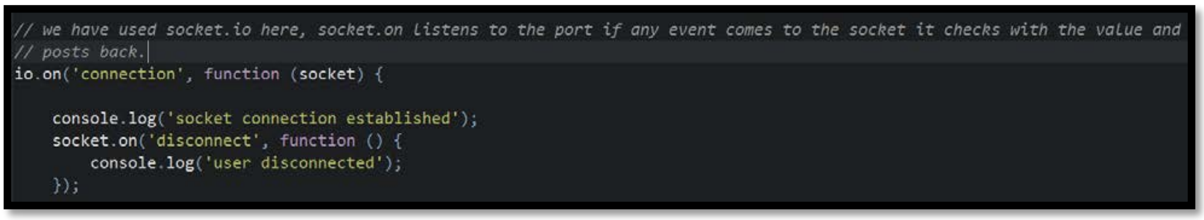
Few examples of the code



```
//subscribing for the data
client.get("http://escop.rd.tut.fi:3000/RTU/CNV8/events/", function (data, response) {

    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z1_Changed/notifs", argument, function (data, response) { console.log("z1" + data);
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z2_Changed/notifs", argument, function (data, response) { console.log("z2" + data);
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z3_Changed/notifs", argument, function (data, response) { console.log("z3" + data);
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z4_Changed/notifs", argument, function (data, response) { console.log("z4" + data);
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z5_Changed/notifs", argument, function (data, response) { console.log("z5" + data);
    });
});
```

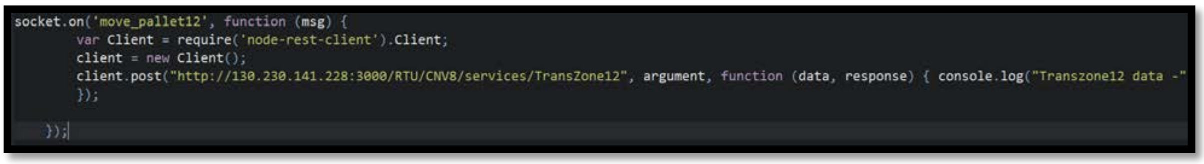
Fig 8: This is the part of the code where we are subscribing for the notifications for all the 5 zones.



```
// we have used socket.io here, socket.on listens to the port if any event comes to the socket it checks with the value and
// posts back.
io.on('connection', function (socket) {

    console.log('socket connection established');
    socket.on('disconnect', function () {
        console.log('user disconnected');
    });
```

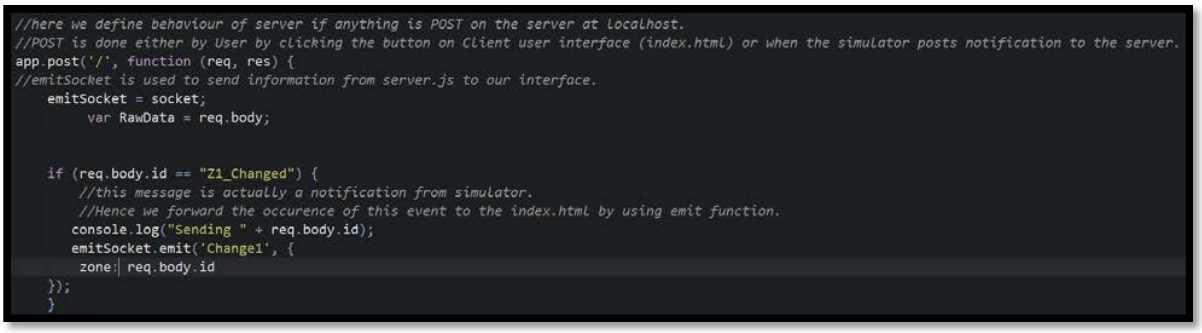
Fig 9: This is the part where we are creating a socket.io connection for server.js and the index.html. Where then HTML files is connected then it displaced “socket connection established” on the console. When the user disconnects it displays “user disconnected” on the console.



```
socket.on('move_pallet12', function (msg) {
        var Client = require('node-rest-client').Client;
        client = new Client();
        client.post("http://130.230.141.228:3000/RTU/CNV8/services/TransZone12", argument, function (data, response) { console.log("Transzone12 data -"
        });

    });
```

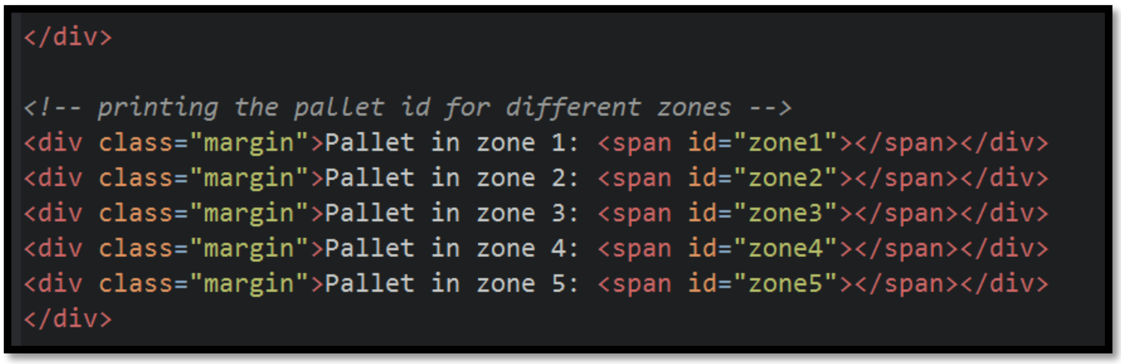
Fig 10: This is one of example where we have used socket.on, the function of this part of the code is to listen when the button on the index.html is triggered and send the response.



```
//here we define behaviour of server if anything is POST on the server at localhost.
//POST is done either by User by clicking the button on Client user interface (index.html) or when the simulator posts notification to the server.
app.post('/', function (req, res) {
//emitSocket is used to send information from server.js to our interface.
    emitSocket = socket;
         var RawData = req.body;


    if (req.body.id == "Z1_Changed") {
        //this message is actually a notification from simulator.
        //Hence we forward the occurence of this event to the index.html by using emit function.
       console.log("Sending " + req.body.id);
       emitSocket.emit('Change1', {
        zone: req.body.id
    });
    }
```

Fig 11: here we are designing the code, where it decides the behavior of the server if anything is post on server at local host. The emitSocket.emit is used to emit the information from server.js to our interface. The use of the emit function is to forward the occurrence of the event to index.html.

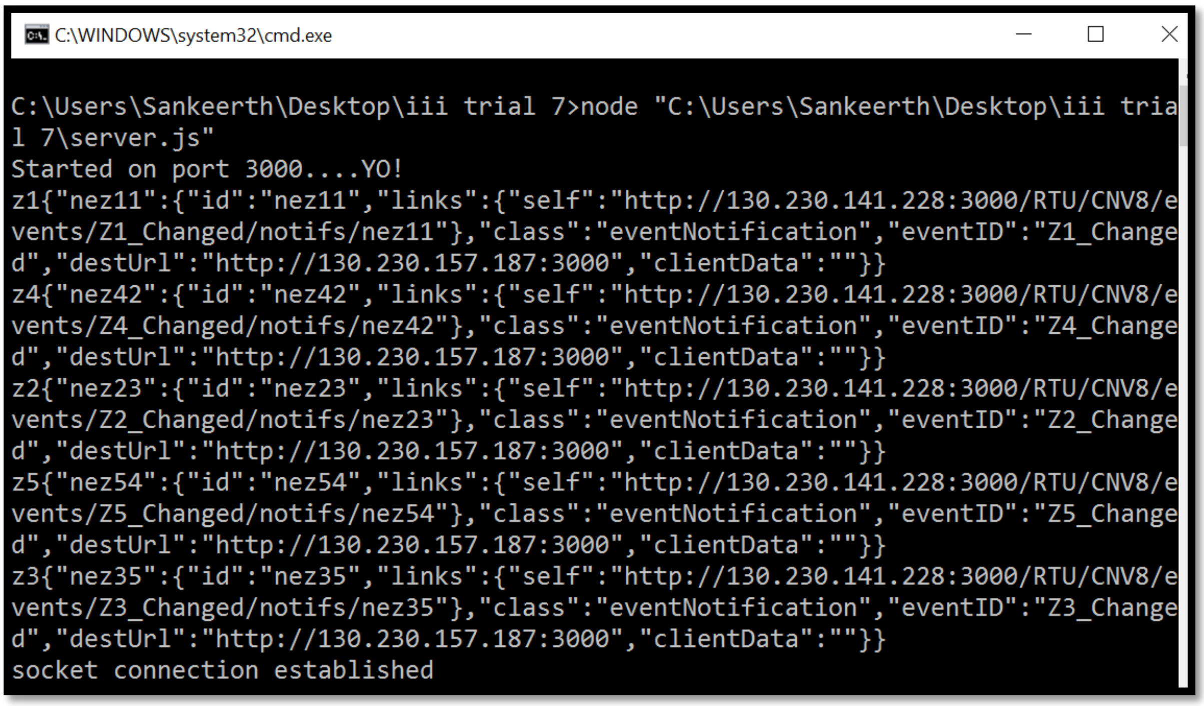


```
</div>

<!-- printing the pallet id for different zones -->
<div class="margin">Pallet in zone 1: <span id="zone1"></span></div>
<div class="margin">Pallet in zone 2: <span id="zone2"></span></div>
<div class="margin">Pallet in zone 3: <span id="zone3"></span></div>
<div class="margin">Pallet in zone 4: <span id="zone4"></span></div>
<div class="margin">Pallet in zone 5: <span id="zone5"></span></div>
</div>
```

Fig 12: here we are printing the occurrence of the pallet in different zones on HTML.

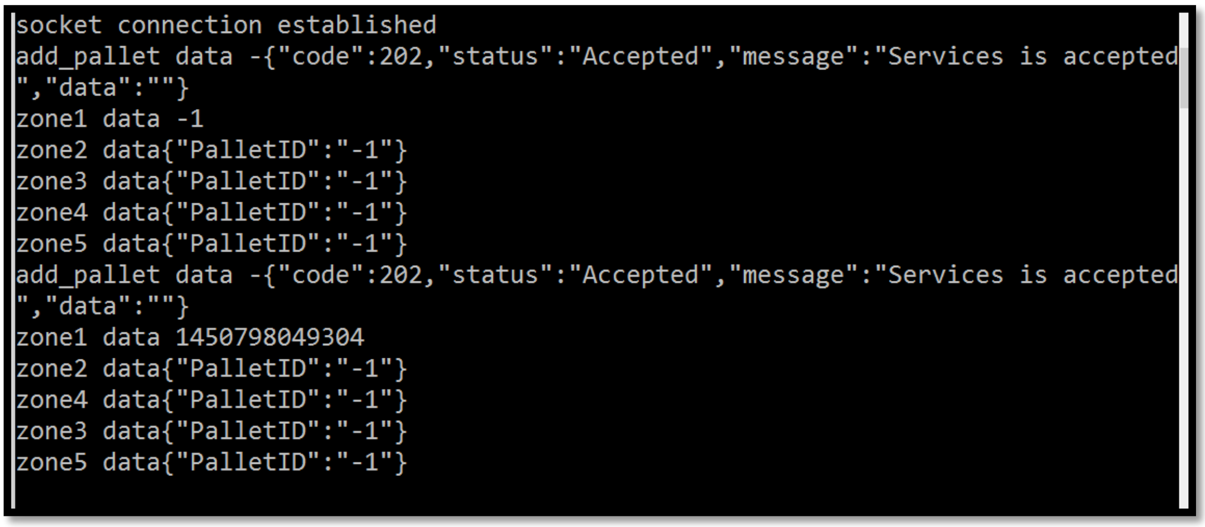
Screenshots of the monitoring and control system.



C:\WINDOWS\system32\cmd.exe

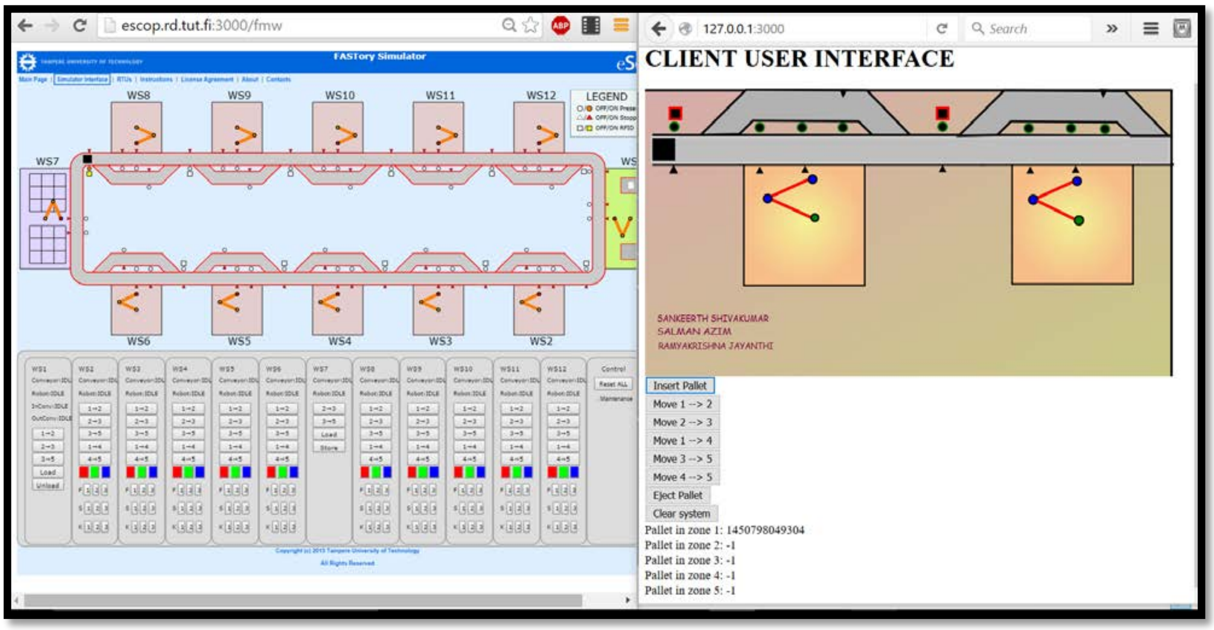
```
C:\Users\Sankeerth\Desktop\iii trial 7>node "C:\Users\Sankeerth\Desktop\iii tria
l 7\server.js"
Started on port 3000....YO!
z1{"nez11":{"id":"nez11","links":{"self":"http://130.230.141.228:3000/RTU/CNV8/e
vents/Z1_Changed/notifs/nez11"},"class":"eventNotification","eventID":"Z1_Change
d","destUrl":"http://130.230.157.187:3000","clientData":""}}
z4{"nez42":{"id":"nez42","links":{"self":"http://130.230.141.228:3000/RTU/CNV8/e
vents/Z4_Changed/notifs/nez42"},"class":"eventNotification","eventID":"Z4_Change
d","destUrl":"http://130.230.157.187:3000","clientData":""}}
z2{"nez23":{"id":"nez23","links":{"self":"http://130.230.141.228:3000/RTU/CNV8/e
vents/Z2_Changed/notifs/nez23"},"class":"eventNotification","eventID":"Z2_Change
d","destUrl":"http://130.230.157.187:3000","clientData":""}}
z5{"nez54":{"id":"nez54","links":{"self":"http://130.230.141.228:3000/RTU/CNV8/e
vents/Z5_Changed/notifs/nez54"},"class":"eventNotification","eventID":"Z5_Change
d","destUrl":"http://130.230.157.187:3000","clientData":""}}
z3{"nez35":{"id":"nez35","links":{"self":"http://130.230.141.228:3000/RTU/CNV8/e
vents/Z3_Changed/notifs/nez35"},"class":"eventNotification","eventID":"Z3_Change
d","destUrl":"http://130.230.157.187:3000","clientData":""}}
socket connection established
```

Fig 13: Where we run the server.js file and connect our UI to the web browser, console shows that the socket connect has been established.



```
socket connection established
add_pallet data -{"code":202,"status":"Accepted","message":"Services is accepted
","data":""}
zone1 data -1
zone2 data{"PalletID":"-1"}
zone3 data{"PalletID":"-1"}
zone4 data{"PalletID":"-1"}
zone5 data{"PalletID":"-1"}
add_pallet data -{"code":202,"status":"Accepted","message":"Services is accepted
","data":""}
zone1 data 1450798049304
zone2 data{"PalletID":"-1"}
zone4 data{"PalletID":"-1"}
zone3 data{"PalletID":"-1"}
zone5 data{"PalletID":"-1"}
```

Fig 14: Where we click the button Add pallet or when you load the pallet from the simulator, console shows that the pallet is in Zone 1, and displays its Pallet ID.



```
Pallet in zone 1: 1450798049304
Pallet in zone 2: -1
Pallet in zone 3: -1
Pallet in zone 4: -1
Pallet in zone 5: -1
```

Fig 15: This is how it looks on our UI and the FASTory Simulator.

<u>Note:</u>

To See more of the operations like this you can click on the below link:

https://youtu.be/QYDbzimNeSA

5. Conclusion – Final Iteration

In the final iteration we reduced the scope to only control and monitor the position of the pallet and provide the status of a zone ie show the pallet id if a pallet is present in a zone, else show -1.

This change is a result of actually understanding the complexities in actually implementing the system, which is also evident from the final iteration of the UML diagrams.

The Use Case diagram: in the first iteration showed how different users would interact with the system as a whole. In the final iteration, the diagram is modified where in the system comprises of three parts, the UI/html, the server and the simulator, each of which interacts with the other. The actors are the user who interacts with the UI and the system which affects the simulator.

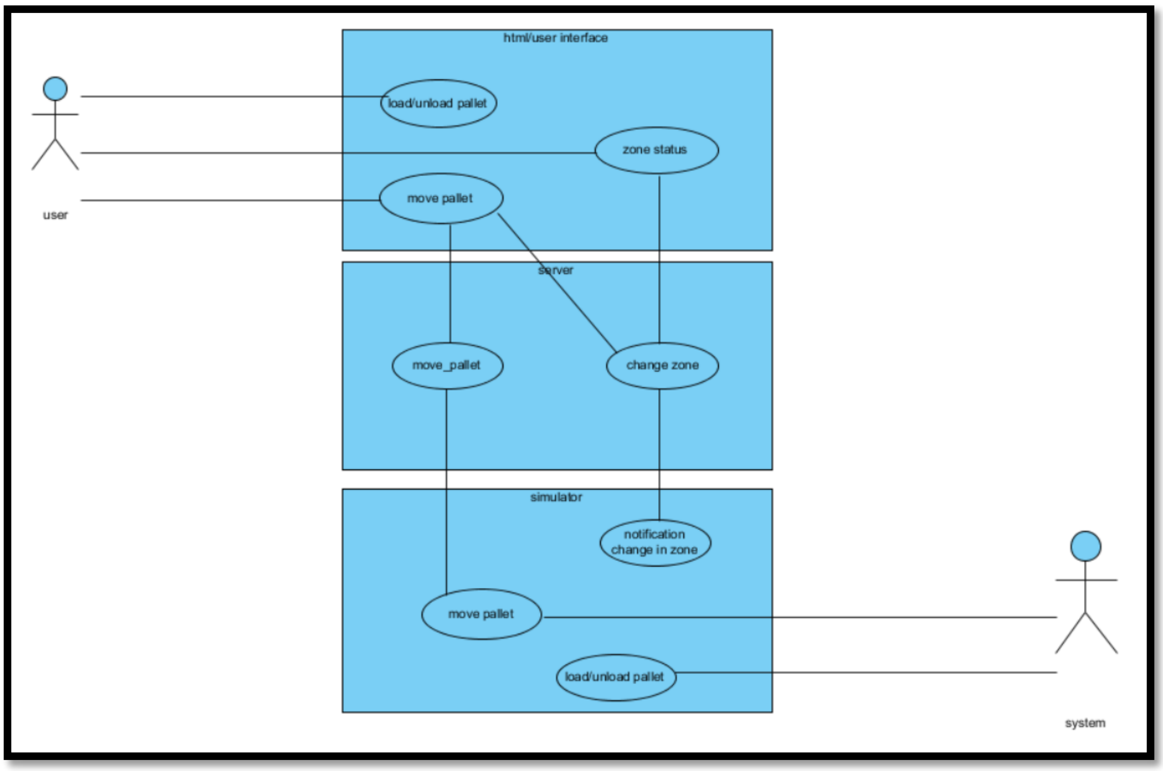


Fig 16: The use case Diagram

Class Diagram: Initially, the classes were understood to be all the physical entities that would be interacting with the system or user and it would describe the attributes or properties of these entities and the functions or capabilities they had. However after using javascript, we realized that classes in javascript work differently. Each of the .js or html file itself acts as a class, having some attributes and can perform certain functions depending on its capabilities.

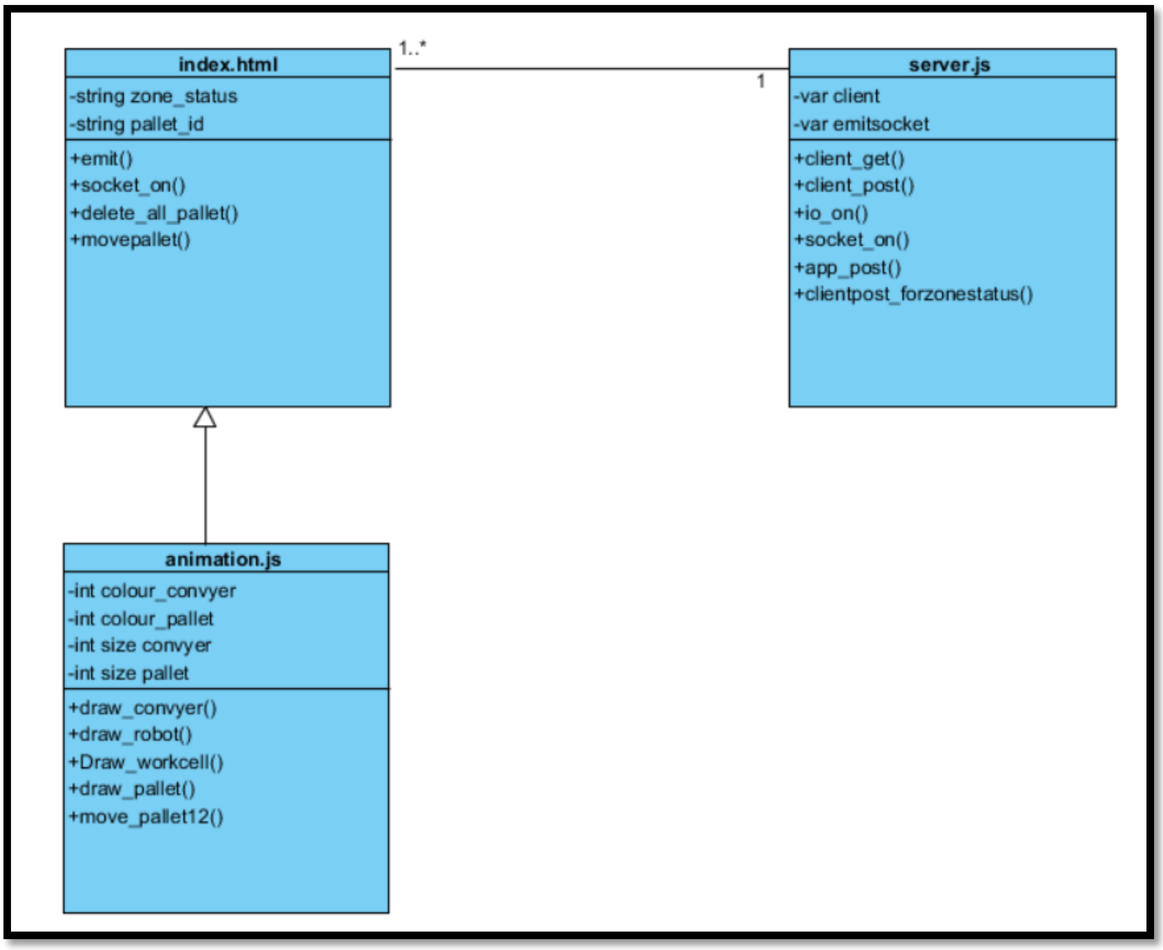


Fig 17: Class Diagram

Sequence diagrams: These diagrams have been modified to show the sequence of actions that take place when a user commands a pallet to be moved from zone 1 to zone 2 (sequence on top) and the sequence of events to monitor a movement of pallets from zone 1 to zone 2 because of a change happening on the simulator (sequence on bottom). As the scope was limited to the simulator and not to the physical equipment, the machine and PLC do not appear as the actors anymore.

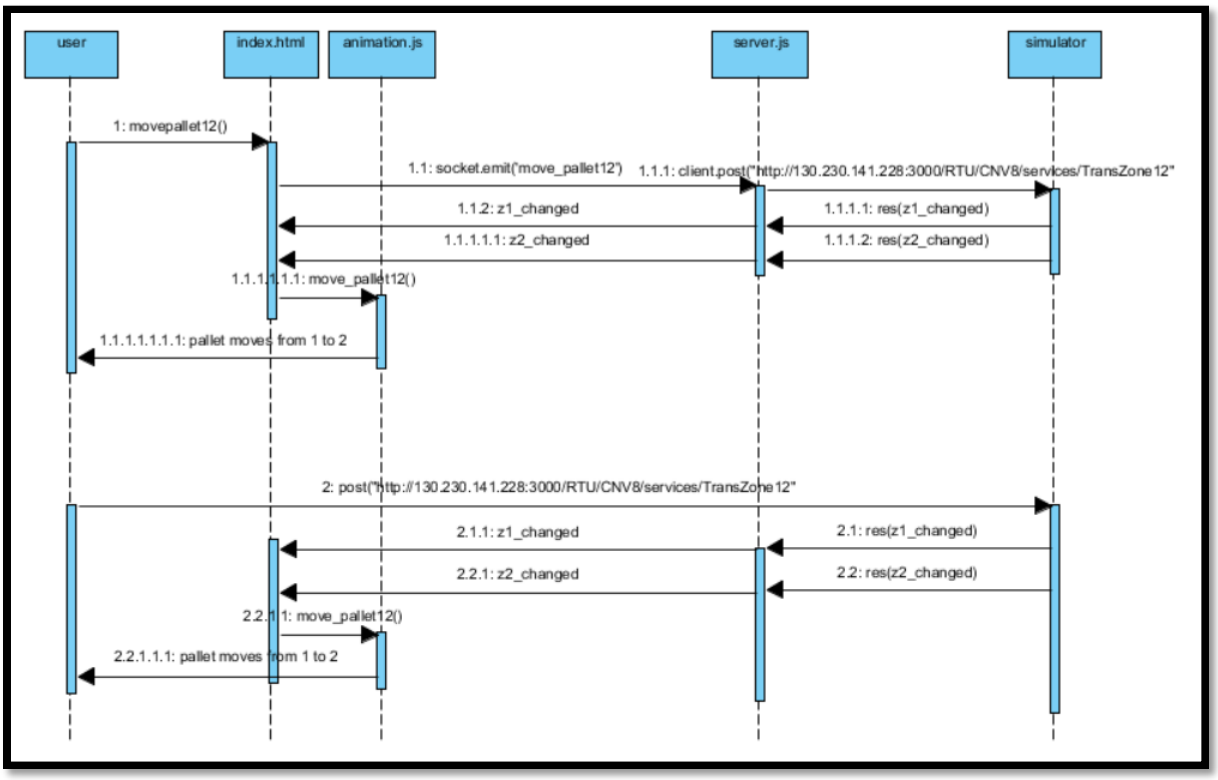

user
index.html
animation.js
server.js
simulator
1: movepallet12()
1.1: socket.emit('move_pallet12')
1.1.1: client.post("http://130.230.141.228:3000/RTU/CNV8/services/TransZone12"
1.1.2: z1_changed
1.1.1.1: res(z1_changed)
1.1.1.1.1: z2_changed
1.1.1.2: res(z2_changed)
1.1.1.1.1.1: move_pallet12()
1.1.1.1.1.1.1: pallet moves from 1 to 2
2: post("http://130.230.141.228:3000/RTU/CNV8/services/TransZone12"
2.1.1: z1_changed
2.1: res(z1_changed)
2.2.1: z2_changed
2.2: res(z2_changed)
2.2.1.1: move_pallet12()
2.2.1.1.1: pallet moves from 1 to 2

Fig 18: Sequence Diagram

State Diagram: The states of the robot are not being monitored. However, the state of the pallets are shown. The pallets goes to different zones (each being a state)

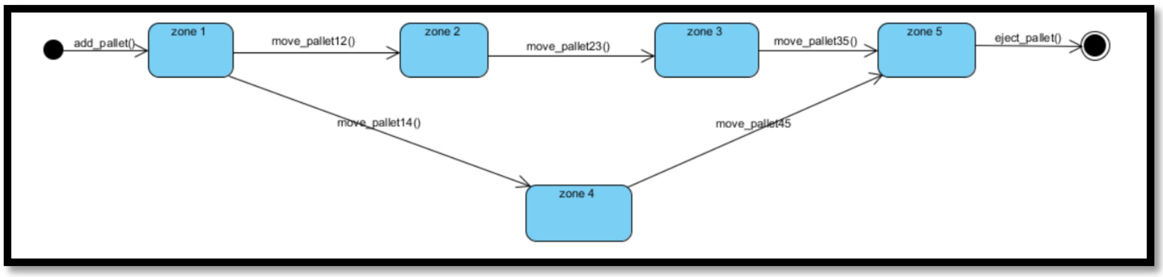


Fig 19: State diagram

Activity Diagram: the activity diagram is shown for the usecase add pallet/load pallet. It shows how from the server listens a command which indicates a change in zone 1, triggering the UI to show a pallet in Zone1.

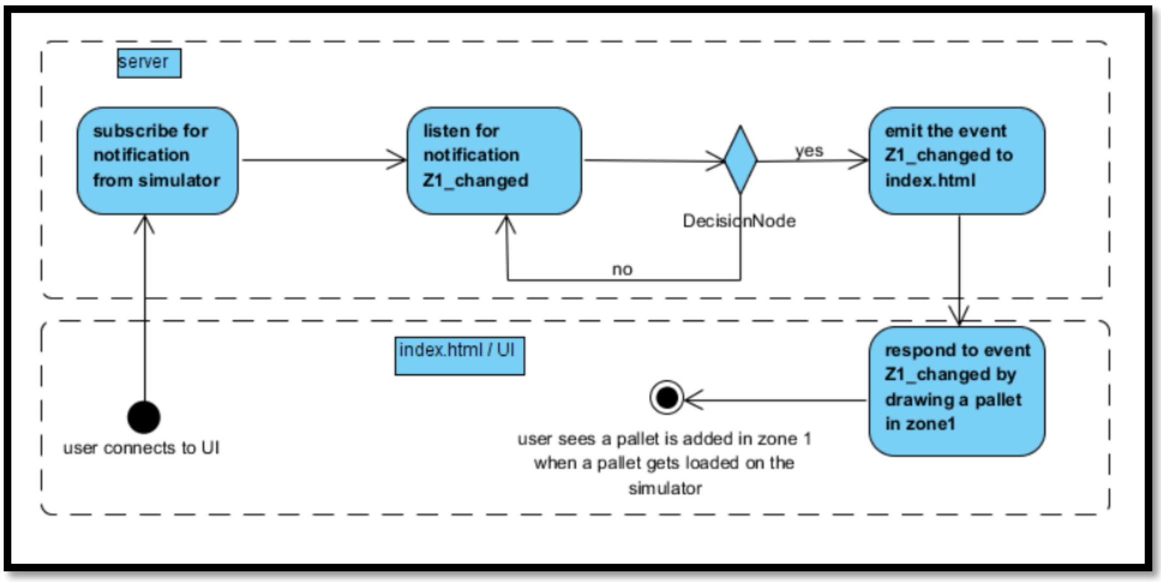

server
subscribe for notification from simulator
listen for notification Z1_changed
DecisionNode
yes
no
emit the event Z1_changed to index.html
index.html / UI
respond to event Z1_changed by drawing a pallet in zone1
user connects to UI
user sees a pallet is added in zone 1 when a pallet gets loaded on the simulator

Fig 20: Activity Diagram

6. Problems faced and learning outcomes

Learning outcomes:

- Learning JSON (Java Script oriented notation) data interchange format.
- Learning Socket.io to establish the connection.
- Learning SVG to draw the structure on the user interface.
- Learnt about HTML script through W3 schools.
- Learnt to manage time in a group. Also learnt to manage the work when a person in a group was sick and not able to work.

Problems:

- The connection of the socket was not so effective. Every time we reset the system, the socket was not getting completely connected in the sense some part of the data like pallet ID was not displaying the second time we run after the reset.
- It was really difficult to figure out why we are not getting data from the simulator when I was working from my home. I used to receive information when I was working with University's IP.
- Sometimes Socket connection totally fails, then we have to restart the console, simulator and UI.

Note: The attached folder will contain a video which shows the working of the Client user interface.

7. References

[1]. http://stackoverflow.com/questions/1789945/how-can-i-check-if-one-string-contains-another-substring

[2]. http://stackoverflow.com/questions/4750225/what-does-object-object-mean

[3]. http://stackoverflow.com/questions/22211966/javascript-cant-access-an-objects-keys-values-they-have-quotes-around-them

[4]. http://stackoverflow.com/questions/9768444/possible-eventemitter-memory-leak-detected

[5]. http://stackoverflow.com/questions/8313628/node-js-request-how-to-emitter-setmaxlisteners

[6]. https://nodejs.org/api/events.html

[7]. https://www.npmjs.com/package/body-parser#bodyparserjsonoptions

[8]. https://svg-edit.googlecode.com/svn/branches/2.5.1/editor/svg-editor.html

[9]. http://www.w3schools.com/tags/default.asp

[10]. http://www.w3schools.com/svg/default.asp

[11]. http://www.visual-paradigm.com/learning/