Design Documents

Team 1

Requirements

Functional Requirements:

The robot should:

- Pick up and place packages.
- Receive and follow control instructions from gateway.
- Identify tags.
- Know it's current position.

The server should:

- Receive temperature and light conditions from BlueMix.
- Store historic information about sensor readings over time.
- Find adequate place to store the package or refuse it.
- Order the robot to rearrange packages when readings change.
- Receive robot commands from the user
- Send commands to the robot through the gateway.

The Arduino boards should:

- Collect readings from sensors.
- Send current sensor readings to the gateway.

The Gateway should:

- Receive live stream from webcam.
- Send live stream to the server.
- Receive sensor readings from arduinos.
- Send sensor reading to BlueMix.
- Receive robot commands from server.
- Send commands to robot.

Non-functional Requirements

The sensors should be:

- Placed on each shelf/row.

The robot should:

- Navigate in the warehouse alone with no other robots there.
- Have a grip to pick up the packages.
- Save location context in x-y coordinates.

A package should be:

- Stored in an adequate location or refused.

The server should:

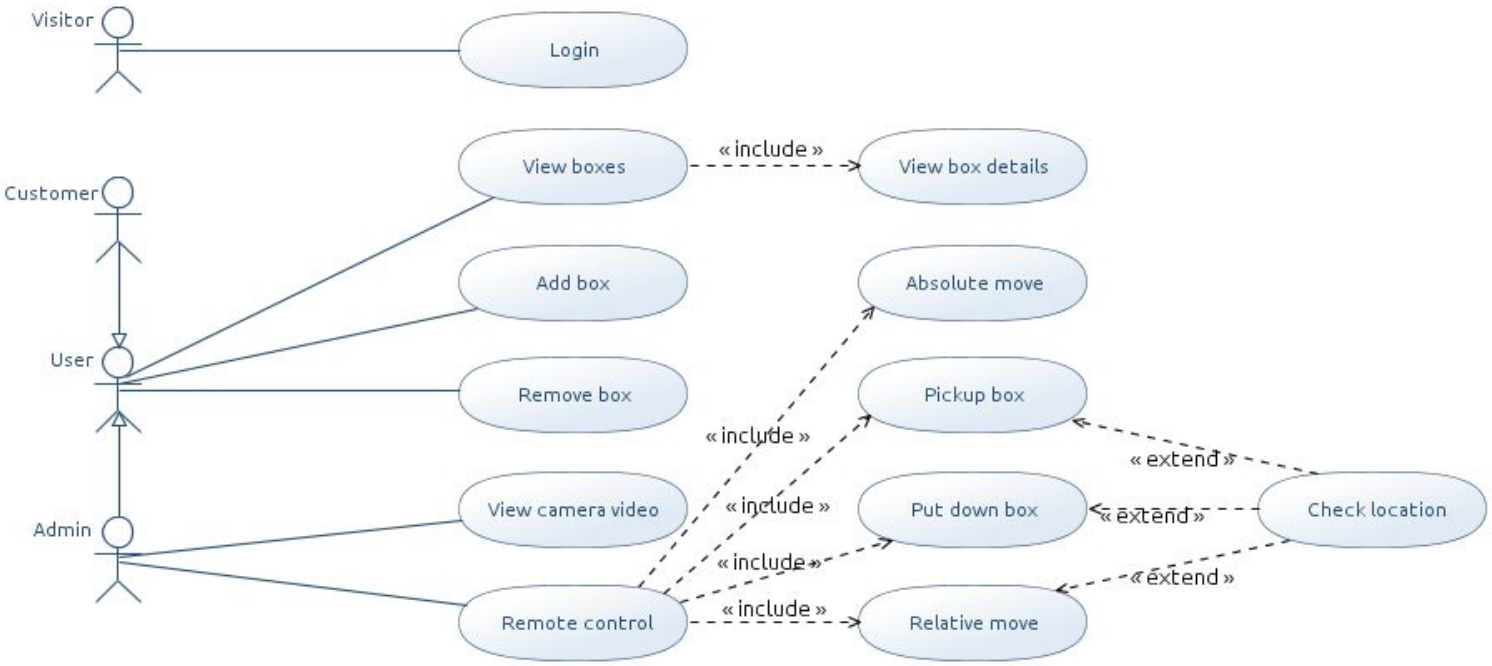
- Be updated with sensor readings every 30 seconds.
- Store the box requirement.

Limitations:

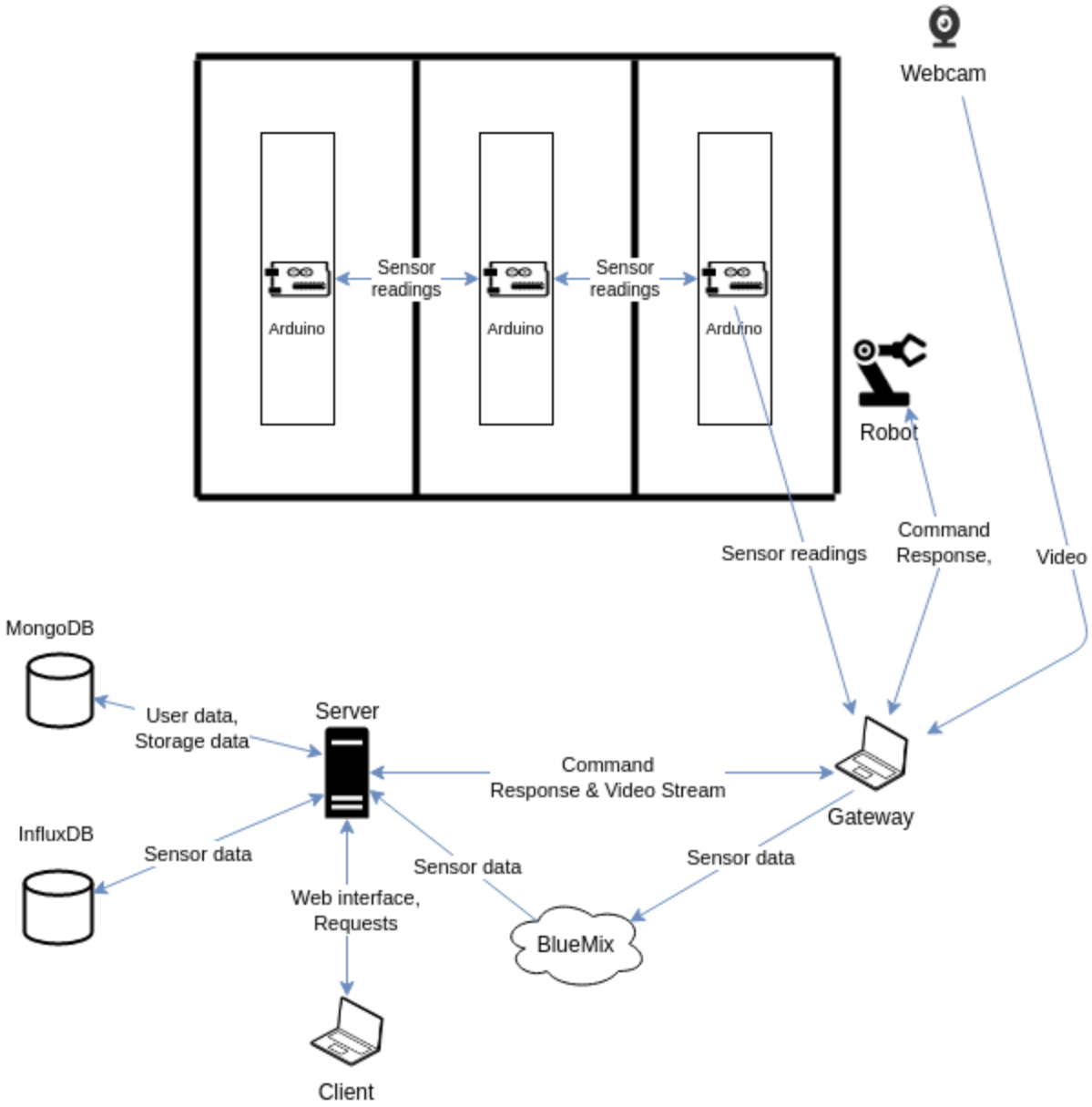
Robot

- The robot environment should only be able to handle one robot at a time.
- The robot can only store boxes that fulfill the measurements of the current test-box. The measurements are: 5.5, 4, 4 (W, H, D in cm)
- The robot can only operate within the predefined environment with the correct color tags and boundary lines.

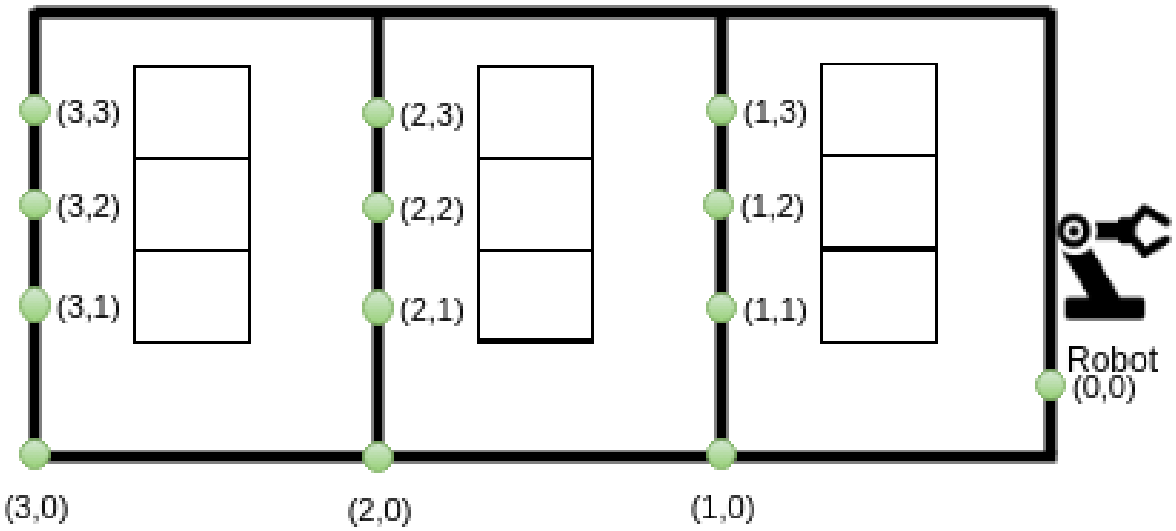
System Use Cases:



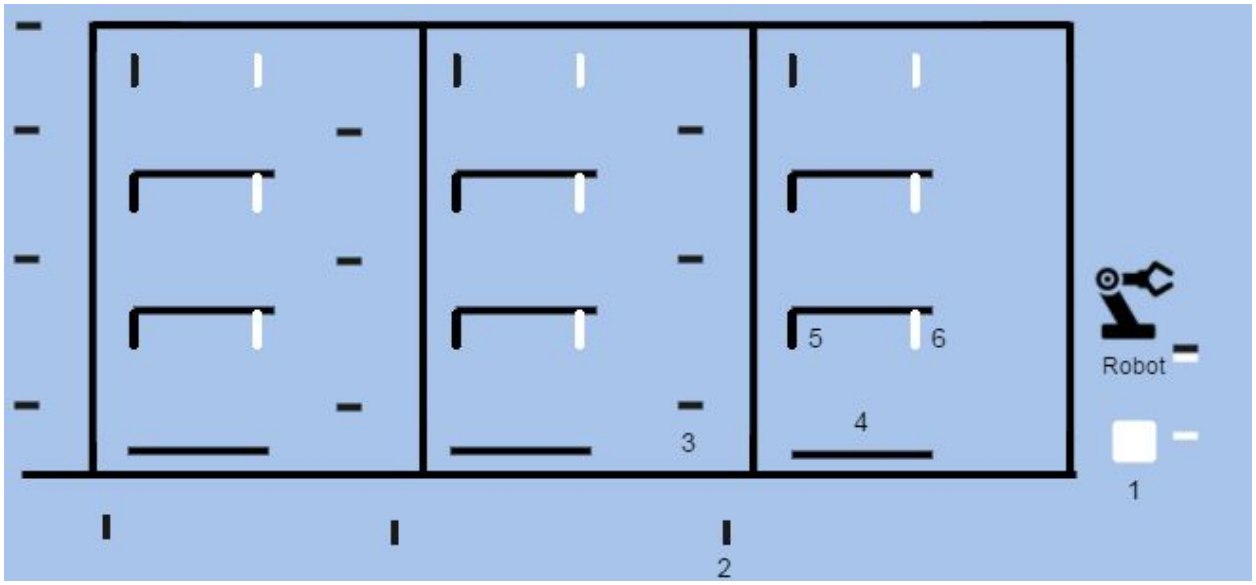
System Overview:



Warehouse Environment and Coordinates System:

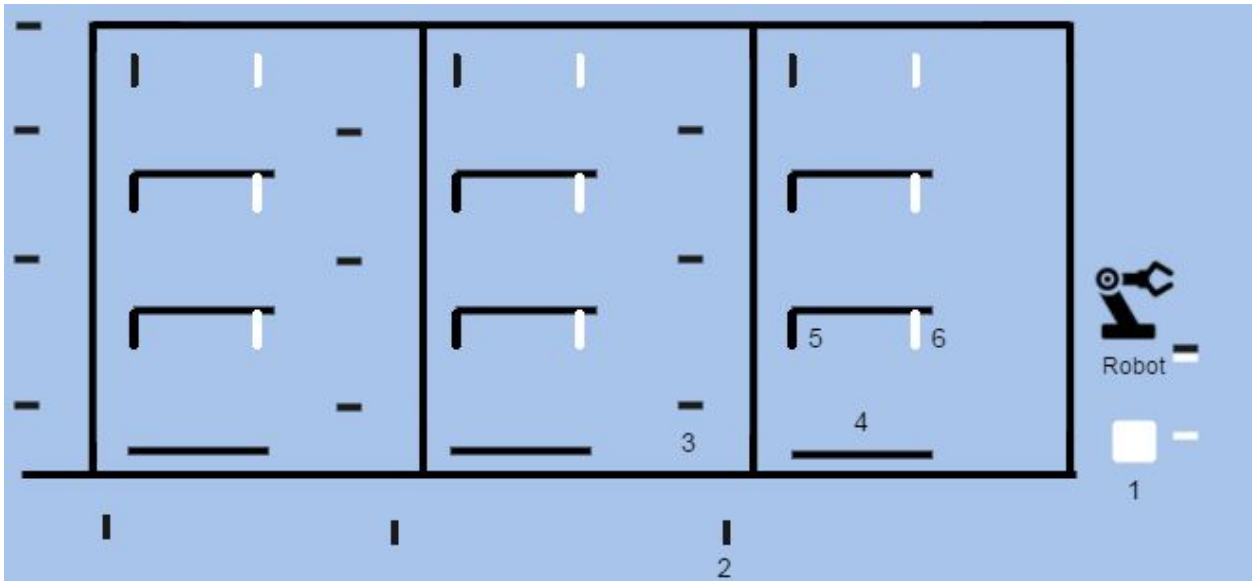


Navigation and environment details



Tag number	Explanation
1	Box pickup/extraction zone
2	Row indication tag. When the robot sensor find this it knows it is at the x row.
3	Shelf indication tag. When the robot sensor find this it knows it is at the y shelf and will turn to right if it should collect a box for that location.
4	Shelf boundary line. The robot always follow a line to keep level in regard to the environment itself.
5	Alignment indication tag. When the robot finds this tag it will stop turning and follow the line described in #4.
6	Box placement tag. When the robot finds this tag it stops and lifts the grip arm and back out of the shelf.

Environment measurements



- Environment:
  - Width: 156.5 cm
  - Width (bottom line): 164.5 cm
  - Height: 49.5 cm
- Tags [2,3,5,6]:
  - Width: ~ 2cm
  - Length: 4 cm
- Box pickup/extraction zone [1]:
  - Width: 8 cm
  - Length: 8 cm
- Shelf boundary line [4]:
  - Width: ~ 2 cm
  - Length: 20 cm

Robot navigation

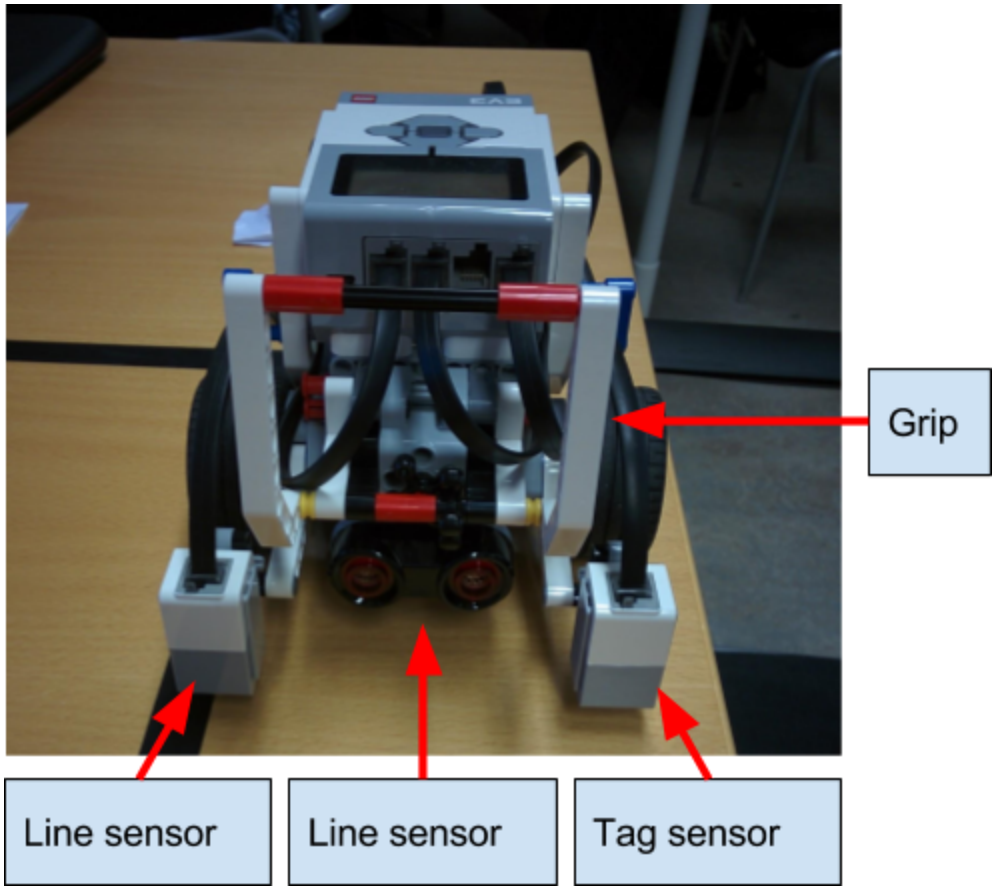
The robot navigates the system in a clockwise rotation. The robot starts at coordinate (0,0) and waits for commands. When a command is received the robot will proceed to the correct location and complete the objective. When the robot has completed an action it will return to coordinate (0,0) and then proceed to the location of the next objective to complete this operation. This may not be the most efficient way of utilizing the robot's power supply but it the implementation we chose to work with. We feel that the implementation works fine in our current environment with only 3 shelves but if an expansion of the warehouse were to be done then a better implementation may be needed. Our implementation could be modified to serve this purpose as well.

An example will be given below to explain how the robot navigation works. Let’s say for example that the robot has just dropped off a package somewhere in the first row. The next operation the robot receives is to remove a package from the third shelf. The robot wil then move to (0,0) and then move to the third row and the correct shelf to remove the box. After that last operation the robot will return to (0,0) to await further instructions.

The robot has been implemented with a location awareness system. This system helps the robot to know where it is in the environment. The tags that are shown in the environment figures are used by the robot to locate: positions (rows, shelves).

When the robot reaches the edge of the warehouse the final row tag will have been read by the robots tag sensor and the robot will turn right and follow the line along the third row. The robot will then follow the line, clockwise.

A picture of the robot from the front can be shown below.



Proposed Technologies

Server libraries:

The server is implemented using Node.js, and with the following libraries/services:

- Routing on the server: Express.js
- Websocket communication between server and client: Socket.io
- Platform to communicate MQTT sensor values: IBM Bluemix (IOT Watson)
- Persistent storage for user accounts, created boxes, shelves: MongoDB (NoSQL)
- Timestamp database: InfluxDB (NoSQL)

Everything on the server is written in Javascript.

Client-side scripting libraries:

- Jquery
- Javascript
- HTML
- Bootstrap
- CSS
- D3.js (data-driven-documents) for graphs

MongoDB will contain the following structures:

```
Object box = {
	createdBy: string,		// The user's username
	boxId: int,			// Autogenerated by MongoDB
	prefTemp: {			// Prefered temperature specified by the user
		max: int,
		min: int
		},
	prefLight: {			// Prefered light specified by the user

		max: int,
		min: int
		}
	},
```

```
        pendingStorage: boolean    // True when processed by the robot. When the robot
is done, it is set to false
}


Object shelf = {
        shelfId: string,                        // Auto Generated by MongoDB
        shelfLocation: int,                     // The physical location of the shelf,
according to our coordinate system
        shelfCapacity: int,                     // How many slots the shelf has in total
        boxes: [ boxObject, boxObject, ...]     // An array of the box Objects stored on
the shelf, indexed by physical coordinate location
}
```

Robot libraries

- Robot programming will rely on a Debian Linux distribution for ev3 called ev3Dev.

Gateway Technologies

- OpenCV to read camera input.
- Mosquitto will be used as an MQTT broker on the gateway.
- A Node.js server will run on the gateway.

Communication Protocols

The communication between the System and the server is handled through these communication channels:

Client-Server communication

- Arduino → Gateway → Bluemix →Server → Database:
  - The readings will be sent from the Arduinos to the gateway through Bluetooth.
  - BlueMix service will get a JSON object for each Arduino from the gateway.
  - The object has the following structure:

```
{ data: {
        sensorId: int,
        temperature_celsius: int,
        light_lux: int
}}
```

  - The server subscribes to the BlueMix category of each of the Arduinos and gets the objects.
  - The server will store the data in a time stamped InfluxDB.
- Webcam → Gateway → Telegram Bot API → Client:
  - The webcams will be connected to the gateway through USB.
  - The gateway will stream the video to the client using Telegram Bot API.
- Server ↔ Client:
  - The client will be a webpage which will communicate with the server using a websocket.

Server-robot communication:

- Server → Gateway → Robot:
  - The server will send commands to the robot through the gateway, as a JSON object, using HTTP requests.
  - The gateway will forward the commands to the robot using a Bluetooth link.
  - The commands will have the following format:

```
{ data: {
        command: string,          // Insert, Remove.. etc
        box: {
                id: string,       // The box id
                x: int,           // The shelf location
                y: int            // The slot on the shelf
        }
}}
```

Robot-Server communication:

- Robot → Gateway → Server:
  - The robot will send success or failure to the gateway after finishing a command through a Bluetooth link.
  - The Gateway will inform the server of the success or failure of the robot using an HTTP request by sending a JSON object.
  - The JSON object will have the following format:

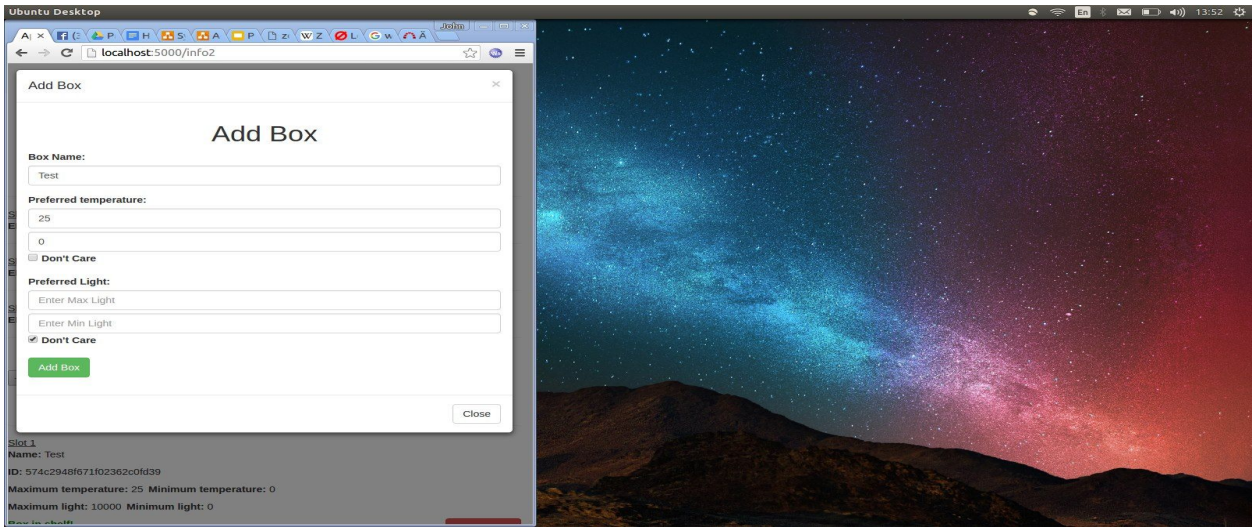
```
{ data: {
        command: string,     // Insert, Remove.. etc
        box: {
                id: string,   // The box id
                x: int,       // The shelf location
                y: int        // The slot on the shelf
        },
        status: boolean      // Success or failure of the robot
}}
```

Robot Pseudocode

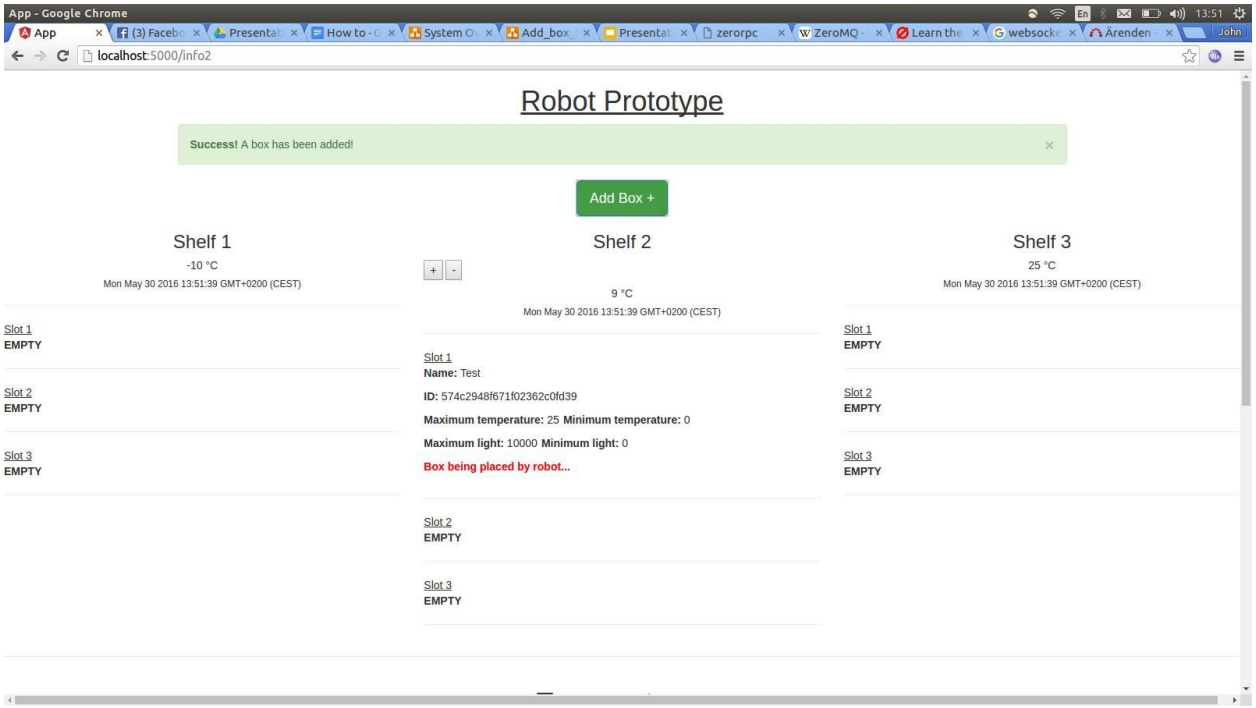
```
idle_state()
insert(x,y)
remove(x,y)
rel_move(udlr)
abs_move(x,y)
pickup()
putdown()

main() {
	idle_state()
	while(true)
}

idle_state() {
	abs_move(0,0)
	wait_event(‘server_command’ command_data)
	call command_data.command and pass command_data.data to it
}

insert(x,y) {
	abs_move(0,0)
	pickup()
	abs_move(x,y)
	putdown()
	abs_move(0,0)
	idle_state()
}

remove(x,y) {
	abs_move(x,y)
	pickup()
	abs_move(0,0)
	putdown()
	idle_state()
}
```

Interface Design

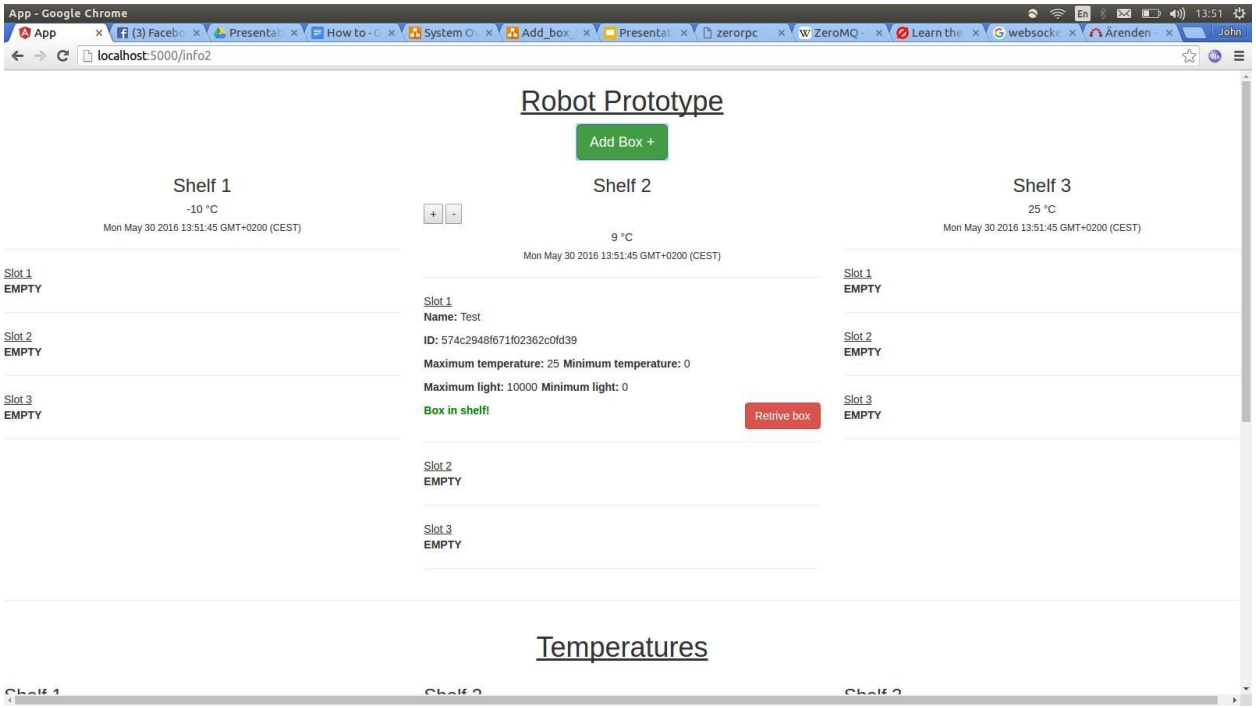
The image below shows how it look when the user adds a box.



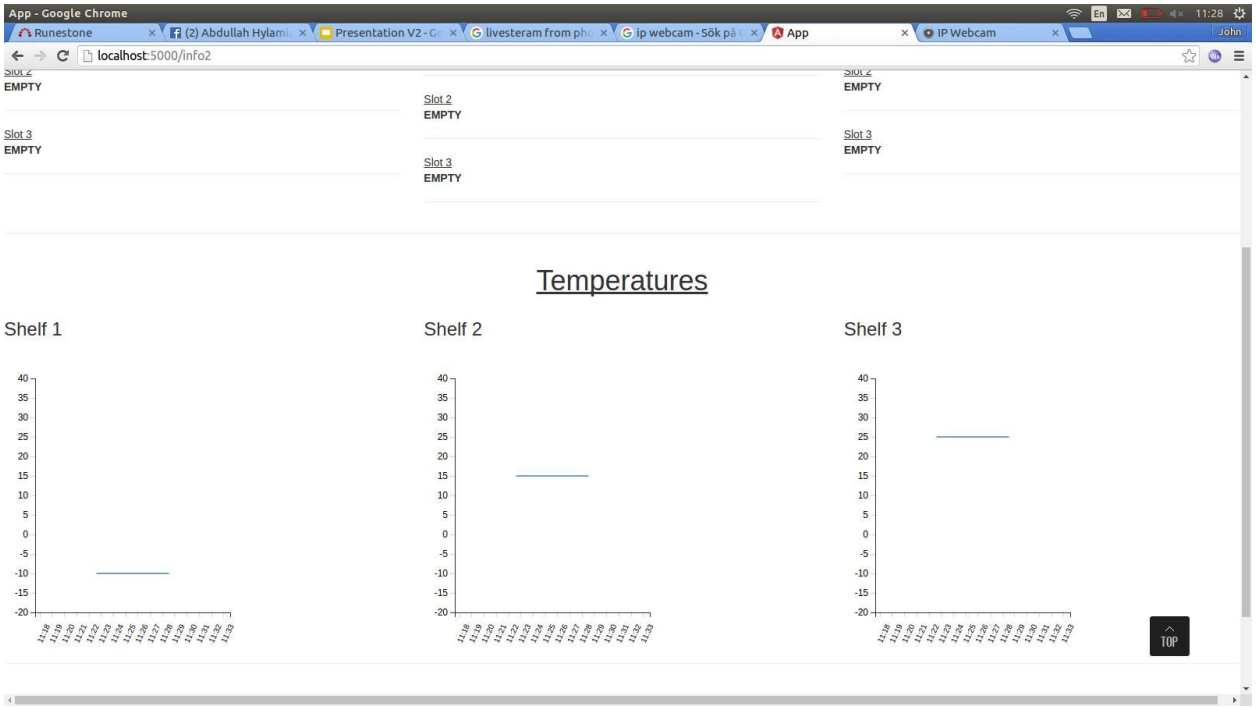
When the user has added the box, the interface shows the user where it is placed and that the robot is currently trying to add the box to a specific slot.



When the box is placed by the robot the red text changes to green and show the user that the box is now in the slot. Since the box now has been placed the user has the possibility to retrieve it.



The interface also provides a timeline of the temperatures in the shelves.



At the bottom of the page the user has the possibility to see a stream of the robot in the environment and move the robot by providing x and y coordinates.