SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I

BRODOGRADNJE

Assembly line documentation

Ivan Madunić, Marin Zglav

**SADRŽAJ**

[1. Description 1](#_Toc536097982)

[1.1. System architecture 1](#_Toc536097983)

[1.1.1. Node.js web server requirements 1](#_Toc536097984)

[1.1.2. Raspberry Pi workstations 2](#_Toc536097985)

[1.1.3. Assembly line 2](#_Toc536097986)

[1.1.4. Task schedule and excel logs 3](#_Toc536097987)

[1.2. Workflow example 5](#_Toc536097988)

[2. Server 7](#_Toc536097989)

[3. Raspberry Pi button 8](#_Toc536097990)

[4. Display 10](#_Toc536097991)

[5. Setup and requirements 10](#_Toc536097992)

# Description

Application assembly-line is a web/IoT based application for time tracking product workflow on an assembly line. The application is optimized for tablet displays and works in all modern browsers.

## System architecture

The whole system consists of 3 major parts:

1. Node.js web server
2. 4 Raspberry Pi workstations
3. Assembly line

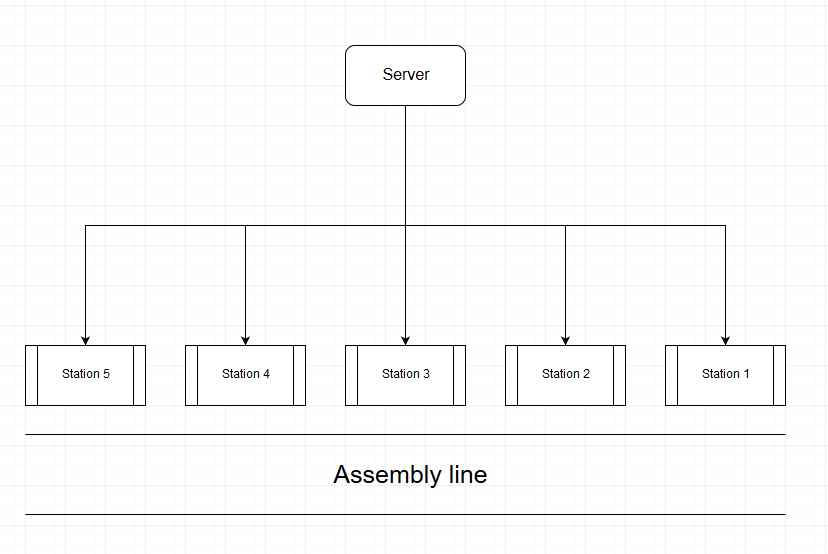


Figure 1. Basic architecture

### Node.js web server requirements

The web server is writen using Express.js framework. Servers primary requirements are:

* Rendering browser interfaces for Raspberry Pi workstations.
* Centralized time tracking for products on assembly line. Server should track total time the product spent on the assembly line, and 4 timers with following names: normal, t, m, z with the option to pause the timer. Also, time that product spent waiting between stations should be tracked.
* Real-time display of product worktime on opened browser interfaces. Each workstation should display worktime of current and previous workstation (i.e. workstation 2 should display workstation 1 and workstation 2 work process).
* Defining task schedule for products on the assembly line. When the workstation begins working on a product, the product is assigned a product name according to the task schedule. Task schedule consists of product name and quantity. Upon each assignment quantity is decremented for that workstation. When quantity reaches zero, task is removed from task queue.
* Displaying images depending on the workstation and product name.
* Reading average time from excel for corresponding produt.
* Writing unique product ID, product name, normal, t, m, z and wait time to excel. Writing is performed when each workstation finishes working on a product. When all workstations finish summary with total work time is writen to the excel.

### Raspberry Pi workstations

Raspberry Pi workstations are connected to the tablet display. Raspberry Pi's should have an installed browser and be connected to the same LAN as the server. 4 workstations are currently supported, but adding more workstations is straightforward. To start the workstation user should go to the following url „https://[server ip]:[server port]/[workstation number]“ (i.e. „https://192.161.1.25:3000/2“ to start workstation 2).

Each workstation should have a NFC reader connected, which upon reading the product tag should send a request to server with an assigned product name and tag ID. NFC tags should be reusable. Upon receiving the request server starts the currently set timer and sends a response. If the request succeded the workstation updates the image depending on the product name.

IMPORTANT: Due to faulty NFC scanners the button is used to simulate the reading of NFC tags. The documentation will continue refering to pressing of button as reading the NFC tag. Refer to section 3. to for more info on the button setup and usage.

### Assembly line

Assembly line should have products with arbitrary tags passing along it. Upon reaching the workstation tag should be scanned with NFC scanner. When read, the product is assinged a name according to the task list stored on server.

### Task schedule and excel logs

Task schedule can be defined using the taskList.json or taskList.xlsx files. To use the taskList.xlsx environment variable MODE in the .env file should be set to value „excel“. Task lists and excel logs are stored in the „/public/data“ folder.

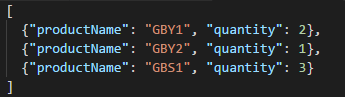


Figure 2. JSON task list

From figure 2. we can see that each task is defined with two object properties. First property defines the product name, while the second defines how many products will be in one sequence. For instance in the figure 2. example we can see that there will be two „GBY1“ products, one „GBY2“ product and lastly three „GBS1“ products.

The equivalent of the figure 2. JSON file is an excel file with the following structure:

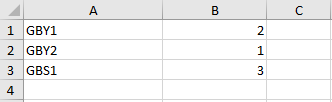


Figure 3. Excel task list

Average times work times for each product are defined in the input table.

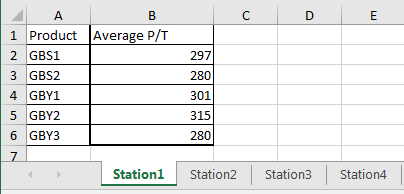


Figure 4. Input table

The input table contains worksheets with station names. Each sheet defines how much time the specific product spent on each workstation. For instance in the figure 4. example we can see that the GBS1 product spent 234 seconds in average on Station1.

Values are parsed from the input table using the utilty functions from „/utils/excelhandler.js“ file. Utilty function for parsing averge times from the input table is extractAverageTimes.

The output.xlsx file contains an event log of the product workflow. When each station finishes working on the product timers are logged to the output file. The outputTemp.xlsx file contains an empty/initial template of the output file, so if the user wishes to clear the log, he can just copy the outputTemp file and name it output.xlsx.

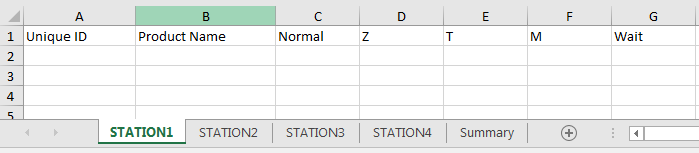


Figure 5. Output file

From figure 5. we can see the structure of the output file. Each product is defined with the unique ID (used to distinguish products with same name, it is generated randomly for each product) and product name. For each product 4 timers are logged and, if present, the wait timer is also logged. Each station has it's own worksheet containing it's product history.

At the end of the work process a summary containing the total work time of a specific product is writen as seen in the following figure:

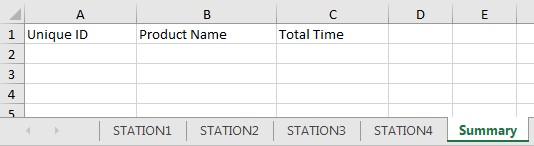


Figure 6. Summary sheet

It is important that the files retain their names. If you wish to rename the input, output and task list files, paths inside the code leading to those files should be adjusted.

## Workflow example

The workflow begins with product entering the assembly line. When the first workstation scans the NFC tag task is requested from the server, and afterwards the timer is started for the workstation 1. If no product with the same tag and data is present in the product queue a new one is added. Upon starting the timer the current task quantity for that station is decremented.

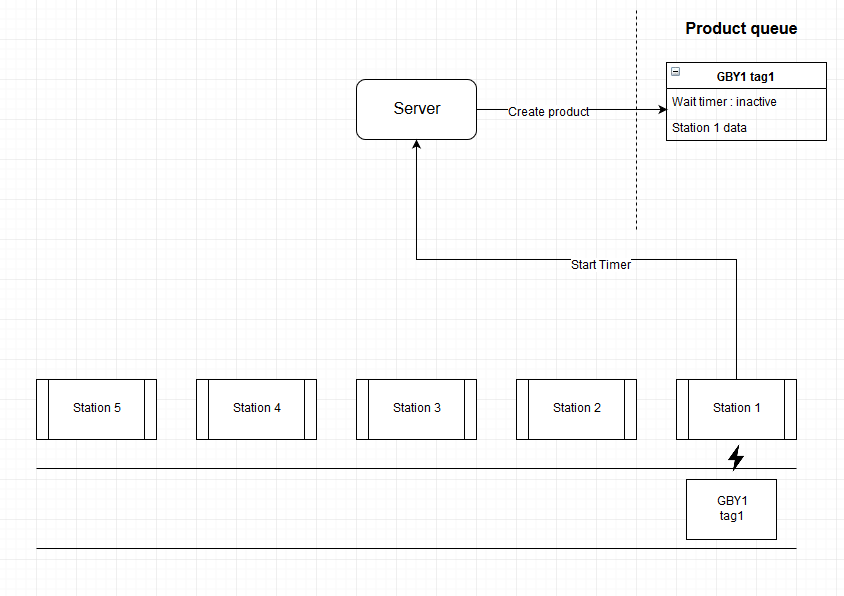


Figure 7. Product enters the assembly line

When product reaches the Station 1. it gets scanned. Upon scanning Station 1 sends start timer request to the server. Start timer is an HTTP POST request which is processed in „/routes/index.js“ file on a „/startTimer“ route. „/startTimer“ route creates a new product if there is no product present in the product queue. Product is created from the task list, as it was mentioned previously. Afterwards, the Station object is added to the stations array (property of Product object) if it's not already present. Objects are defined in the „/utils/datamodels.js“ file. Station object defines timers which are incremented via the timer function defined in the startTimer method of the Product object. The timer function messages the connected clients using socket.io. For each product current and next workstations are messaged. Each message contains product info, which is used to display needed information (product name, timers, average time) on the user interface. Timer can be stoped with stopTimer method.

Product work is finished when next product gets scanned. After the product finishes timers for corresponding station are stored to excel and wait timer is activated for that product until it reaches the next workstation. While the product is present at a particular station, the wait timer is inactive.

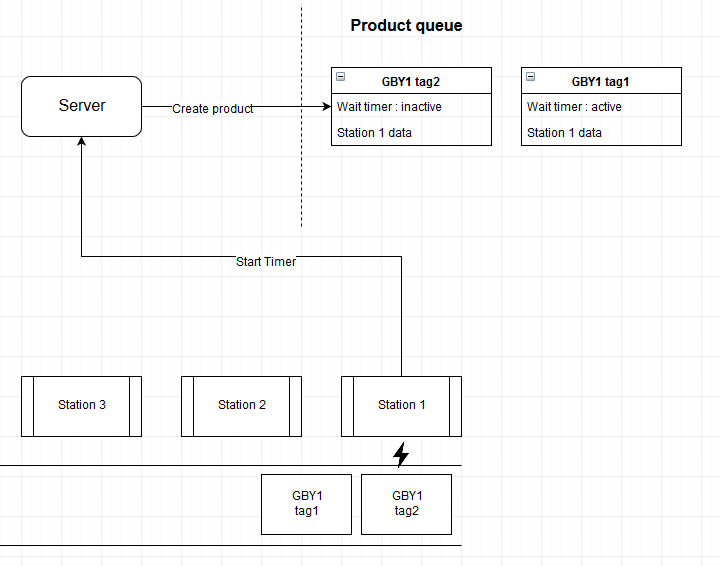


Figure 8. New product enters the assembly line

From figure 8. we can see that product GBY1 with NFC tag1 is not being worked on. Consequently it's wait timer is active. Also the new product that just entered the assembly line is being worked on and is now present in the product queue. Afterwards the first product continues down the assembly line which we can see in the following figure:

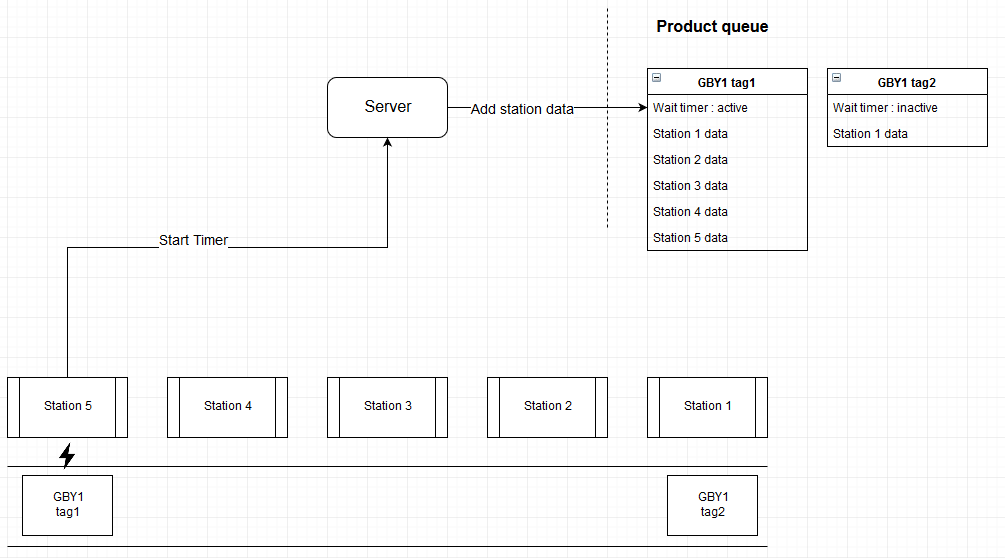


Figure 9. First product reaches the last workstation

We can see that product has reached the last workstation and that it's data object recorded work from all previous stations. Upon exiting the assembly line total time the product spent on the assembly line is writen to output excel file as mentioned previously.

# Server

Server is writen for Node.js environment using the Express.js framework. To set up the server it is neccesary to insert environment variables. Environment variables are read from the .env file. Currently present environment variables are:

* SERVER\_IP : represents the IP address on which the server is running. Use ipconfig (ifconfig on Linux) to read the IP address.
* PORT : Port on which the server will run. If the variable is left empty default port 3000 will be used instead.
* NUMBER\_OF\_WORKSTATIONS : Represents the number of workstations that will be used. Currently supported number is 5.
* MODE : If the user wishes to use the excel file to define the task list, mode should be set to „excel“. Otherwise the default setting will be used, which is the .json file.

Code for server is contained in the app.js file, while server routes are contained in routes folder. The application uses server side rendering, which means that the server renders the web page when user browser enters the correct URL address. Beacuse of Express.js framework PUG is used instead of HTML. PUG code is stored in the views folder. Static files (like images and CSS) servered by server are contained in the public folder.

Public folder contains images, CSS, excel data and client side Javascript code. Images are stored in the imgs folder. The folder needs to have subfolders with correct names. Names should correspond to product names set in the task list. Each folder should contain images of .jpg format with following names: STATION1, STATION2, STATION3 etc.. There should be no missing images.

As mentioned previously data folder contains task lists, input and output tables. Task lists are supported in .json or .xlsx format. User should adjust the task lists in correspondace to dummy files. Input data defines the time averages for each product and station. New products and stations can be added, but the file structure must remain the same as in the dummy file. Server stores workstation data to the output.xlsx file.

Client side Javascript code consists of index.js and socket.js. The application uses socket.io which is defined for client side in the socket.js folder. Both files are bundled to dist folder using npm run compile in the terminal.

Utils folder contains general purpose code which is used to implement the application logic.

For more details on how to start the server refere to README.md file in assembly-line repository.

# Raspberry Pi button

Connect the button in accordance to the following schema:

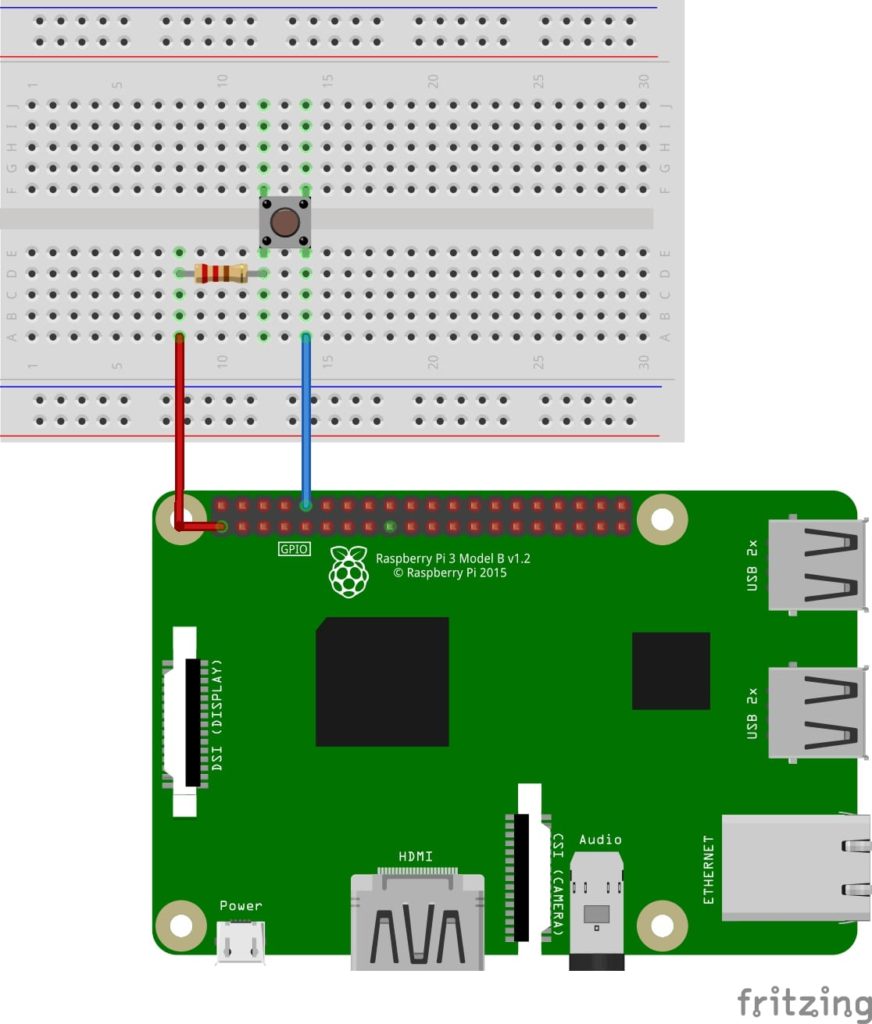


Figure 10. Button schema

To start the button use the button.py file in the assembly-line-button repository. Default pin used to connect the button to the Raspberry Pi board is pin number 10. To use a different pin change the value of constant PIN in button.py file.

Button is used to start the timer for the working product. When clicked button will send an HTTP request to start the timer if there is no currently working product. Fake tags and station name are used to indicate which station is starting work on which product.

For more details on button startup refere to the README.md in assembly-line-button repository.

# Display

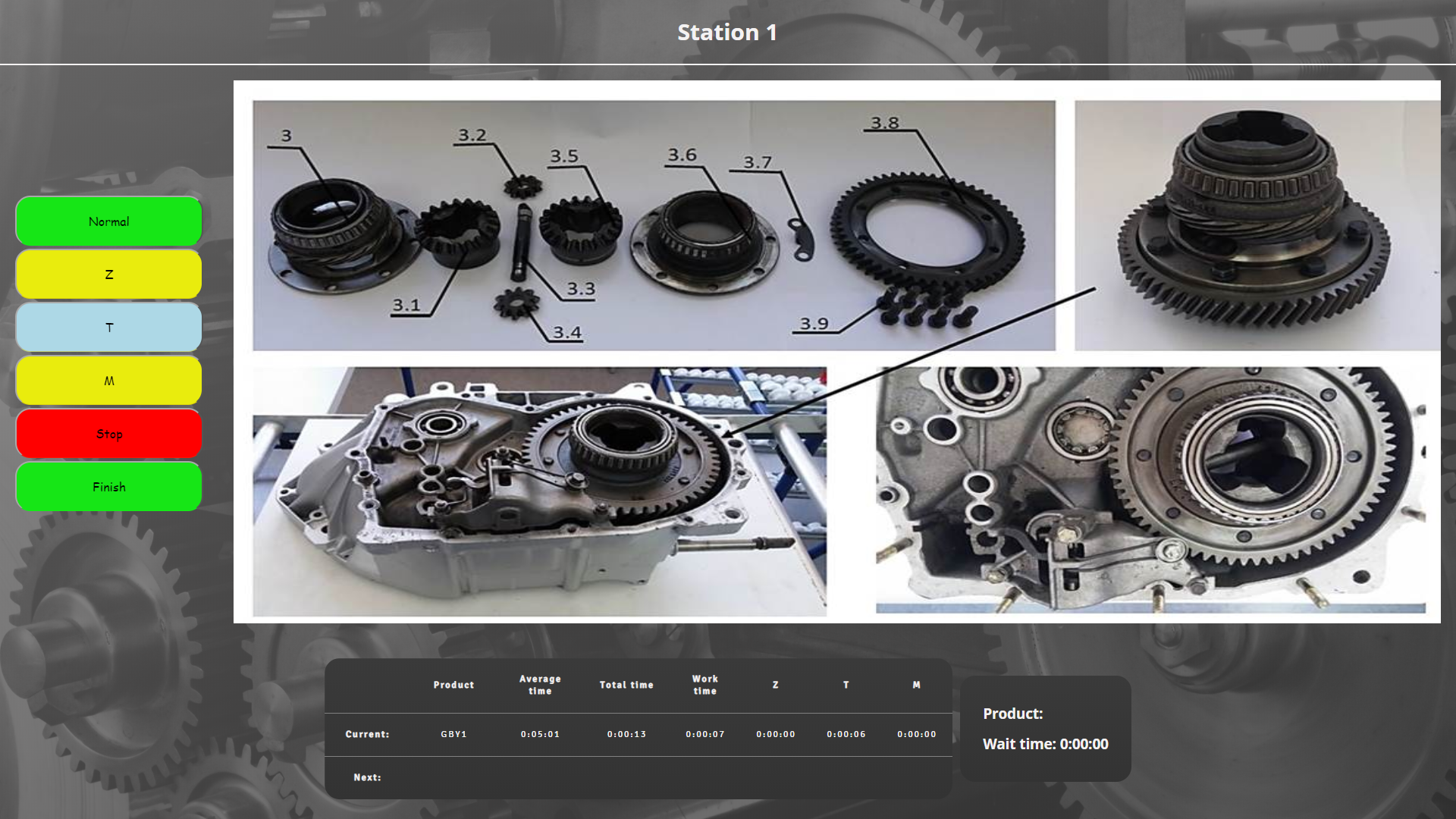


Figure 11. Working Station 1 example

User can interact with timers via the user interface on left - hand side of the display. Image displays instructions for that product.

Normal, Z, T, M buttons are used to control 4 available timers. Stop button is used to pause the work process. After the user finishes work on the current product he must press the Finish button to indicate that his work is done.

Timer table displays product name, average work time of a specific product and work timers. Row indicated with label Current displays product work info on the current station, while the row with label Next displays product work info on the previous station.

Wait table displayed to the right of the timer table indicates the product name and time it spent waiting between two stations.

# Setup and requirements

To run the server Node.js environment has to be installed along with npm package manager. Installer for Node.js can be downloaded on the following link: <https://nodejs.org/en/>. Installer includes the npm installation.

For file editing and running the server it is recommended to use the Visual Studio Code IDE. Server is started from the command line (terminal). VSCode supports the integrated terminal which can be enabled in the view dropdown menu. VSCode can be downloaded from <https://code.visualstudio.com/Download>. For more info on using the IDE consult the official VSCode documentation available on the offical webpage.

On Raspberry Pi's it is recommended to use the Chromium browser to connect to the webpage. Raspberry Pi's don't need Node.js, since they're only connecting to the server which servers them the webpage. Therefore only a browser will suffice.

To run the Raspberry Pi button Python3 needs to be installed. For more info on installing the Python3 on Raspbian OS see the guidlines on the following link:

“ <https://liudr.wordpress.com/2016/02/04/install-python-on-raspberry-pi-or-debian/>“

If the link is inactive keywords for googling a guide are: „Install python3 on raspbian“.