**Communication Specification Document   
(Order System, software).**

**Social distancing robot.**

**Fontys**

**Eindhoven**

|  |
| --- |
| **Project owner: Cees van Tilborg** |
| **Dat****e: 08-10-2020** |

|  |
| --- |
| **Project team:** |
| **Yves Lardenoije, Atanas Marchev, Bram Odenthal, Hugo Vaessen, Nicky Verhees, Martijn Vissers** |
|  |
|  |

#### Version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Editor(s)** | **Edits** | **Status** |
| V0.1 | 2-10-2020 | Hugo, Nicky | Start document | Concept |
| V1 | 8-10-2020 | Hugo, Nicky | Specifications and Protocol chapters have been made and filled. |  |
| V1.1 | 15-10-2020 | Hugo, Nicky | Reworked STARTORDER to given feedback |  |
| V1.2 | 11-12-2020 | Hugo Vaessen | Rework document to current code standards and json formats. |  |
|  |  |  |  |  |

**Distribution**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **To** |
|  |  |  |
|  |  |  |

**Table of contents**

1. Introduction 4

1.1 Background and Context 4

2. Specifications 5

2.1 General 5

2.2 Communication 5

3. Protocol 6

3.1 Headers 6

3.1.1 PLACEORDER 7

3.1.2 CANCELORDER 8

3.1.3 STARTORDER 8

3.1.4 EDITORDER 9

3.1.5 RETURN 9

3.1.6 STATUSUPDATE 10

3.1.7 RESULT 10

# Introduction

## Background and Context

Since the outbreak of the COVID19 virus, people who meet many others are potential super infectants. Examples of such people are waiters in restaurants and terraces, or nurses who distribute meals in hospitals or nursing home. These people have tasks that can be automated by robots which will in turn lower infection rates and unnecessary contact will be avoided.

Robot solutions for social distancing are already on the market, but they are either expensive to purchase or provide limited in functionality. The professionally deployable solutions also require a high level of technological infrastructure and specialist personnel.

In this document the communication between the devices of the customer, staff, server, and robot will be specified. This includes protocol between the devices.

# Specifications

Specifications of the communication between the customer, staff, server and the robot.

## General

The ordering process can be split into parts. The process begins with the customer or employee placing an order for a specific table. When the order has been placed the bar will prepare the order and let the robot bring it to the table.

## Communication

For communicating between the tables and the bar and robot, sockets will be used for sending and receiving data. This can be a simple TCP/IP socket or a WebSocket solution would also be sufficient.

We do have to keep in mind that the order dashboard also needs to open a connection to the server when a customer wants to order, this means the picked solution for communicating should also be available for web projects (for example in JS).

When placing an order, the active session of the customer will send the order specifications to the server. All communication data is converted to a byte array and based on its incoming header the server will handling accordingly. After the order has been prepared by the staff, another message will be transmitted to the socket of the robot to instruct the robot to start handling the order(s).

# Protocol

There are multiple different messages sent over the socket to the client and/or server. All messages sent are formatted as JSON string and will be deserialized when received for handling the message and its data. Messages exists of a header and data; the content of the data depends on the header.

## Headers

Each received message contains an identifier (header) to specify what kind of data or action is received and the system will call the correct function to process it. The headers are defined in an Enum which is shared between the client(s) and server.

|  |  |  |  |
| --- | --- | --- | --- |
| Header | Value | Description | Direction |
| **PLACEORDER** | 0 | Used to create a new order | Staff à Server  Customer à Server |
| **CANCELORDER** | 1 | Used to delete an existing order | Staff à Server |
| **STARTORDER** | 2 | Used to let the robot start bringing certain order(s) | Staff à Server  Server à Robot |
| **EDITORDER** | 3 | Used to edit an existing order | Staff à Server |
| **RETURN** | 6 | Used to let the robot return to its starting position | Staff à Server  Server à Robot |
| **STATUSUPDATE** | 4 | Used to update the status of an order when the robot has an update available | Robot à Server |
| **RESULT** | 5 | Used to indicate how handling the previous header went | ALL |
| **BILL** | 7 | Used to request the bill from an order. | Client à Server |
| **ID** | 8 | Used to identify the client(s) on the server. | Robot à Server |
| **GETOPENORDERS** | 9 | Used to get the current open orders. | Client à Server |

### PLACEORDER

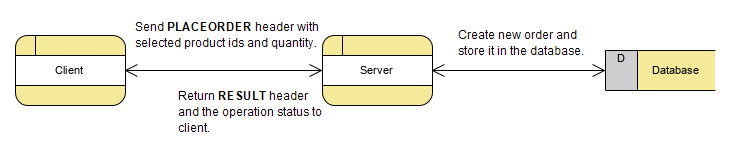
To place an order the customer must scan the QR-code on their table to open a web-app.

Alternatively, an employee can place the order on their handheld device they normally use.

Within the web-app, the customer can see the menu and order any item available, like what an employee can order.

After the payment of an order has been confirmed the customer- or employee’s device will send the PLACEORDER header to the server. After that the server will send the order to the bar so the order can be prepared.

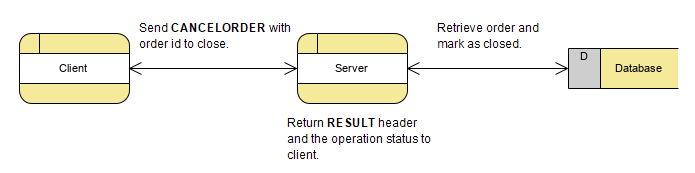
|  |
| --- |
| {  "header": int,  "table": int,  "totalPrice": int,  "products": [  {  "product\_id": int, "quantity": int  "product\_id": int, "quantity": int  },  ]  } |



### CANCELORDER

Cancelling an order can only be done by the staff. This can be done by sending CANCELORDER to the server, which will delete the order with the specific order id.

|  |
| --- |
| {  "header": int,  "order\_id": int,  } |



### STARTORDER

This header uses data which contains the specific order that will be send to the robot and will be brought by the robot.

The social distancing robot must first be filled with the order, by the staff, before starting to bring the order. Be aware that the robot has 2 tray locations inside itself. The staff will select the order(s) that have been placed on the robot and send STARTORDER to the server. The server will then send the same header to the robot, which will start bringing the order to their destination.

In case the order is too big to be delivered in one run, the *orderComplete* Boolean will be set to false, indicating the current tray the robot is carrying does not contain all drinks of the order. In this case the STATUSUPDATE header will not be sent for this order (to complete).

|  |
| --- |
| {  "header": int,  "orders": [  {  "order\_id": int,  "orderComplete ": bool  "table\_id": int  "tray\_location": int  }  ],  } |

### EDITORDER

Editing an order can be done by the staff when any mistake has been made. The “old” order will be copied, and edits can be made. After editing, confirming, and paying the “new” order, EDITORDER will be send to the server. The server replaces the “old” order with the “new” order with the same order id.

|  |
| --- |
| {  "header": int,  "order": {  "order\_id": int,  "status\_id": int,  "products": [  {  "product\_id": int, "quantity": int  "product\_id": int, "quantity": int  },  ]  }  } |

### RETURN

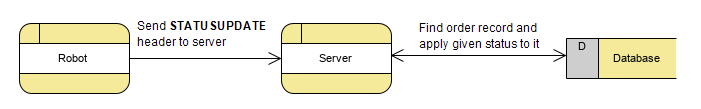
The RETURN message can be sent by staff to let the robot return to its starting position, if for example the robot has been gone for a long time. The message will be sent to the server which will send it to the robot right away. This header does not contain a payload. As an extension this payload could contain a certain position to return to (specific loading station or any other specific location).

|  |
| --- |
| {  "header": int,  } |

### STATUSUPDATE

This header is sent by the robot to let the server know that the status of a certain order has been changed. For example, when the robot finished delivering an order to a table it will update the server that the order has been given.

|  |
| --- |
| {  "header": int,  "order\_id": int,  "status\_id": int,  } |



### RESULT

Used to indicate how handling the previous header went. This can mean that for example, when sending EDITORDER, the order does not exist so the return bool will be false, and the message will contain more details about the error. When successfully handling the previous header, the result bool will be set to true.

|  |
| --- |
| {  "header": int,  "result": bool,  "message": string  } |

### BILL

The bill header is used to get the bill of a certain table which is passed inside the message structure of the bill. The orders that have been delivered to that table will be returned back as response.

|  |
| --- |
| {  "header": int,  “table”: int,  } |

The response will look as following and the header will also be BILL:

|  |
| --- |
| {  "header": int,  "totalPrice": int,  "products": dictionary<int, int>,  } |

### Bill

The ID header is used to identify a client on the server. For example, when the robot is connected to the server, it will send the ID header along with its name to the server for future referencing.

|  |
| --- |
| {  "header": int,  "name": string  } |

### GETOPENORDERS

Used to retrieve the currently open orders from the server. When this header is received, all orders that are not marked as finished will be returned back in the following structure:

|  |
| --- |
| {  "header": int,  “orders”:  [  {  “idx”: int,  “table”: int,  “totalPrice”: int,  "products": dictionary<int, int>,  }  ]  } |

The products property is an array of the structure shown above.