**Order System Proposal**

**Social distancing robot.**

**Fontys**

**Eindhoven**

|  |
| --- |
| **Project owner: Cees van Tilborg** |
| **Dat****e: 02-10-2020** |

|  |
| --- |
| **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 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Distribution**

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

**Table of contents**

1. Introduction 5

1.1 Background and Context 5

1.2 Definitions 5

2. Proposal 6

2.1 General 6

2.2 Order dashboard 6

2.3 Order dashboard (API) 6

2.4 Websockets 6

# Introduction

## Background and Context

This document is written for the purpose of proposing different types of solutions for handling orders in a bar or restaurant situation.

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.

## Definitions

|  |  |
| --- | --- |
| RIGS | Robot is geen super verspreider. |
| Component | A server that can contain databases, websites, etc. |
| Websockets | TCP communication protocol solution. |
| API | Application Programming Interface. |

# Proposal

## General

For the purpose of handling the orders of the customers at each table, we agreed on the usage of a website for showing and managing the orders.

The choice of a website gives us flexibility, we could redirect users to the menu or list of drinks using the (usually) already available QR code. These codes are (post COVID) commonly used for registering at the restaurant or bar the person is at.

## Order dashboard

In this design, only 1 component is required. The single component contains the database and the website. This means the bar personnel and customers both use this component for ordering and for managing orders at the bar.

The pro of this setup is that only a single component is necessary, which makes it easier to setup and use, this also means the price would be significantly less than solutions that use multiple components.

The downside is that in this scenario that it’s not possible to react instantly to events that take place on the website, such as sending instructions to the robot or to instantly notify the bar that a new order has been placed.

## Order dashboard (API)

This is slightly the same design as the first proposal (Order dashboard). But uses a central API for sending and retrieving information from the database. Information will be obtained from the API and new orders will be stored using the API endpoint. Using polling at a certain interval the dashboard for the bar personnel will be updated to list newly placed orders at the tables.

The pro of this environment is that the API can be accessed from any component (if we would like to), which gives us more scalability for the dashboard. The limitation is again reacting instantly to events, there will usually be some delay.

In figure 2 you see an example of this setup, please note that this solution can be a built as single component if necessary (The table also uses the order dashboard for placing an order, but on the customer side of the dashboard).

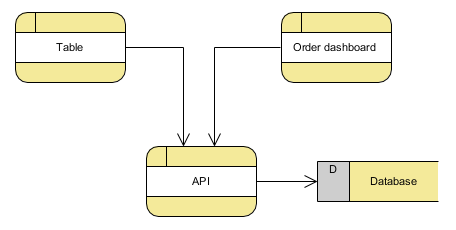


Figure 1 Example of the API setup with the dashboard

## Websockets

In this design websockets are used so it is possible to have a two-way communication between a webserver and a client which works in real time.

As previously mentioned, the guest at tables will order their drinks by scanning a QR-code. When scanning the code their device will connect to the server as a client of a table (see Figure 1). When an order is received by the server, it will be sent to the bar right away. This means very little waiting time between receiving orders.

The server can send and retrieve data from the database whenever this is needed.

The guests will only have to connect to the server if they want to place an order. This means that it is possible to have multiple clients of the same table, which order drinks at the same time. The bar will then get all these orders separately.

A big advantage is that the clients or server push their data instead of polling for updates. This would make the server poll constantly for changes in which it can’t do other things.

A downside of websockets is that it requires a reliable communication, which usually aren’t available in outside areas. There will need to be a retry/reconnection strategy that will always make sure an order is correctly send to the server. To prevent frustration, this must be done without cancelling an order and making the guest order again.

This solution requires the most development time.

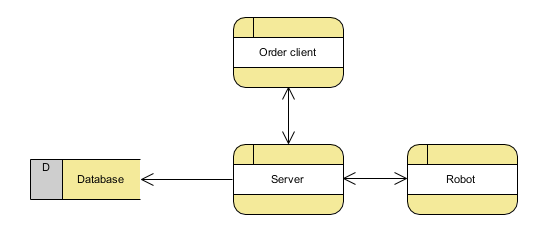


Figure 2 Example of websocket setup (the table is the client)