Transfer Document

Group:	DB05-1
Course:	Software
Teacher(s):	Marc van Grootel
Date:	23-12-2021
Version:	V1

Contents

Version history

Version	Date	Author(s)	Changes
V1	23-12-2021	– Jurgen Kalkers – Remco van Swaemen – Teun van Brakel – Wesley Arends – Mathijs Jansen	– Crew – Introduction – Documentation – Front-end – Backend – CI/CD

Crew

Jurgen Kalkers

- Mail: 435914@student.fontys.nl

Remco van Swaemen

- Mail: 479084@student.fontys.nl

Teun van Brakel

- Mail: 450192@student.fontys.nl

Wesley Arends

- Mail: 477269@student.fontys.nl

Mathijs Jansen

- Mail: 455150@student.fontys.nl

Introduction

Rigs wants a robot that will serve orders in restaurant. The goals of our project is to create an application to manage these robots. This application will be divided into three parts. The admin panel where all the settings and menus are set up; employees panel where the employees can communicate with the robot and complete orders and finally, the customers app where customers can place and view their orders.

In the chapter documentation all our documents are described with what information they hold. In the chapter front-end all pages are explained with the components that are used. The backend chapter explains the structure of the backend and what each component does. Next, CI/CD explains how our workflows work within GitHub and what we did to deploy the application.

Link to our repository:

https://github.com/horeca-robot/customer-app

Documentation

Samenwerkingsovereenkomst

Within this document you can find all agreements between the team members. How the documents need to be organized and where you can find the product. This contract basically holds all information agreed upon between team members at the first week of the project and the consequences that will happen when these team members don't keep themselves to the contract.

Project Plan

Here you can find the context, goal and the scope of the product. The scope is divided in who the stakeholders of the product are; what research method we use; how we communicate; how we will test the product and the budget for the product

Agile Research

In the Agile Research document, we wrote about our research of different agile methods. This research is written following the DOT framework. First, we decided what criterium would be important for this product. Then we researched and put the methods in a longlist with this criterion, to see which of the two was most applicable for the product. From this we conducted a brainstorming session to see which method was most known between the team members and concluded that Scrum is the best subject for this semester.

Design Document

What you will encounter in this document is the general styling and fonts that will be used throughout the entire application. The colours are changeable, but there has been documented a primary- and secondary colour to show where these colours should be applied. This document also concludes of wireframes and high-level designs.

Process Analysis

Within this document is a general analysis of the process without the use of a service robot and the process that would become the new process with the use of a service robot.

Ethics Document

In the Ethics document, the outlook of the different team members on the general ethics of this service robot is described and a few other topics. From these different outlooks we concluded on the topic.

Functional Design

The purpose of this document is to get a general overview of all requirements for this product. The most important ones have been divided into use cases and test cases, to ensure that the vision of these requirements is aligned within the team.

Technical Design

Finally, in the technical design document is located the technical side, as the name sounds, aspects of the product. Here you can find the entity diagram, this links all entities used for this product; the database diagram, to show what tables and columns there are to define the entire product and the class diagrams, to show the general structure of the project.

Front-end

Our front-end is made in a Vue application. All the pages people can see can be find under views. These views have different components underneath which can be find under components. Foreach page I will give a brief overview of which files are used, where you can find them, and which functionalities were implemented. Underneath each page title you can find the explanation of it.

Menu

The view for the menu page can be found in the folder src → views → Menu.vue. This page uses different components to display all the categories that are on the menu. The components CategoryCard, SearchBar and CartButton are imported to use them on this page. The SearchBar and CartButton will be explained later in this document, but I will explain the CategoryCard component in more detail. The CategoryCard can be found in the folder src →components → CategoryCard.vue. in this component there is a method called GoToCategory. This method is responsible for going to the right page when the user clicks on the category. If you are on the menu page, you will be directed to the category page. Furthermore, is the category image and name displayed on the menu page to tell the users which category belongs to which card.

Category

The category page can be found in the folder src → views → Category.vue. This page uses the components SearchBar, CartButton, BackButton and CategoryCard. The first 3 components will be explained later in this document however the CategoryCard is a little different for this page. Because when you click on this page instead of directing you to a category page you will be directed to a subcategory page. This is done in the GoToCategory method. If there are products available for a category it will go to the subcategory otherwise it will open the category page.

Subcategory

The view for the page Subcategory can be found in the folder src → views → ChildCategory.vue. This page imports the following components: ProductCard, SearchBar, CartButton and BackButton. Only the ProductCard is a specific component which I will explain in this page. Furthermore, has the view a method called filteredProducts. This method will be explained underneath CategoryFilter. The ProductCard component can be found in the folder src → components → ProductCard.vue. In this component you can see that all the information of the product is shown. Furthermore, you have a couple of methods named addToCart and GoToProduct. The addToCart isn't used on the subcategory page. The function that is important for the Subcategory page is the GoToProduct method. This method allows the component to redirect the user to the product specific page when it clicks on that product. This is done by adding the categoryId and productid to the params.

Product

The Product page can be found in the folder src → views → Product.vue. The page imports the Byproduct, CartButton and BackButton component. The Byproduct will be explained here the others will be explained under the custom-tags. This has a few methods for adding the products to your local storage. The Byproduct component can be found in the folder src → components → Byproduct.vue. All the functions that are related to the by-products are shown in the Byproduct component.

Your order

Your order page can be found under src → views → Cart.vue. This vue page consists mostly out of methods to add or remove products from your cart. Also, it contains a method called OrderingItems to order the items with each other. The CartItemCard component is imported in this page that can be found under src → components → OrderItem.vue. This component allows us to display the various items on a good and easy way. In the cart you also can add a note to your order and change the note and products you have ordered.

Orders

The orders page can be found in the folder src → views → OrderHistory.vue. this imports the OrderHistoryItem component, which can be found in src → components → OrderHistory → OrderHistoryItem.vue. the OrderHistoryItem contains methods for going to a specific order, to see the order information at the order info page, and counting how many products are delivered to your table. The orders page itself contains a function which adds the functionality of downloading a bill off all your orders.

Order info

This page can be found in the folder src → views → OrderHistoryDetails.vue. This page consists of functionalities to count the delivered products for the specific products but also it contains a method to put the same products together. This page imports the CartItemCard which can be found in src → components → OrderItem.vue. However, this page doesn't include the remove and add button even ass the functionality to add a note to the product.

Home

The home page can be found in the folder src → views → Home.vue. This page consists of the TableQRScanner and TableListPicker component that ables the user to login on a certain table. These components can be found in src → components → TableListPicker.vue and TableQrScanner.vue. The tableListPicker displays a list of all the tables where the user can click the table when he or she doesn't have a camera. The TableQrScanner opens a scanner to scan the QR code that is on the table. Furthermore, when you log in on a table you will be redirected to the TableValidator.vue page after which you will be send to the menu page.

Custom-tags

Besides the page specific pages, we also have some components and functionalities that can be found back on every single page. For each of these components I will give a brief explanation where to find them and which functionality is used.

Back button

The back-button allows the user to go back to the last page and select other pages. This will be available in almost every page. You can find the button underneath src → custom-tags → backbutton.vue.

Cart button

The cart button is a button that will be shown on every page in the header. This button can be found in the folder src → custom-tags → cartbutton.vue. This button redirects the user to your orders page when he or she clicks on the button.

Menu button

The menu button redirects the user to the menu page. This button can be found in the header of every page. In our code you can find the button in the folder src → custom-tags → menubutton.vue.

Search bar

The search bar is available in every header. This includes a search bar and a tag filter. The components that are imported are the CategoryFilter and the SearchResults. In the SearchResults, which can be found in src → components → SearchResults.vue, a list of the products that match the input are shown. When a user clicks on a product they are being redirected to the page of that specific product. The CategoryFilter that can be found in the folder src → components → CategoryFilter. Enables the user to add specific categories while you're searching to only find products that have these tags. In the searchbar there are multiple methods to search for every character you put into the input field.

Navigation

On every page except the home page, you can find a hamburger on the top right corner. When you click on this button you open the navigation menu. The navigation page can be found under src → components → Navigation.vue. This is a modal that opens when it is called. It contains buttons to go to the different pages. It also contains buttons to call help and change from table when you selected the wrong one.

Help

The help button that is shown in the footer of every page and in the navigation, page can be found in the folder src → components → HelpModal. When you click the help button it opens modal where you can fill in a form. When the user is done it clicks on the button confirm. When the user clicks on the button confirm the form will be send through a WebSocket to the employee app and they can than see the request of help.

Tests

For the front-end we also made some tests. All of these tests can be found in the folder customerapp-vue → tests\unit. The tests are divided into the components and the views. Each of the test classes contains the most important functions of that page. For the tests we used Vue CLI Babel with the Vue version of Jest. The main library we use for the test is called Vue Test Utils.

Backend

Our backend consists of only one application: A RESTful API written in java using Spring Boot. This application can be found inside our repository in the folder 'back-end'. The API connects with a database using a database-library dependency we wrote. This is because the data we send to the database also needs to be read by the admin and employee application. Thus, we setup a shared database. To make sure every backend is using the same implementation of the database we wrote a library that can be implemented on every backend. This library uses hibernate to handle all database transactions.

Furthermore, the API consists of a three-layered structure.

1. **Controllers**: This is the application-layer. In the layer all of our controllers are listed. Within these controllers are several endpoints that can be used to retrieve or post data.

2. **Services**: This is the business-layer of our project. In this layer we created multiple services that handle all of our business logic. Each service is implemented in the according controller to handle all of the logic that is needed to retrieve or post the data that is requested in the endpoint.

3. **Database-library**: This is the persistence layer of our project. It is more of a dependency than a layer really, but it is still a big part of the application. Using Hibernate it will access the database and retrieve or post data from/to it.

Models

In the database-library we created the base models for our project. Each of these models has its own table within the database and has a repository to retrieve or post data to the database using this model's repository. In short, the models are:

Category: In this model all of the data that is needed for a category is stored, for example a name and image, but it also had a relation to products that are in this category.

Product: In this model all of the data that is needed for a product is stored. For example, the name, price and image. It also has a relation to the ingredients that are in this specific product. Furthermore, it has a relation to the tags linked to the product and the categories this product is in.

Ingredient: In this model all of the data that is needed for an ingredient is stored. This is mainly just the name of the ingredient.

IngredientProduct: This model saves the relation between a product and an ingredient. This model was needed because within this relation we also needed to store a Boolean on if the ingredient was required for a specific product, so that customers won't be able to remove them from their order.

Tag: In this model all of the data that is needed for a Tag is stored. Tags are used to mark certain products with certain tags, for example: 'contains fish' or 'contains gluten'.

RestaurantTable: within this model the data for a restaurant table is stored. Meaning the table number and X/Y axis. With this we know how many tables there are, and to which tables our customers can register to.

RestaurantOrder: This model stores the data for each specific order. It has a subtotal, paid status, a note for more instructions to the kitchen and a created date. Each RestaurantOrder is linked to a RestaurantTable. It also has a link to a list of ProductOrder.

ProductOrder: This model stores the relation between a product and an order. We needed to make this model because we also needed to store an order-status on each specific product in an order. This way we can see for each specific product in an order if it's already delivered or not.

RestaurantInfo: In this model all of the information on the restaurant is stored. For example, the name of the restaurant, the color themes used, the contact information etc.

Data Transfer Objects

Because the database-library contains some recursive loops we use DTO's to transfer data from our backend to our front-end application. For example: A product has a category, and this category also has to product. The original model will now keep retrieving the category and the product and never break the loop. To avoid this problem, we implemented DTO's that can ignore properties within the JSON. For example, we have a ProductDto. Within this product in the category relation the products within the category will be ignored. So, when we retrieve a ProductDto model we can ask in which category it is, but it will ignore all products within that category since that will cause an infinite loop. Using the modelmapper dependency we map our DTO's into entity's and backwards. Each controller that uses Dto's has access to two methods for this process: 'ConvertToDTO' which will take an entity and map it to a DTO, and 'ConvertToEntity' which will map a DTO to an entity.

Services

Because we don't want to put all of our logic in the application layer, we made services for each controller. Each service is implemented in the corresponding controller. For example, the ProductService is injected into the ProductController. These services implement functionality such as retrieving models by ID, name, creation date, etc. To accomplish these functionalities each service implements a repository linked to the model from the database-library.

We also have two controllers that are not connected to a specific model. These are the AdminMockService, and the PDFService.

The AdminMockService is used to fill up the database with mock data since there is no functionality within our application to add products, categories etc. because normally the admin app will provide this data. But while developing we needed to truncate or even drop the database multiple times. Using this service, we don't have to fill up the database manually. This functionality is also not supposed to be shipped into production and is purely written for development testing.

We also made a PDFService, this service is used to create a PDF of the customer's bill using a HTML template that can be found in the resources of this project. The PDF service will manipulate the template and convert it to a PDF which the customer can download.

CI/CD

Workflow

The project is divided in 2 subprojects: Front-end and Back-end. Creating a workflow using GitHub is useful to check your application before pushing it to a greater branch.

Vue

The front-end is build with Vue.js that is executed by Node.js.

The command that needs to perform for building vue is "**npm run build**".

Before that can occur in the workflow it must have installed all its dependencies, Node.js needs to create the vue application with the belonged dependencies.

Therefore the command that must be inserted is: "**npm ci**".

Before performing any of these actions it is needful to setup a virtual node.js system that can preform the tasks in the cloud.

Maven

The back-end is established with Maven from Java. By the usage of maven it is important to use its commands for creating a ci pipeline.

Therefore there needs to be a command executed that is: "**mvn -B package --file pom.xml**".

Before performing that action it is needful to setup a virtual java system that can preform the tasks in the cloud.

Deployment

For production branches there is a possibility to implement the option to automatically create a Docker image. The image can then be published on Docker.io for production usage.

Having those docker images of the front-end and back-end it is useful to implement them to Azure web services. Azure can read the docker images by docker compose. When having that set-up is it possible to allow continues deployment for the production server.

If the services are set-up it is important to use a database server, for our part it was a MySQL-server that contains a username and password that needs to be accessed by the REST-API (back-end).

When the application is build in production mode it must have its production base-URL (This needs to be adjusted in the front-end).

For example:

- Development: http://localhost:8080/
- Production: https://{example}.azurewebsites.net/

Secrets

Utilizing secrets is useful for constructing a workflow or docker compose. You will be wanting to implement secrets so that the code doesn't show your usernames and passwords. GitHub and Azure have options available to create these secrets.

Known issues

- Order of categories should be sorted according to an order given by the admin app. This is not yet implemented by the admin app. So, we haven't implemented it in our app.
- Table number not showing on the navigation menu.
- Table number is not sent to the websocket for the employee app.
- Issue where ingredients do not show on live version on the product page.