Faculty of Automation and Computer Science

Hungrezy

Subject: Data Transmission

2022 – 2023

Link: github.com/NTimea302/Hungrezy

Team Members:

Nemes Raluca

Nagy Timea

Coordonating teacher:

Dr.Ing. Camelia Claudia AVRAM

1. Objectives

The objective of this project is to develop a fully functional client-server application utilizing the Vue.js framework. The application will be connected to a database, allowing various operations such as data selection, insertion, modification, and deletion.

The project entails the development of a food delivery application that offers a diverse range of restaurants and menus to its users. Upon creating an account and logging in, users have the ability to select a desired restaurant, browse through its available food options, and specify their delivery address. The delivery cost is dynamically determined, incorporating both a fixed base fee and an incremental charge per kilometer beyond a predefined distance. The calculation of the delivery cost takes into consideration factors such as the total distance traveled, the standard delivery fee, the standard delivery distance, and the additional cost per kilometer.

2. Architecture of the application

The application comprises two distinct components: the client and server. To initiate the application, follow these steps:

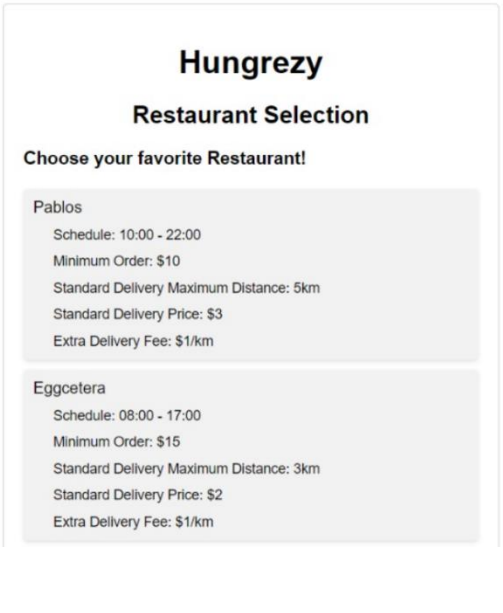
1. Start by downloading the application from the provided GitHub link.
2. Begin by running the server component. Navigate to the "my-server" folder in your command line interface.
3. Execute the command "node server.js" to launch the server. Ensure that you have Node.js installed on your system.
4. Once the server is up and running, proceed to the client component. Open another command line interface and navigate to the "my-project" folder.
5. Type the command "npm run serve" to initiate the client part. Make sure you have npm (Node Package Manager) installed on your machine.
6. The client part will now be running, allowing you to interact with the application.

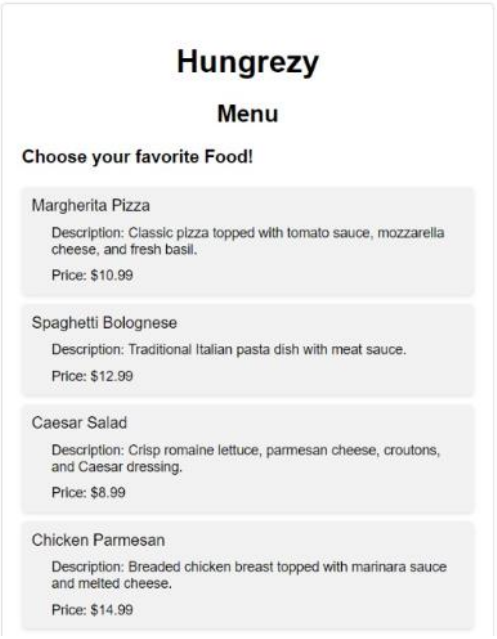
Note: It is imperative to run the server before the client to establish the necessary communication between the two components.

After following the before mentioned steps, the client part of the application will be accessible via your web browser at "localhost:8080". This local address serves as the entry point for interacting with the application's user interface.

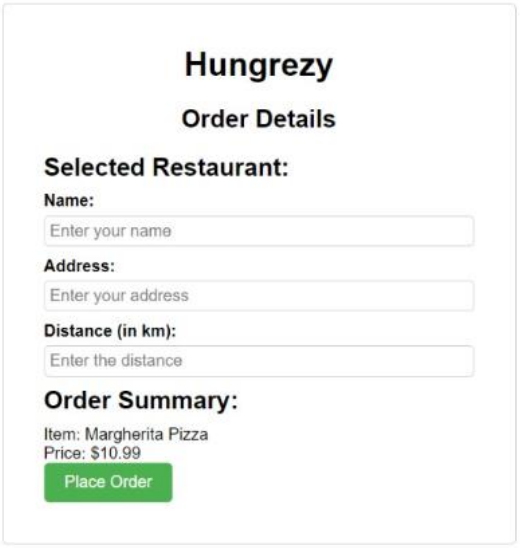
2.1 Client part of application

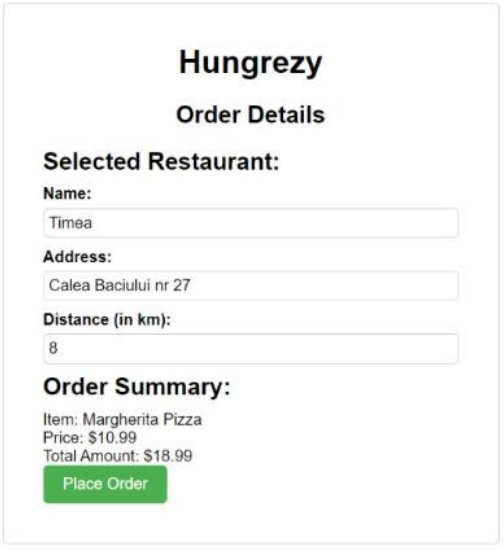
The user interface showcases the following design and layout:





Once a restaurant is selected, the user gains access to the available dishes provided by the chosen establishment. Upon selecting a specific food item, the user is then prompted to enter their personal details in order to proceed with the food ordering process.





Following that, a confirmation page will appear, presenting essential order details such as the client's name, address, total payment amount, the restaurant, the food item's name, and an order code. Additionally, users can proceed to the next page to check the status of their order using the provided confirmation number.

2.2 Server part of application

The server is available in browser on localhost:3000. It is connected to the database, allowing for efficient data retrieval and management. The client application interacts with the server to obtain the necessary information and data. Once the server is running using the specified steps, you can open your web browser and enter either "localhost:3000/restaurants" or "localhost:3000/menu" in the address bar. By doing so, you will be able to access and view the available data from the connected database. This allows you to explore the restaurant information or menu items directly through the browser interface.

2.3 Database

The database for this application is implemented using SSMS (SQL Server Management Studio). It consists of multiple tables, which are visually represented in the accompanying diagram. The diagram provides an overview of the database structure, illustrating the relationships between the tables and their respective attributes. It serves as a helpful reference to understand the organization and connectivity of the database schema.

3. Application Implementation

The application was developed using the Visual Studio Code editor as the primary development environment. To run the application, the command window was utilized to start both the server and the client components. The browser was then used to access the specified local URLs, such as localhost:3000 and localhost:8080, to interact with the application's endpoints and view the user interface. This setup allowed for seamless development, testing, and visualization of the application in a coordinated manner.

3.1 Server

To ensure that the necessary dependencies are available for your project, follow these steps:

1. Install Node.js on your system if you haven't already. You can download it from the official Node.js website (https://nodejs.org).
2. Create a new project directory or navigate to an existing one in your command line interface.
3. Run the following commands to install the required packages:

```
npm install express

npm install cors

npm install msnodesqlv8
```

These commands will download and install the 'express', 'cors', and 'msnodesqlv8' packages from the npm (Node Package Manager) registry.

The following code sets up an Express application, includes CORS middleware to handle cross-origin requests, and adds JSON parsing middleware to process requests with JSON data.

```
5   const app = express();
6   app.use(cors());
7   app.use(express.json());
```

The "localhost:3000/restaurants/id" is an API endpoint that allows users to access specific resources related to a restaurant based on its unique identifier (ID). The given code snippet sets up a route handler for a GET request to '/restaurants/:id' endpoint. It extracts the 'id' parameter from the request URL and executes an SQL query to retrieve restaurant data from a SQL Server database. The retrieved data is then returned as a JSON response. Error handling is included to handle database errors and cases where no restaurant is found.

```
28  app.get('/restaurants/:id', (req, res) => {
29    const id = req.params.id;
30    const query = 'SELECT * FROM Restaurants WHERE Id_restaurant = ?';
31    const connectionString =
32      'server=DESKTOP-I7T6E5Q;Database=Food;Trusted_Connection=Yes;Driver={SQL Server Native Client 11.0};Protocol=TCP;';
33
34    sql.query(connectionString, query, [id], (error, rows) => {
35      if (error) {
36        console.error(error);
37        res.status(500).send('Error retrieving restaurant');
38      } else if (rows.length === 0) {
39        res.status(404).send('Restaurant not found');
40      } else {
41        res.json(rows[0]);
42      }
43    });
44  });
```

The to '/menu/:id' endpoint is implemented according with minimal modifications.

3.2 Client

The client part is responsible for delivering a user-friendly interface, handling user interactions, and communicating with the server to fetch data and perform necessary actions.

It sends *HTTP* (*Hypertext Transfer Protocol* which is an application protocol used for transmitting and receiving data over the internet) requests to the server to fetch data and perform actions, and receives responses to update the interface accordingly.

To ensure that the necessary dependencies are available for your project, follow these steps:

1. Install Node.js on your system if you haven't already. You can download it from the official Node.js website (https://nodejs.org).
2. Create a new project directory or navigate to an existing one in your command line interface.
3. Run the following commands to install the required packages:

npm init

npm install

npm install vue

npm run serve

```
$.ajax({
  url: "http://localhost:3000/restaurants",
  method: "GET",
  dataType: "json",
  success: function(restaurants) {
    restaurants.forEach(function(restaurant) {
      var restaurantElement = $("<div>")
        .addClass("restaurant")
        .append($("<div>").text(restaurant.Name).addClass("restaurant-name"))
        .append($("<div>").text("Schedule: " + restaurant.Start_time.substring(11, 16) + " - " + restaurant.End_time.substring(11, 16)).addClass("restaurant-info"))
        .append($("<div>").text("Minimum Order: $" + restaurant.Minimum_order).addClass("restaurant-info"))
        .append($("<div>").text("Standard Delivery Maximum Distance: " + restaurant.Max_distance_delivery + "km").addClass("restaurant-info"))
        .append($("<div>").text("Standard Delivery Price: $" + restaurant.Delivery_price).addClass("restaurant-info"))
        .append($("<div>").text("Extra Delivery Fee: $" + restaurant.Extra_delivery + "/km").addClass("restaurant-info"))
        .click(function() {
          localStorage.setItem("selectedRestaurant", JSON.stringify(restaurant));

          // Extract the restaurant ID
          var idRestaurant = restaurant.Id_restaurant;

          // Redirect to the menu.html page with the selected restaurant ID as a URL parameter
          window.location.href = "menu.html?idRestaurant=" + idRestaurant;
        });
      restaurantList.append(restaurantElement);
    });
  },
  error: function(xhr, status, error) {
    console.error(error);
  }
});
```

The provided code snippet demonstrates the usage of AJAX (Asynchronous JavaScript and XML) to retrieve restaurant data from the server without refreshing the page. The code makes an HTTP GET request to "http://localhost:3000/restaurants" and expects a JSON response. Upon success, it creates

 elements for each restaurant, populating them with information. AJAX allows for seamless data retrieval and dynamic content updates in web applications without disrupting the user experience.

```
103   // Generate a random order code
104   function generateOrderCode() {
105     var characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
106     var codeLength = 8;
107     var orderCode = "";
108     for (var i = 0; i < codeLength; i++) {
109       var randomIndex = Math.floor(Math.random() * characters.length);
110       orderCode += characters.charAt(randomIndex);
111     }
112     return orderCode;
113   }
```

The above provided code snippet demonstrates the generation of order codes. It defines a function called **generateOrderCode()** that uses a set of characters and a specified code length. Inside a loop, a random character is selected from the available characters using **Math.random()** and appended to the **orderCode** string. After looping for the desired code length, the generated order code is returned. This function allows for the creation of unique and randomized order codes.

```
78    $("#distance-input").on("input", function () {
79      var distanceInput = parseFloat($(this).val());
80      var selectedRestaurant = JSON.parse(localStorage.getItem("selectedRestaurant"));
81      var extraDeliveryFee = selectedRestaurant.Extra_delivery;
82      var extraFee = extraDeliveryFee * distanceInput;
83      var totalAmount = selectedMenuItem.Price_food + extraFee;
84
85      var totalAmountElement = $("#total-amount");
86      totalAmountElement.text("Total Amount: $" + totalAmount.toFixed(2));
87      totalAmountElement.show(); // Show the total amount element
88    });
```

The above code snippet demonstrates the calculation of the distance fee based on the restaurants extra delivery fee, standard delivery, standard delivery distance and the total distance from the restaurant's location to the customers address.

5. Conclusion

In conclusion, the development of the Hungrezy food delivery application has been has been a significant achievement for us. This project aimed to create a user-friendly platform using Vue.js, enabling users to explore various restaurants, select dishes, and place orders. The application's architecture follows a client-server model.