IŞIK UNIVERSITY
COMPUTER
SCIENCE AND
ENGINEERING

FeastFive

Bachelor's Thesis

Ahmet Aksu 20SOFT1042

Zehra Uğurlu 19SOFT1031

Eren Kaya 19SOFT1096

Supervised by

Tuğba Erkoç

January 2024

ABSTRACT

Many people struggle to find the meals and quality eating experiences they want. On the other hand, restaurant owners frequently find their businesses becoming buried in the broad internet world, pushed aside by algorithms. This leads to a huge problem because users are unable to find the diversity and quality they seek, and restaurants are unable to effectively reach their target audience.

The FeastFive website will provide a smooth and user-friendly experience for users to browse and buy or sell meals. With a large selection of meals and affordable prices, our website will be easy to use and explore. It will be capable of accurately responding to user queries, and it should provide feedback to assist users as needed. It will be able to respond accurately to people's requests and there should be feedback to guide people where necessary. Also, the site will have a user-friendly interface that protects personal information so that customers can buy safely from this site. The system guarantees safe payments, data security, and defense against online threats. We wanted to create a platform where individuals can discover new meals, order food, and provide feedback,comments on their orders. At the same time, in this initiative, where restaurant owners can also earn sales, we hope to bring together restaurant owners and users in that region while also empowering restaurant owners by giving them a platform to enhance sales and connect. Also feastfive is providing the easiest user experience for restaurant owners to build an online shop and get noticed by people. We developed a platform that is simple and easy to use. Users may navigate across many functions with ease thanks to our intuitive panel. We provide personal dining experiences based on individual taste preference, ensuring that every type of restaurant is noticed and connected with restaurants target audience.

We used MongoDB as our database to store data JSON documents, which allows for flexible scalability. The server-side API(Application Programming Interface) is built with Node.js, which allows for great performance. We use the Stripe API to process payments, which is secure and reliable. On the front end, we present our project using React, which has a component-based structure and provides a dynamic and component based user experience. This technological mix provides a reliable, scalable, and efficient system for data administration, payment processing, and user engagement.

ACKNOWLEDGEMENTS

- Special thanks to Tuğba Erkoç for her support and guidance
- Thanks for MS Office 365 for providing this document free to use for students
- Thanks for Node.js Library to provide area for create API`s and also React Framework to present project with dynamic credentials

Table of Contents

LIST OF TABLES

1. INTRODUCTION

1.1. Problem Definition

Many people struggle to find the meals and quality eating experiences they desire, while restaurant owners often find their businesses getting lost in the vast internet world, overshadowed by algorithms. This results in a significant problem: users are unable to find the diversity and quality they seek, and restaurants cannot effectively reach their target audience.

The FeastFive project aims to address these issues by establishing a comprehensive online platform that meets the needs of both users and restaurant operators, providing the best possible food purchase experience. Our platform makes it easy for clients to access the foods they want, discover new meals, order food, and provide feedback on their orders. Simultaneously, FeastFive offers restaurant owners a platform to increase sales and connect with users in their region.

FeastFive provides restaurant owners with the simplest user experience for building an online shop and getting noticed by people. Our platform is designed to be simple and easy to use, with an intuitive panel that allows users to navigate various functions effortlessly. We offer personalized dining experiences based on individual taste preferences, ensuring that every type of restaurant is discovered and connected with its target audience.

1.2. Definitions, Acronyms, and Abbreviations

- **HTTP (Hypertext Transfer Protocol):** A protocol used for transmitting data over the internet, allowing web browsers and servers to communicate.
- **JS (JavaScript):** A programming language commonly used to create interactive effects within web browsers.
- **JSON (JavaScript Object Notation):** A lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
- **API (Application Programming Interface):** A set of rules and tools for building software applications, allowing different software programs to communicate with each other.
- **UI (User Interface):** The space where interactions between humans and machines occur, focusing on the design and layout of an application.
- **UX (User Experience):** The overall experience a user has when interacting with a product or service, emphasizing ease of use and satisfaction.
- **GUI (Graphical User Interface):** A type of user interface that allows users to interact with electronic devices using graphical icons and visual indicators.
- **HTML (Hypertext Markup Language):** The standard markup language used to create and design pages on the web.
- **CSS (Cascading Style Sheets):** A stylesheet language used to describe the presentation of a document written in HTML or XML, including colors, layout, and fonts.
- **ReactJS:** A JavaScript library for building user interfaces, particularly for single-page applications, by creating reusable UI components.
- **NodeJS:** A JavaScript runtime built on Chrome's V8 JavaScript engine, allowing the development of server-side and networking applications.
- **MongoDB:** A NoSQL database program that uses JSON-like documents with optional schemas to store data, known for its scalability and flexibility.
- **ID (Identifier):** A unique value used to identify a particular object, record, or element within a system.
- **CRUD (Create, Read, Update, Delete):** The four basic operations of persistent storage in databases and data manipulation.

1.3. Organization of the Thesis

The first part of the thesis which is “Introduction” simply gives general informations about the project and informs the reader about what they should expect from the project. So it include some shallow information and definitions for reader who do not have deep technical knowledge. Also include explanations of the terms and abbreviations which are used in this thesis. Second chapter which is “Literature Review” is edited to inform and gain objective point of view the reader about what are the similar successful projects did and what our project did some certain topics better or worse. Readers can also gain a superficial understanding of how we performed better in this project compared to others. In the third part of the thesis which is “Proposed System” readers have informed about Architecture of the project. Also the building blocks of the architecture were presented to the reader, including information about the project's data and file structures. The backend functions`s were explained to the reader who has technical knowledge about development process. Graphical User Interface has introduced to reader as image and explained what are the design criterias and how we achieved them with technical knowledge. Services , which the project can perform , and Entities , which contains the database ,has also been introduced. This sections are also include intensive technical terms so it is for readers who have high technical knowledge. The forth parth which is “Implementations , Tests , Experiments” explains the how the project implemented and tested in the development and the test part of the project. Technologies which are used for develop this project has introduced and explained what they has used for. Every test scenario and every test case of the scenario for the scenario has recorded with their results. In the fifth part which is “Conclusion and Future Work” as Feastfive Team what are the main goals and success criterias about this project. Also we explained what are the problems we had faced and how we solved them. Ideas about how the project could be expanded has been given. The sixth and final path is “References”. In this part, the sources from which the information shared throughout the thesis and during the project were obtained are listed with citations.

2. LITERATURE REVIEW

2.1. Similar Applications

2.1.1. YemekSepeti

Yemeksepeti is a leading online food delivery service operating primarily in Turkey. Established in 2001, it revolutionized the way people order food by providing a convenient, online platform that connects users with a wide range of restaurants and cuisines. YemekSepeti has the greatest number of member restaurants.

2.1.2. Getir

Getir Yemek is an online meal delivery service operated by Getir, a pioneering Turkish rapid delivery company. Getir Yemek, which debuted in 2019, expands Getir's fast delivery concept into the food industry, enabling quick and convenient access to meals from a variety of eateries. Getir provides exact location of courier and present it in simple UI

2.1.3. Tıkla Gelsin

Tıkla Gelsin is an online food ordering and delivery platform in Turkey. It connects users to a variety of restaurants, allowing them to easily buy meals online and have it delivered to their house or take away.

2.1.4. Comparison of Similar Applications

Below, Table 2.1, you can find a summary and comparison of existing applications. (Please note that, it contains hypothetical data, has been filled randomly.)

Table 2.1. Comparison of meal ordering applications

	Yemeksepeti	Getir	TıklaGelsin	Feastfive
OfferFood based on previous offers				✓
Offer food	✓	✓	✓	✓
Support for local neighborhood		✓	✓	✓
Live courier location		✓		
Take away	✓			

2.2. User Based Suggestion Algorithms

Our platform uses a smart homepage algorithm to improve users' experience. It looks at your past orders, the restaurants and dishes users browse, and users favorite types of food. Using this information, it shows you the restaurants and meals users are most likely to enjoy. This makes it easier and faster for users to find what they want, and helps restaurants connect with the right customers.

2.2.1. Cookie for user experience

In our project, cookies are used to improve your experience. Cookies temporarily store information about your orders, browsing preferences, and favorite cuisines. Cookies maintain this information, allowing our homepage system to remember your preferences and display you related restaurants and meals more quickly. This speeds up and personalizes the food ordering experience, ensuring that you always get what you're looking for with least effort. Furthermore, this allows restaurants to contact clients who are more likely to present their products.

3. PROPOSED SYSTEM

3.1. Introduction

The proposed system is a web-based application. While it allows customers to access and order the food they want, it also allows restaurant owners to create online restaurant pages and sell food and menus online. It also improves the experience of restaurant owners in presenting statistical data and graphical tables with them. In addition, all users of the site have an algorithmic front-end thanks to the Cookie provided by our site. The FeastFive website has an MVC architectural pattern. MVC architectural pattern includes Model, View and Controller, each responsible for a specific part of the application's operation.

3.1.1. Model

The Model component corresponds to any data-related logic that the user interacts with. This can refer to either the data being transmitted between the View and Controller components or any other business logic-related data. This component of the proposed system is built on a customized NoSQL service called MongoDB. There are 3 main tables in this system: restaurants, users and kitchens.

- **User Model:**
    - **_id:** MongoDB provides a user-specific credential.
    - **name:** It represents the user's name. This field is required and an error will be raised if the name is not provided.
    - **surname:** It represents the user's surname. This field is required and an error will be raised if the surname is not provided.
    - **email:** It represents the user's email. This field is mandatory and unique and if the email is not provided or the same email is found in the db, an error will occur.
    - **password:** It represents the user's password. This field is required and an error will be raised if the password is not provided.
    - **registerDate:** It represents the register date of the user.
    - **loginDate:** It represents the user's last login date. This field is updated every time the user logs in.

- **uniqueId:** Unlike the _id provided by mongodb, it is created by the function we provide and is unique. It is actively used in transactions such as sending emails and making payments.
- **role:** It represents the user's role. This field can take either the value "user" or "admin" and is set to "user" by default.
- **comments:** It represents an array containing comments made by the user.
- **orders:** It represents an array containing orders made by the user.
- **address:** It represents an array containing user-entered addresses.
- **activated:** It represents the field indicating whether the user's account's email is approved or not. By default it is set to false.

- **Restaurant Model:**
  - **_id:** MongoDB provides a user-specific credential.
  - **restaurantName:** It represents the name of the restaurant. This field is required and an error will occur if the name is not entered.
  - **ownerName:** It represents the name of the owner of the restaurant. This field is required and an error will occur if the name is not entered.
  - **ownerSurname:** It represents the surname of the owner of the restaurant. This field is required and an error will occur if the surname is not entered.
  - **email:** It represents the email of the restaurant. This field is mandatory and unique and if the email is not provided or the same email is found in the db, an error will occur.
  - **password:** It represents the password of the restaurant. This field is required and an error will be raised if the password is not provided.
  - **registerDate:** It represents the register date of the restaurant.
  - **loginDate:** It represents the last login date of the restaurant. This field is updated every time the restaurant logs in.
  - **uniqueId:** Unlike the _id provided by mongodb, it is created by the function we provide and is unique. It is actively used in transactions such as sending emails and making payments.
  - **role:** It represents the role of the restaurant. This field takes "restaurant" and is set to "restaurant" by default.
  - **address:** It represents an object containing an owner-entered address.
  - **image:** It represents the header image of the restaurant.

- **meals:** It represents an array containing the meals on the restaurant's menu. By default it is set to an empty array.
- **orders:** It represents an array containing the orders of the restaurant. By default it is set to an empty array.
- **labels:** An array field containing the restaurant's labels. By default it is set to an empty array.
- **comments:** This is an array containing comments made by the customers of the restaurant.
- **activated:** It represents the field indicating whether the restaurant's account's email is approved or not. By default it is set to false.

- **Kitchen Model**
  - **_id:** MongoDB provides a user-specific credential.
  - **name:** It represents the cuisine's name.
  - **image:** It represents the cuisine's image.

3.1.2. View

The View is responsible for interacting with users through functions such as providing information and receiving feedback from them. GUI elements are included to be as simple and easy to understand as possible. It provides the user with data from the Model and transmits user inputs to the Controller. Also, it is passive and does not directly interact with the Model.

The React framework is used to represent the View model's components, and there is a representation for every page. The elements are:

- **PurchaseAccepted.js:** After a successful payment, the user is directed to this page and the user is shown that the payment has been made.
- **PurchaseRejected.js:** After a failed payment, the user is redirected to this page and the user is shown that the payment has not been processed.

- **CartPage.js:** This page allows the user to view and manage their shopping cart. After this page, you will be directed to the checkout page provided by Stripe API for payment.
- **Cart.js:** This side page allows the user to view and manage their shopping cart. Unlike the other cart page, it appears as a side window when the cart button is clicked.
- **ChangePassword.js:** This page allows the user to change their password.
- **RestaurantChangePassword.js:** This page allows the restaurant owner to change their password.
- **Favoritie.js:** Restaurants that the user likes are displayed on this page.
- **ForgotPasswordPage.js:** If the user forgets her/his password, user is directed to this page and asked to enter her/his e-mail address.
- **RestaurantForgotPassword.js:** If the restaurant owner forgets her/his password, the user is directed to this page and asked to enter her/his e-mail address.
- **HomePage.js:** On this page, traditional cuisines and restaurants are shown to the user. Also on this page the user can search for restaurants and cuisines.
- **FilterCuisine.js:** If a user wants to see restaurants of a particular cuisine, they are directed to this page. On this page, only restaurants of the type requested by the user are listed.
- **FoodHome.js:** On this page, the user can view all the restaurants and filter the restaurants as he/she wishes with the filtering component on the left of the page.
- **LoginPage.js:** This page has been created for the registered user to log in to the system.
- **RestaurantLogin.js:** This page has been created for the registered restaurant to log in to the system.
- **NoPage.js:** If the user tries to go to a path that does not exist or does not have permission, this page is displayed.
- **GivenOrder.js:** When the restaurant and the user want to see their active and passive orders and status, they are directed to this page. On this page the restaurant can cancel or confirm the order. The user can comment and vote on his/her finished order.
- **RestaurantProfile.js:** This is the page where restaurant information is displayed, information can be edited and addresses can be added. On this page, past orders are

displayed, the restaurant menu and label can be edited, and the account can be deleted.

- **UserProfile.js:** This is the page where user information is displayed, information can be edited and addresses can be added. On this page, past orders are displayed, and the account can be deleted.
- **SignUpPage.js:** This page has been created for users who want to register to the system. It takes some information from the user and allows them to register to the system.
- **RestaurantSignUp.js:** This page has been created for restaurant owners who want to register to the system. It takes some information from the restaurant owners and allows them to register to the system.
- **Menu.js:** On this page, the restaurant owner can view, edit, add and delete menus.
- **UpdateMenu.js:** When the restaurant owner wants to edit her/his menu, the user is directed to this page and updates the menu to be edited on this page.
- **RestaurantFoods.js:** When the user clicks on a specific restaurant on this page, the menus, comments and ratings of that restaurant are displayed.
- **RestaurantPanelPage.js:** This page is the home page of the restaurant. On this page, certain statistical data and charts about the restaurant are shared with the restaurant user.

We also have many components that support these pages:

- **Navbar.js:** It has the site title and is found at the top of each page. The symbols for the user profile and logout are also included.
- **Footer.js:** Located at the bottom of these pages.
- **ShowAlert.js:** Pop-up component that appears for information and warning purposes during the user's actions. We used “react-toastify” for this component.
- **SearchBar.js:** The component on the home page that allows the user to search.
- **Loader.js:** The loading component we use to show the user that data is being loaded while fetching data.
- **OrderRestaurant.js:** On the given order page, if the user role is restaurant, it is the orders component displayed.

- **OrderUser.js:** On the given order page, if the user role is user, it is the orders component displayed.
- **RestaurantPanelMenu.js:** It shows the menu items of a restaurant and is used to edit or delete these items in the administration panel.

3.1.3. Controller

Controllers conduct all business logic and incoming requests, use the Model component to manipulate data, and communicate with the Views to render the final product by acting as an interface between them. When it receives input from the user, it modifies the Model appropriately and updates the View to show the changes in the Model. It includes data transformation and input validation, among other application logic.

- **User Controller**
    - **registerUser:** This function is called if the user is not registered. When users want to register on the FeastFive site, they enter their information on the web page and this function adds that user to the database. If the process of adding to the database is successful, a verification email is sent to the user's email via nodemailer. Password information is encrypted with bcrypt.
    - **loginUser:** This function is called if the user is registered. When users want to log in to the FeastFive site, they enter their information on the web page and this function queries that user in the database. If the email and password match and the user's email is verified, it returns the user's information. Password information is encrypted with bcrypt.
    - **editUser:** This function is used to update user information. If the user exists in the database, then his/her details are replaced with the new data and saved. It will return an error if the user is not found.
    - **deleteUser:** This function is used to delete user information. If the user exists in the database, deletes and saves the user. If the user cannot be found, it gives an error.
    - **forgotPassUser:** This function sends a recovered password e-mail to the user's email, if the user email is available in the database.
    - **forgotPassRedirect:** This function is used to handle redirection for password reset requests. This function locates the associated user, validates the unique ID, and sends the user to the password change page when they click on a

password reset link that contains a unique ID. It returns the appropriate error response if an error happens or it cannot find the user.

- **changePassword:** This function requests the new password from the user and encrypts it with bcrypt and processes it into the database.If the password is sent empty, an error is returned.
- **activateAccount:** When the user clicks on the link in the email sent during registration, this function is triggered and detects the user in the database and makes the activated value correct.
- **getOrders:** This function returns users' orders. It lists all the orders.
- **addAddress:** This function is used to add a new address to the user's database data.
- **getAddress:** This function returns users' addresses. It lists all the addresses.

- **Restaurant Controller**
  - **registerRestaurant:** This function is called if the restaurant is not registered. When restaurant owners want to register on the FeastFive website, they enter their information on the web page and this function adds that restaurant to the database. If the addition to the database is successful, a verification email is sent to the restaurant’s email via nodemailer. Password information is encrypted with bcrypt.
  - **loginRestaurant:** This function is called if the restaurant is registered. When a restaurant owner wants to log in to the FeastFive site, they enter their information on the web page and this function queries that restaurant in the database. If the email and password match and the restaurant's email is verified, it returns the restaurant's information. Password information is encrypted with bcrypt.
  - **activateRestaurantAccount:** When the restaurant owner clicks on the link in the email sent during registration, this function is triggered and detects the restaurant in the database and makes the activated value correct.
  - **getRestaurant:** This function returns restaurants. Lists all restaurants in the database.

- **editRestaurant:** This function is used to update restaurant information. If the restaurant exists in the database, then its details are replaced with the new data and saved. It will return an error if the restaurant is not found.
- **deleteRestaurant:** This function is used to delete restaurant information. If the restaurant exists in the database, deletes and saves the restaurant. If the restaurant cannot be found, it gives an error.
- **getSpecificRestaurant:** This function returns the specific restaurant requested by the user. Thanks to this function, when the user clicks on or searches for a restaurant, he is directed to the restaurant's page.
- **getOrders:** This function returns restaurant orders. It lists all the orders.
- **getSearchRestaurants:** This function allows users to look for restaurants using a certain key, examining both restaurant names and labels for matches. It returns the matching restaurants and handles potential errors.
- **updateLabel:** This function is used to update the label of the restaurant. If the restaurant exists in the database, then its details are replaced with the new data and saved. It will return an error if the restaurant is not found.
- **forgotPassRestaurant:** This function sends a recovered password email to the restaurant's email, if the restaurant email is available in the database.
- **forgotPassRedirect:** This function is used to handle redirection for password reset requests. This function locates the associated restaurant, validates the unique ID, and sends the restaurant to the password change page when they click on a password reset link that contains a unique ID. It returns the appropriate error response if an error happens or it cannot find the restaurant.
- **changePassword:** This function requests the new password from the restaurant and encrypts it with bcrypt and processes it into the database.If the password is sent empty, an error is returned.

- **Mailer Controller**
  - **sendMail:** This function is designed to send an email using the nodemailer package. It confirms the recipient's address, configures the SMTP transporter with the required credentials, and transmits the email.
  - **sendActivationEmail:** This function is called when the user registers successfully and sends an activation link to the user's e-mail.

- **sendActivationRestaurantEmail:** This function is called when the restaurant registers successfully and sends an activation link to the restaurant's e-mail.
- **forgotPasswordEmail:** This function is called when the user forgets her/his password and sends a recover password link to the user's e-mail.
- **forgotPasswordRestaurantEmail:** This function is called when the restaurant forgets its password and sends a recover password link to the restaurant’s e-mail.

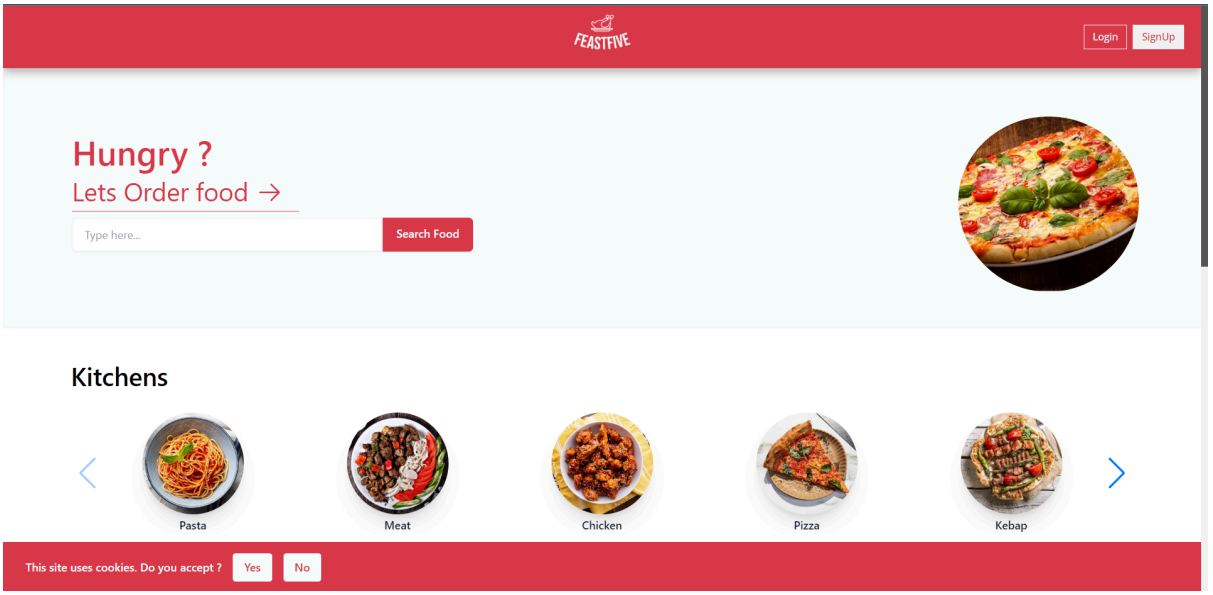
- **Purchase Controller**
  - **addOrder:** This function is used to create a new order in the system, associate it with both a restaurant and a user, and save the updated records in the database.

- **Meal Controller**
  - **getKitchens:** This function returns kitchens. Lists all kitchens in the database.
  - **updateMeal:** This function is used to update the restaurant's meal. If it exists in the restaurant database, it finds the meal in the meals array, replaces it with new data and saves it. If the restaurant is not found, it will give an error.
  - **deleteMeal:** This function is used to delete the restaurant's meal. If the restaurant exists in the database, it finds the meal in the meals array and deletes it.
  - **addMeal:** This function is used to add new meals to the restaurant. If the restaurant is available in the database, the new menu is added to the dishes array.
  - **checkComment:** This function checks the user's order and allows the user to comment if he/she has an order at that restaurant.
  - **addComment:** This function examines the existence of the order using checkComment and if this order exists, it adds the user's new comment to the comment array on the restaurant side and the user side.
  - **doneOrder:** This function is used to mark an order as "Done" and update the status of the order in both the restaurant and user records.

- **rejectOrder:** This function is used to mark an order as "Rejected" and update the status of the order in both the restaurant and user records.

- **create-checkout-session**
  - This function defines routes for creating a Stripe checkout session and processing an order after a successful payment. The POST route /create-checkout-session creates a Stripe payment session and redirects to /purchase. Accepted on success, which then calls addOrder to update the database with the new order information.

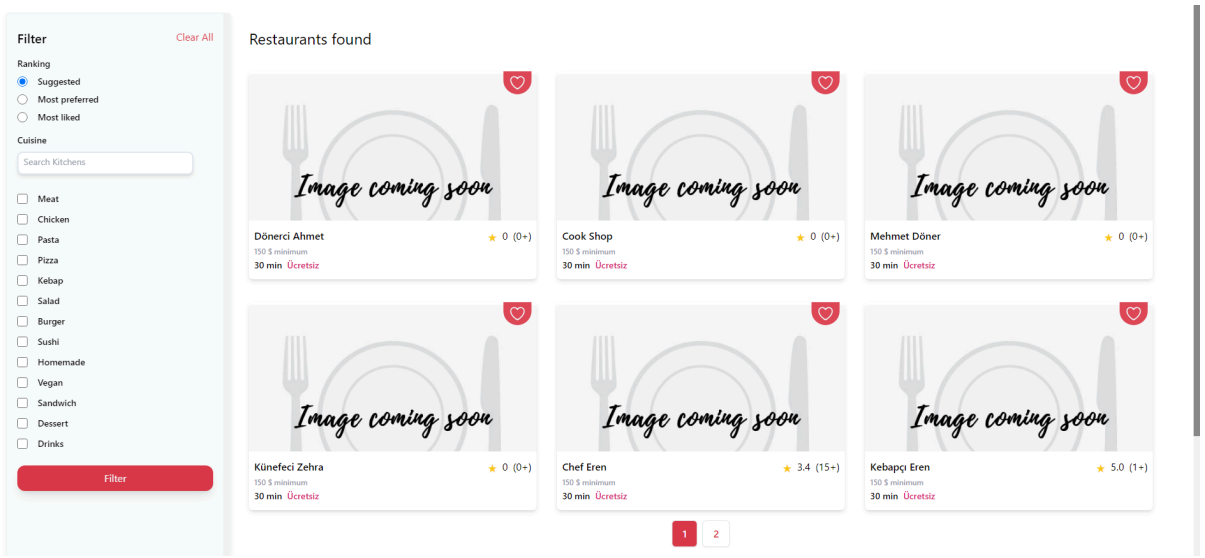
3.2. Graphical User Interface



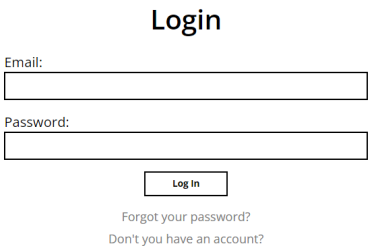
In our project we utilized ReactJS for frontend development and modular CSS along with Tailwind CSS for styling. This combination allowed us to create a user-friendly and efficient graphical user interface (GUI).

In our design principles we focused on user-focused accessibility performance optimization and error management. By adhering to these principles and utilizing selected technologies we ensured that our project provides users with a flawless and enjoyable experience.

Our user-centered design principle ensured that the interface matched the needs and preferences of our target audience. For example we used cookies with the user's consent to help them easily find the most suitable restaurants. Additionally, users can filter and search for specific types of food to quickly find what they are looking for. We also took accessibility aspects into account to ensure that the app can be effectively accessed by users of any ability level.

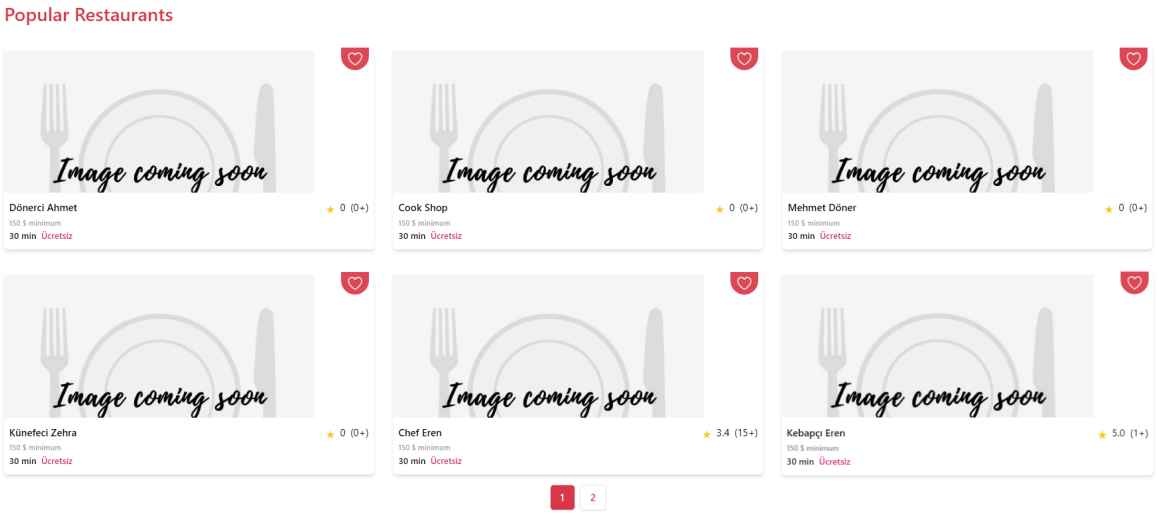


Users can register and log in to our system in two different roles. Users can easily update their information and reset their passwords via email when they forget their passwords. Additionally they must activate their account via email when registering.

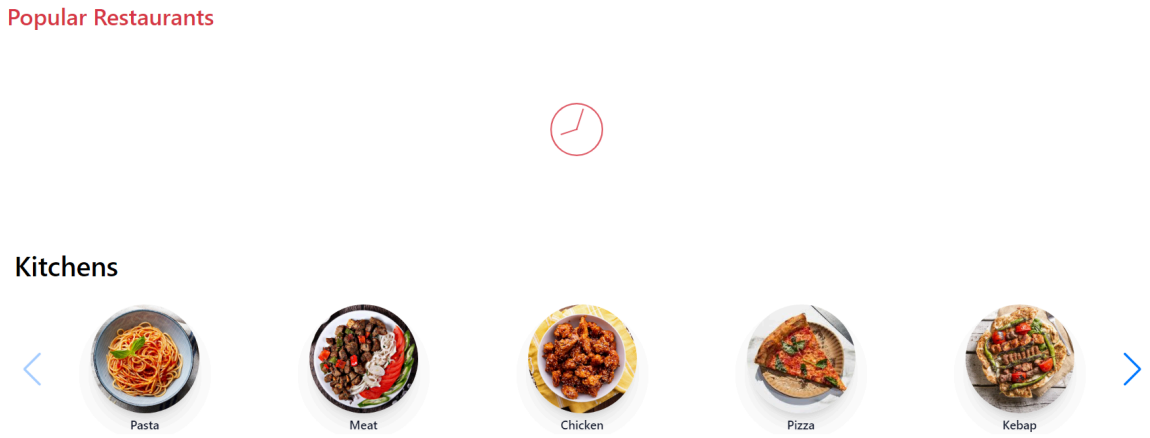


We have defined the pages that users and restaurant owners can access. For example the user cannot access the order screen or any page other than the restaurant owner panel without logging in.

In the performance optimization part we chose to list all restaurants using a pagination system rather than a single list to reduce loading times and increase the overall response speed. Instead of listing all the restaurants we designed it to list a maximum of six restaurants per page.



For additional functionality we used several npm modules such as slider and loader components which enhanced the user experience by providing smooth transitions and visual feedback during loading times.



In summary by focusing on user needs and interface optimization along with the strategic use of ReactJS modular CSS and Tailwind CSS we have created a food delivery application with a user-friendly GUI that offers a seamless experience to our users.

3.3. Business Logic

Stripe API is used for payment and jwt is created for each user type. We provide the user with a more dynamic and algorithmic experience thanks to Cookies. We do not allow Cookies without the user's permission and we do not process information without the user's permission. SSL certificate will be used as it is very secure against attacks. It is important that the system is also incredibly scalable as requests come in.

3.3.1. Login Services

When the user first interacts with the system, the user can freely browse certain pages, but the user is required to register in the system in order to purchase. If the user tries to purchase without registering or logging in, the user is warned with a pop-up page. While registering, the user enters certain valid information into the system and if the email he entered is not in the database, a new user is created in the system by performing the create function. While performing these operations, the password is encrypted using the bcrypt package and stored in the database. When the user is first created, the user is not seen as an active user, but is

activated by clicking on the link sent to his/her email. This link directs the user to the login page. When the user logs in, it is first checked whether the email is in the database. If there is an email in the database, the matching password is checked. The encrypted versions of the password are compared and the password is not decoded during all these processes. Finally, the user's activity is checked and if the user is active, he/she is successfully logged in to the system.

- **registerUser**
- **loginUser**
- **forgotPassUser**
- **changePassword**
- **activateAccount**
- **forgotPassRedirect**
- **registerRestaurant**
- **loginRestaurant**
- **activateRestaurantAccount**
- **forgotPassRestaurant**
- **forgotPassRedirect**
- **changePassword**
- **sendMail**
- **sendActivationEmail**
- **sendActivationRestaurantEmail**
- **forgotPasswordEmail**
- **forgotPasswordRestaurantEmail**

3.3.2. Order Services

If the user is logged in to the system, the user should be able to shop. The user can view the restaurants recommended to him/her on the home page, view various cuisines, and also view the filtered version of the restaurants. The user enters the restaurant of his/her choice and adds the menus the user wants to eat to his/her cart. The system gives the user a single rule: meals can be added to the cart from only one restaurant. The user is expected to complete that order in order to shop from another restaurant. Cart operations are done on the front end and the cart is stored in the redux store and addition and removal operations are

performed by redux store functions. The user can view the status of his orders on the orders page, but cannot cancel his order.

- **getOrders**

On the restaurant side, orders are managed in a different way. The paid orders are reflected on the restaurant side and if the restaurant does not accept the order, the 'rejected' button must be pressed. If the restaurant completes and sends the order, the 'Accepted' button should be clicked. The status of the order is reflected to the user dynamically.

- **doneOrder**
- **rejectOrder**

3.3.3. Purchase Services

Stripe API was used for the purchasing process in the project. After the user adds the desired meals to his/her cart, the user is directed to the order page and the last addition and removal operations are carried out on the order page. When the ‘Apply Order’ button is clicked, the user is directed to the ‘checkout’ page provided by Stripe API. The products in the user's cart, their prices and quantity are transmitted to this API in JSON format. The user enters his/her card information on the page provided by this API and if the payment is made, the user is directed to the successful page, if not, the user is directed to the rejected page. At the same time, requests from this API are met with the function in the GET route and if a positive response is received, this order is processed into the user database and restaurant database.

- **create-checkout-session**
- **addOrder**

3.3.4. Comment Services

The user can only comment and rate the restaurant from which the user has previously ordered. There is only one comment and rating allowed for each order. These comments are reflected on the restaurant's page and dynamically affect its score.

- **addComment**
- **checkComment**

3.3.5. Profile Services

A page where the user can view and edit his/her own information is provided by the system. On this page, the user can add more than one address to the system and select the address they want to use. He can also update his information and delete his account.

- **editUser**
- **deleteUser**
- **addAddress**
- **getAddress**

A similar information updating service is also offered on the restaurant side. But only one address is allowed to enter the restaurant. At the same time, the restaurant account can be deleted.

- **editRestaurant**
- **deleteRestaurant**

3.3.6. Meal Services

On the restaurant side, the label of the restaurant can be updated and labels can be added or deleted. Labels are recommended by the system. (For example, if the restaurant sells pasta, it should add pasta to the label.) When the restaurant owner wants to add a new dish to his restaurant, he enters the menu information from the menu add option and the menu is dynamically added as an object to the restaurant's menu. Additionally, menus can be edited and deleted. In short, the restaurant can easily perform CRUD operations.

- **updateMeal**
- **addMeal**
- **deleteMeal**
- **updateLabel**

3.3.7. Search Services

When the user wants to search, he enters a word the user wants into the search bar on the home page. There is a block in the search bar to prevent requests for each letter entered. Based on the input, a design pattern is created on the back side, and a search is performed based on the restaurant name and restaurant label. As a result, search by restaurant name and search by label are listed on the front page, but they are separated by design to make it easier for the user to understand.

- **getSearchRestaurants**

3.3.8. Address Services

Users and restaurants can add addresses from their profile section. The user must select an address among the addresses he has entered more than once and the order must be placed accordingly. The system lists restaurants according to the user address, and if the restaurant is far from the user, this restaurant is not shown to the user.

- **getAddress**

3.3.9. Filter Services

The user can filter the type of restaurants the user wants. Restaurants can be filtered by cuisine, recommended, most ordered, or highest rated.

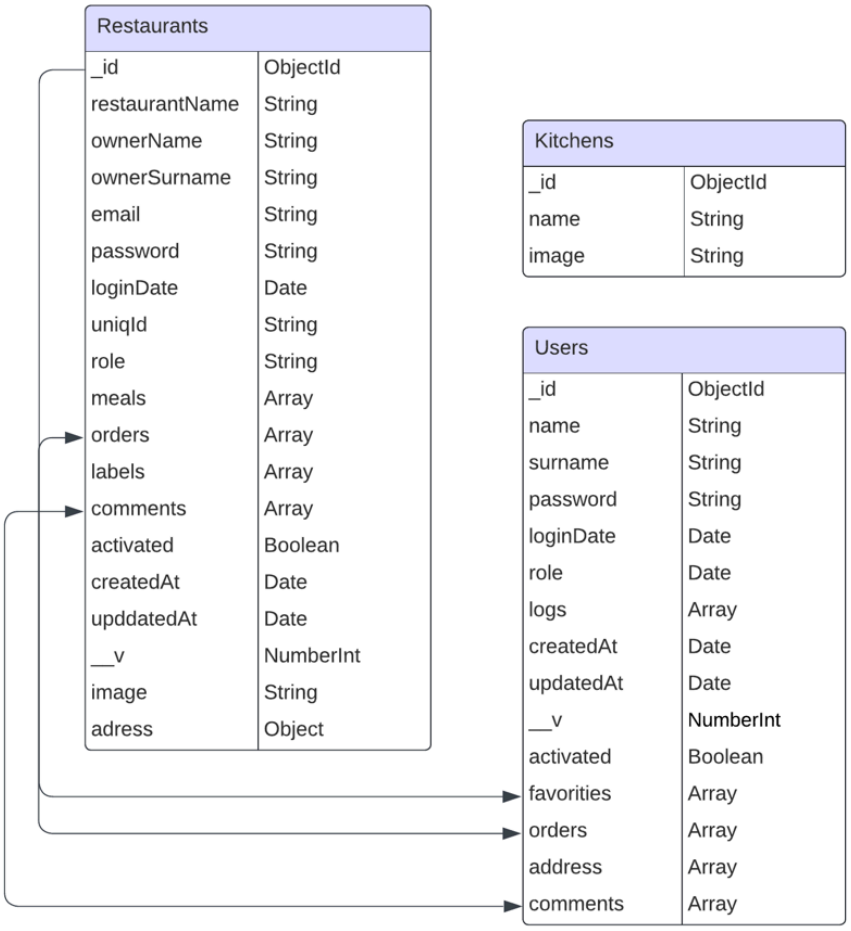
- **getRestaurants**

3.3.10. Meal Suggestion Services

We have developed a food recommendation system in our Feastfive food ordering application to provide a better experience for users. Our priority in this system was to record the labels of the restaurants clicked by the users along with the number of clicks on the cookie side, in line with the user's consent. For example, when a customer who previously clicked on a restaurant in the Chicken category clicked on a restaurant in the Chicken category again, instead of adding Chicken to the cookie side, we only increased the number of clicks on the Chicken category. We developed our project using reactjs and integrated the js-cookie library into our project for cookie management. In order to create a json object list on the cookie side, we saved the list with the JSON.stringify() method, which is a JSON method, and the javascript object as a JSON array. When we want to use this list, we use JSON. We made .parse() usable by converting the json data type into a javascript object. Before even listing the restaurants, we subjected them to the main filtering process. We ranked the restaurants by adding up the labels of each restaurant with the label numbers we obtained from the cookie. In our system, we have now prioritized the restaurants that users are more interested in.

3.4. Data Management

In our FeastFive food delivery project we used NoSQL MongoDB database and explanation of each entity.



Users Entity:

```
_id: ObjectId('661ad1f5731a92f7a104a159')
name : "zehrass"
surname : "ugugugs"
email : "zeugsurlu@outlook.com"
password : "$2a$10$x3NXWlXItgPwwtXPT4FK3.iPZ8iP/Tj7gvFfjCD0gVhxrqSl7zVGO"
loginDate : 2024-04-13T18:41:57.439+00:00
role : "user"
▸ logs : Array (empty)
createdAt : 2024-04-13T18:41:57.442+00:00
updatedAt : 2024-05-26T21:55:43.391+00:00
__v : 2
activated : true
▸ favorites : Array (empty)
▸ orders : Array (empty)
▸ address : Array (empty)
▸ comments : Array (empty)
```

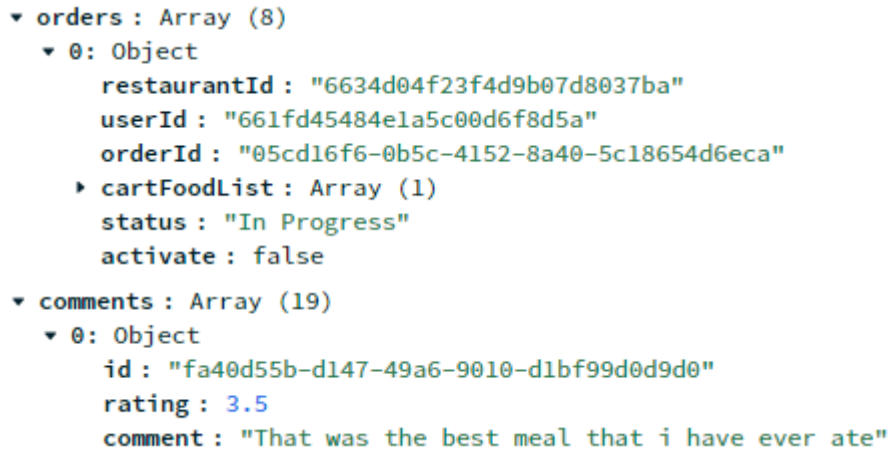
Email is kept as String as the index of the user table. The password is stored encrypted using the bcrypt hash. All users who can log in to the system are stored in this table. The user table itself is independent and does not need other tables to be created. However restaurant reference to these actions is required to track user activities such as placing orders and commenting on orders.

The enabled flag is set to false when a user signs up. The user must check their email to verify their account; this will set the enabled flag to true after successful validation.

The role variable indicates the user's permissions within the system which is set to "user" by default for public accounts.

Initially, the arrays for daily favorites, order addresses, and comments are empty indicating that there has been no activity or interaction yet. These arrays will be populated as the user interacts with the system. The user's orders are linked to the restaurant entity's orders, establishing a relationship between users and restaurants to track orders. Favorites just contains restaurant Id which are favorite restaurants.Adresses contains an object list which address object is: id, adressName, province, district and addresDescription.

The primary index is the email and id field. The super key consists of userId and email. User orders are connected with the restaurant entity's orders establishing a relationship for tracking orders and comments. userId is used for the email system, payment system and comment system.



Restaurant Entity:

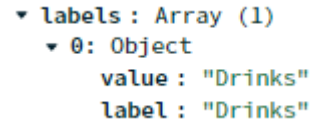
The email is kept as String as the index of the restaurant table. The password is stored encrypted using the bcrypt hash. All restaurants that can log into the system are recorded in this table. The restaurant table itself is freestanding and does not need other tables to be created. However, order and comment data are linked in the same way to the example objects we have given in the user entity above.

When a restaurant registers the activated flag is set to false. The restaurant must check their email to verify their account which upon successful verification will set the activated flag to true.

The role variable indicates the user's permissions within the system defaulting to "restaurant" for restaurant accounts.

Initially arrays for meals, orders, labels and image value are empty implying no activity or interactions yet. These arrays will be populated as the restaurant engages with the system.

The labelList contains all label objects to make efficient filtering and searching restaurants.



Restaurant menus are recorded in the system under the meals list. The Meals list contains food objects. Each food object contains required information and a list of options. In this options list, option objects are listed. Options, for example, are objects where all the options that the food object should include, such as thin edge or thick edge, are recorded when we

want to order a pizza. Each option object is a main heading and has options under it. For example, Edge options are an option. Options such as thin edge and thick edge are also recorded in the elements list inside the option.

The ability to select these options single or multiple is determined by the quantity value in the option object. The user can organize this event as per his/her wishes.

The primary index is the email and id field. The super key consists of _id and email. Restaurant orders and comments are connected with the user entity's orders and comments establishing a relationship for tracking orders and comments. _id is used for the email system, payment system and comment system for.

Kitchens Entity:

ıis entity was created so that labels.

4. IMPLEMENTATION, TESTS, EXPERIMENTS

4.1. Implementation

In this part we explain the technologies and structure of the system. In our project, we use React for the front-end to create a dynamic and responsive user interface, and Node.js for server-side operations and API development for communication between data and front-end. Our database is built on MongoDB, a NoSQL database which is for flexibility and scalability. To handle payments fastly and securely, we use the Stripe API. In addition, we use cookies with Cookie.js packages to construct a user-based homepage structure. These cookies save user activity data, allowing better user experience by tracking and remembering individual preferences and activities.

4.1.1 MongoDB

We use MongoDB asNoSQL database, in our project. With MongoDB's flexible structure we could easily adapt the database without giving it SQL queries when we need to use new credentials. This simplicity allows us to efficiently manage user data, facilitating smooth development and growth user management capabilities.

4.1.2 React

React is a JavaScript package developed by Facebook , which is Meta now , to create user interfaces on the web. It allows you to build dynamic web applications with reusable UI components . In our project, we use React to develop a dynamic and responsive front-end web application allowing us to construct modular components that are dynamic with user inputs, resulting in a good user experience.

4.1.3 Node.js

Node.js is a JavaScript based open-source package used in the development of back-end applications and APIs. In this project, backend operations are performed on Node.js providing APIs for transferring interactively with a React-based front-end connecting to our MongoDB database. Node.js libraries are also employed to generate cookies which save user activities hence enhancing personalization.

4.1.4 Stripe API

Stripe API is a service that is used for get user payments , transections easily. Also it provides good log pages to analyze the transactions. We used stripe API in our project to provide users to buy meals from restaurants.

4.1.5 Cookie.js

Cookies are used for store users behavior and landings , which on specific site , to provide better user experience on site for users. We used Cookie.js to develop cookies in our project to create personalized homepage for users. We suggest the user restaurant based on: previous orders , landings and preferences.

4.2. Tests

Scenario	Test Cases	Test Result
Delete Restaurant	Delete an existing restaurant successfully , Return 404 if restaurant is not found , Handle server errors	Pass
Edit Restaurant	Return 404 if user is not found , Handle null , Handle server errors	Pass

Address Restaurant	Return 404 if no restaurant found by id , Handle server errors	Pass
Orders Restaurant	Retrieve orders successfully , Return 404 if no orders found , Handle server errors	Pass
Login Restaurant	Return 401 if email or password is missing , Return 401 if restaurant is not found , Return 401 if password is incorrect , Handle server errors	Pass
Register Restaurant	Return 500 if there is a server error , Fail with empty object , Fail with missing fields , Fail if restaurant already exists , Handle unexpected errors	Pass
Address User	Add address successfully , Return 404 if no user found , Handle server errors	Pass
Change Password User	Change password successfully , Return 400 if new password does not exist , Return 404 if user is not found , Handle server errors	Pass

Delete User	Delete an existing user successfully , Return 404 if user is not found , Handle server errors	Pass
Edit User	Edit an existing user successfully , Return 404 if user is not found , Handle null , Handle server errors ,	Pass
Address User	Retrieve address successfully , Return 404 if no user found , Handle server errors	Pass
Order Users	Retrieve orders successfully , Return 404 if no order found , Handle server errors	Pass
Login User	Login successfully with valid credentials , Fail with non-activated account , Fail with invalid credentials	Pass
Register User	Fail with empty object , Fail with missing fields , Fail if user already exists , Handle unexpected errors	Pass

5. CONCLUSIONS AND FUTURE WORK

As Feastfive team we determined success criterias as : To ensure users can successfully order foods , restaurant owners can successfully set their restaurants up and make sales securely, and provide service to both parties can easily find each other. We aimed to design simple and minimalistic user interface to not distract users and prove them to reach their goal in minimum steps.We faced some errors when making requests to our own API. Integrating the Stripe API for payments was difficult due to insufficient documentation.We have solved the problem after 10-15 youtube videos and too many tries. Adding the address functionality was challenging because we had trouble understanding the provided service. We resolved these issues through thorough research. Additionally, making each page responsive for mobile, tablet, and PC took a lot of time because we had to design for all devices. We could have developed and implemented an advanced ai model to recommend for both users to buy food and restaurant owners to growth his/her business. We could have implement ai voice assistant for manage transaction and recommendation with user. This could have create sense of belonging. We could have added a chat section between restaurant and user who gave order from them.

6. References

Yemeksepeti - Vikipedi. 15 Dec. 2009, <tr.wikipedia.org/wiki/Yemeksepeti>

Getir - Vikipedi. 2 Feb. 2020, <tr.wikipedia.org/wiki/Getir>

HTTP | Node.js v22.2.0 Documentation. <nodejs.org/api/http.html>

ReactJS <https://react.dev >

NodeJS <https://nodejs.org/en >

JavaScript <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript >

Stripe API Reference. <docs.stripe.com/api>

"Npm: Cookie." *Npm*, <www.npmjs.com/package/cookie>

Khan, Aneeqa. "Creating REST API Routes in Node.js." *DEV Community*, 5 Nov. 2023, <dev.to/aneeqakhan/creating-rest-api-routes-in-nodejs-207k>.

"Install Tailwind CSS With Create React App - Tailwind CSS." *Tailwind CSS*, <tailwindcss.com/docs/guides/create-react-app>

Adding a CSS Modules Stylesheet | Create React App. 4 Jan. 2019, <create-react-app.dev/docs/adding-a-css-modules-stylesheet>

"Npm: Nodemailer." *Npm*,<www.npmjs.com/package/nodemailer>

"Npm: Bcryptjs." *Npm*, <www.npmjs.com/package/bcryptjs>

"Npm: Jsonwebtoken." *Npm*,< www.npmjs.com/package/jsonwebtoken>

"Npm: Uuid." *Npm*, <www.npmjs.com/package/uuid>

"Npm: Mongoose." *Npm*, <www.npmjs.com/package/mongoose>

Testing React Apps · Jest. 12 Sept. 2023, <jestjs.io/docs/tutorial-react>

"Npm: React-toastify." *Npm*, <www.npmjs.com/package/react-toastify>

"Npm: Js-cookie." *Npm*,< www.npmjs.com/package/js-cookie>