Master Team Project WS 2022: Risto

Team 3 of Global Distributed Software Development (Hochschule Fulda):

- **Luis Miguel García Marín (Team Lead)**
  luis-miguel.garcia-marin@informatik.hs-fulda.de
- **Jesús Moreno Durán (Back End Lead)**
- **Noman Ali (Front End Lead)**
- **Paras (GitHub Master)**
- **Vichitar Dagar (Back End Team Member)**
- **Alhassane Dondo Toure (Front End Team Member)**

Milestone 4
Date: 13.02.2023

History Table	
Date submitted	**Date revised**
10.03.2023	

1. Product summary

<u>Name of the product:</u> Risto.

<u>List of all major committed functions (priority 1):</u>

1. Restaurants can register in the system.
2. Restaurants can upload information about themselves to subscribe to the service.
3. Site administrators can approve or decline restaurant info for posting.
4. Site administrators can deal with typical admin duties.
    - 4.1. Site administrators can manage user registrations (they are able to ban registered users and accept or decline restaurant registrations).
    - 4.2. Site administrators can accept or decline review posts.
    - 4.3. Site administrators can delete posts.
    - 4.4. Site administrators can ban users with bad behavior.
5. Customers can search for restaurants.
6. Customers can manage their reservations.
    - 6.1. Customers can make a reservation.
    - 6.2. Customers can change their reservation.
    - 6.3. Customers can cancel their reservation.
7. Customers can make orders of food before their arrival to a restaurant in which they have made a reservation.
8. Customers can manage posts of reviews.
    - 8.1. Customers can post reviews.
    - 8.2. Customers can delete their reviews.
9. Managers (Host/hostess) can review daily calendar.
10. Managers (Host/hostess) can check incoming guests to greet them.
11. Managers (Host/hostess) can check the table status of their restaurant.
12. Managers (Host/hostess) can set tables for free when they are available.
13. Managers (Host/hostess) can set parking slots for free when they are available.
14. Waiters can check the food order list.
15. Waiters can fetch orders and their corresponding tables.
16. Waiters can set food orders which they have given to a table as delivered.
17. Customers can check the list of food with their price.
18. Customers can reserve a parking spot near the restaurant parking point.
19. Customers can chat with restaurant managers.

<u>Functionalities that make us unique:</u>
<u>Live View (Not finished yet)</u>
With our system the customer can have a live view (top view of a graphical map of the restaurant with tables represented by diagrams) of the tables present in the restaurant and status of their occupancy. So customers can see exactly what tables are available and where those tables are located, for example if customers want a table near a window or at the very end, something more discrete, they can see that on the web page and book accordingly. *The status of table occupancy will be updated by the manager of the restaurant when the customer has paid and left the table.*

<u>Pre-order Food</u>

With us, customers can even pre-order food during your reservation. They will have full access to the Menu on the web page and can pre-order food before even reaching the restaurant. When they arrive at the time of reservation, the food will be there in no time. The chef will be notified of the upcoming pre-orders shortly before the time of arrival of the customer.

Waiter food ordering management
Waiters of the restaurant can manage the user's food orders, so they can see what they have ordered and to which table they have to deliver it. After doing that, they can set those orders as delivered, so they can filter the orders they have not delivered yet. This is useful for them in order to make their management of orders easier and more organized, and also to give feedback to the chef.

Reserve Parking while booking Table
Customers can even reserve a Parking spot while booking a table. So when they arrive at the restaurant they don't have to look for a parking spot, this results in a much more smooth and stress-free dining experience.

URL to product: http://52.3.248.66/
This IP is dynamic, so it can change every time the web server is deployed.

2. Usability test plan

Test Objectives:
The main objective of this usability test is to evaluate the effectiveness, efficiency, and user satisfaction of the "Make a Reservation" function in Risto. The test aims to identify any usability issues that may hinder the user's ability to make a reservation, pre-order food, and manage their reservations.

Test Background and Setup:
System Setup: The testing will be conducted using the Risto web application. The application will be accessed through a desktop computer running Windows 10, using the Google Chrome browser or Microsoft Edge browser.
Starting Point: The starting point for this test is the home page of the Risto web application.
Intended Users: The intended users of this test are individuals who are interested in making a reservation at a restaurant using the Risto web application.

URL of the system: http://52.3.248.66/ (This IP is dynamic, so it can change every time the web server is deployed)

What is to be Measured: The test will focus on evaluating the usability of the "Make a Reservation" function in Risto. Specifically, we will be measuring the effectiveness, efficiency, and user satisfaction of the reservation process.

Usability Task Description:

1. Imagine that you want to make a reservation at a restaurant using the Risto web application.
2. Go to the Risto home page.
3. Search for a restaurant in your area using the search bar.
4. Select a restaurant from the search results.
5. Click on the "Make a reservation" button.
6. Fill out the reservation form, selecting the date and time of your reservation, the number of people in your party, an available parking place if you want and any food items you want to pre-order.
7. Submit the reservation form.
8. Check the "Reservations" section of the web app to confirm that your reservation has been successfully made.

Effectiveness Measurement:
The effectiveness of the "Make a Reservation" function will be measured by the percentage of successfully completed reservations. A reservation will be considered successful if it is created without any errors or issues.

Efficiency Measurement:
The efficiency of the "Make a Reservation" function will be measured by the time taken to complete the reservation process, from selecting the restaurant to submitting the reservation form. Maximum time to complete this task should be 15 minutes.

Likert Scale Questions:

- How easy was it to find and select a restaurant using the Risto search bar?
    1: Very difficult
    2: Somewhat difficult
    3: Neutral
    4: Somewhat easy
    5: Very easy
- How clear was the reservation form in Risto?
    1: Very unclear
    2: Somewhat unclear
    3: Neutral
    4: Somewhat clear
    5: Very clear
- How satisfied are you with the overall reservation process in Risto?
    1: Very dissatisfied
    2: Somewhat dissatisfied
    3: Neutral
    4: Somewhat satisfied
    5: Very satisfied

3. QA test plan

QA Test Plan for "Make a Reservation" Feature in Risto Web Application:
Test Objectives:
The objective of this QA test plan is to verify the functionality and user experience of the "Make a Reservation" feature in Risto web application, ensuring that it meets the following requirements:

- The reservation form is displayed correctly with all necessary fields.
- The user can select a date, time, number of guests and pre-order food.
- The user can submit the reservation form and receive a confirmation message.
- The reservation is saved correctly in the database.
- The user can view their reservations in the "Reservations" section of the web app.

HW and SW Setup:

- URL: http://52.3.248.66/ (This IP is dynamic, so it can change every time the web server is deployed)
- Frontend: ReactJS
- Backend: NodeJS with Express and Sequelize for MySQL database management
- Browsers: Google Chrome and Microsoft Edge

Feature to be Tested:
Make a reservation for a Restaurant in Risto web application.
QA Test Plan Table:

Test #	**Test Title**	**Test Description**	**Test Input**	**Expected Correct Output**	**Test Results (Chrome)**	**Test Results (Microsoft Edge)**
1	Reservation form display	Verify that the reservation form is displayed correctly with all necessary fields.	Click "Make a Reservation" button on a restaurant page.	Reservation form is displayed with fields for date, time, number of guests and pre-order food.	PASS	PASS

2	Submit reservation	Verify that the user can submit the reservation form and receive a confirmation message.	Fill out the reservation form and click "Submit" button.	Confirmation message is displayed and reservation is saved correctly in the database.	PASS	PASS
3	View reservations	Verify that the user can view their reservations in the "Reservations" section of the web app.	Click "Reservations" tab in the header menu.	The list of all reservations made by the user is displayed with the correct details.	PASS	PASS

Note: All tests were conducted with a user account that has previously logged in to the Risto web application.

Test Environment:

- Operating System: Windows 10
- RAM: 8GB
- Browser 1: Google Chrome version 97.0.4692.99
- Browser 2: Microsoft Edge version 97.0.1072.62

Test Results:

All test cases passed in both Google Chrome and Microsoft Edge. No defects or issues were found during testing.

4. Code Review

Our coding style:

- Code should be self-explanatory, but comment when needed or required.
- Semicolons are mandatory at the end of the instructions.
- Use of Camel case notation.
- Code should have correct indentation in order to be easily readable.
- Minimum use of database requests.
- Header and in-line comments of files should be correct.

Knowing our coding style, we are going to do some peer review of the code that our team mates have done. The distribution will be the following:

This person:	-	Is reviewed by:
Luis	-	Jesus
Jesus	-	Vichitar
Vichitar	-	Luis
Paras	-	Hassan
Hassan	-	Noman
Noman	-	Paras

That means:

Luis reviews Vichitar's code.
Jesus reviews Luis' code.

Vichitar reviews Jesus' code.
Paras reviews Noman's code.

Noman reviews Hassan's code.
Hassan reviews Paras' code.

Now, we specify the name of the files that we are going to provide for code review:

Luis:
Frontend: Chat.jsx, Navbar.jsx
Backend: chat.controller.js

Jesus:
Frontend: RestaurantTabDetails.jsx
Backend: orderReservation.controller.js, restaurantRegistrationPetition.controller.js

Vichitar:
Frontend: ManagerReservationCard.jsx
Backend: Tables.controller.js,
Reviews.controller.js (**excluded**: findAllAccepted, findAllPending, findByRestaurantAccepted)

Paras:
Frontend: OrderList.jsx, ReviewCheckBox2.jsx
Backend: manager-waiter.model.js, users.controller.js (partially done - waiter-manager mapping part)

Hassan:
Frontend: OtherUserProfile.jsx, userProfile.jsx
Backend: rols.controller.js

Noman:
Frontend: ManagerPanel.jsx, ManagerReservationCard.jsx
Backend: foodCategory.controller.js

The reviewed files with their comments have been sent by email in a .zip file and they can be also found inside the “peer review” folder of the repository.

5. Self-check on best practices for security

Major Assets:

- User and restaurants data (personal information, login credentials)
- Application code

Major Threats:

- Unauthorized access or theft of user data
- Code injection, hacking or theft of application code

Protection Measures:

- User data is stored in a secure database with access controls and checking roles. Passwords are encrypted using a secure hashing algorithm (Bcrypt).
- Code is stored in a secure version control system with access controls (Github private repository), and the application is designed with secure coding practices such as input validation and parameterized queries.

Confirmations:

- Yes, passwords are encrypted using a secure hashing algorithm (Bcrypt) before storing them in the database.
- Input data validation is implemented on the search bar for up to 40 alphanumeric characters. The input is validated using client server code, since in the server-side it is prevented by restricting the input to identifiers, so we can avoid SQL injection attacks and other types of malicious input.

In addition to these self-checks, it's important to regularly review and update security measures to ensure that the application remains protected against emerging threats and vulnerabilities. This may include periodic security audits, penetration testing, and ongoing security training for developers and other personnel involved in the development and maintenance of the application.

6. Self-check: Adherence to original Non-functional specs

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from chosen cloud server. **DONE.**

2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers. **DONE.**
3. All or selected application functions must render well on mobile devices. **DONE.**
4. Data shall be stored in the database on the team's deployment cloud server. **DONE.**
5. Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner. **DONE.**
6. No more than 50 concurrent users shall be accessing the application at any time. **DONE.**
7. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. **DONE.**
8. The language used shall be English (no localization needed). **DONE.**
9. Application shall be very easy to use and intuitive. **DONE.**
10. Application should follow established architecture patterns. **DONE.**
11. Application code and its repository shall be easy to inspect and maintain. **DONE.**
12. Google analytics shall be used (optional for Fulda teams). **DONE.**
13. No e-mail clients shall be allowed. **DONE.**
14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. **DONE.**
15. Site security: basic best practices shall be applied (as covered in the class) for main data items. **DONE.**
16. Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today. **DONE.**
17. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development. **DONE.**
18. For code development and management, as well as documentation like formal milestones required in the class, each team shall use their own GitHub to be set-up by class instructors and started by each team during Milestone 0. **DONE.**
19. The application UI (WWW and mobile) shall prominently display the following exact text on all pages "Fulda University of Applied Sciences Software Engineering Project, Fall 2022 For Demonstration Only" at the top of the WWW page (Important to not confuse this with a real application). **DONE.**