<Restaurant Booking System>

System Design

<1.0>

<15.12.2019>

<Oğuz Kaan Yazan>

<Volkan Davşan>

<Berke Biten>

<Ali Atahan Topal>

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc433996772)

[1.1. Purpose of the System 1](#_Toc433996773)

[1.2. Design Goals 1](#_Toc433996774)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc433996775)

[1.4. References 1](#_Toc433996776)

[2. Current Software Architecture 1](#_Toc433996777)

[3. Proposed Software Architecture 1](#_Toc433996778)

[3.1. Overview 1](#_Toc433996779)

[3.2. System Decomposition 1](#_Toc433996780)

[3.3. Hardware Software Mapping 2](#_Toc433996781)

[3.4. Persistent Data Management 2](#_Toc433996782)

[3.5. Access Control and Security 2](#_Toc433996783)

[3.6. Global Software Control 2](#_Toc433996784)

[3.7. Boundary Conditions 2](#_Toc433996785)

[4. Subsystem Services 2](#_Toc433996786)

[5. References 2](#_Toc433996787)

SYSTEM DESIGN DOCUMENT

# Introduction

## Purpose of the System

Our purpose by creating this system is to create a platform where people can make reservations from any restaurant he/she likes. Also, people can learn a lot of information about that specific restaurant before making any reservation. This reduces the people’s discomfort because he already knows what to expect when going into that restaurant. In our system, we also give users the freedom to look any restaurant they want until they find one that meets their taste. So, by using our system people can make booking and they do it by losing much less time than going into that specific restaurant to do booking. Also, all of the informations about restaurants will be shown to the user before user does booking. Users can also drop a review to that specific restaurant to let other users know his thoughts.

## Design Goals

So in our project, we divided design goals into four parts which is performance, dependability, end user criteria and maintenance.

**First part is Performance:**

Our systems response time is very quick, when a user requests something, it gets acknowledged almost instantly, there is no delay that makes them wait and the user gets the feedback regarding the operation.

**Second part is Dependability:**

So our system is robust because it can handle invalid user inputs by showing a warning which explains why the user can not do that operation. By doing that, we disable all of the invalid inputs from entering our system.

Our system also holds true for reliability because when the Users wants to do anything related to our system, all he have to do is clicking the right button, for example when user clicks the “Make a Reservation” button, he can make reservations in the following page. In other words, our system’s specified behaviour matches with the user’s observed behaviour correctly.

Our system can be used 24/7 with no problem but when doing booking to a specified restaurant, you must choose a restaurant hour when the restaurant is available. But other than that function, all of the other functions are available for users to use. So it checks the Availability criteria.

**Third part is Maintenance:**

So our system is easy to understand by looking at the code because we named all of the buttons or classes or id’s based on our function. For example if our function is booking, all of our variables are named according to booking. Also our css’s are gathered under one page and all of the css’s are ordered by their functions so it is very easy to understand which css codes’ belongs to which function. Lastly, we noted some comments in the code to make it easier to understand for us and for anyone else who reads the code.

Also, our system holds true for extensibility criteria because, we can add new functions to our existing system with no problem or minimal problem.

**Last part is the End User:**

Our system is very easy to use, all of the functions are clearly specified by a button which activates it. So all user’s have to do is click the right button(buttons are clearly named in order to be no conflict.) and enters the valid input. Even if they enter wrong inputs, they get feedback about what to fix. We also have a Support Page in the homepage which makes things easier for all users.

Utilization of our system is also good, system supports the work of the user by showing him alerts to let them know if their operation is a success or failure. If it is failure, system explains what to fix and how to fix it.

## Definitions, Acronyms, and Abbreviations

**User:** A customer that is registered to the web site and can use the booking function to book a table from a restaurant.

**Restaurant Owner**: Owner of a restaurant that is registered to the website and can see the bookings that have been made through to our website from his main page.

**Guest:** A customer that is not registered but can register and search restaurants from web site but not make bookings.

**Admin:** Admin is a user with access and handle all the pages and responds to the normal user and restaurant owners’ tickets and requests.

**IDE:** An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.

**Local Host:** A host that is always on your computer. A virtual server that can test web applications without connecting to internet.

**RBS:** Restaurant Booking System.

## References

Restaurant Booking System RAD Document Final Version.

# Current Software Architecture

The current architectures that are similar systems to our system have the same layers as RBS. These are Database, Web Server and Interfaces. So, most of the systems that are similar are using the three-layered architecture.

# Proposed Software Architecture

## Overview

This section describes the requirements of the distinct proposed software architecture.

We divided the system into multiple subsystem and layer.Each subsystem provide or require a service and interact with each other. We have shown that interactions and described the subservices.These subsystems can run on separate hardware or software.We described it in the hardware software mapping.

In this system, we’ll have too much data.We explained our decisions about the storage and management of the data.We have to keep safe these datas.In the part 3.5, we described how will we ensure the security and how does work the authentication system.

Also we described global software control and boundary conditions in this section.

## System Decomposition

**Interfaces:** Interface is the website. It will be used by users (guest, user, restaurant owner, admin). Provides a graphical user interface for related functions.

**Application Logic:** Provides the functionalities of the system. It gets the data sent from the website and delivers it to the database. And gets data from database to provide statistics or results to the interface.

**User Management:** Provides the function which enables admins to manage users.

**Search:** Gets inputs from interface and searches related data in database. Delivers results to the interface.

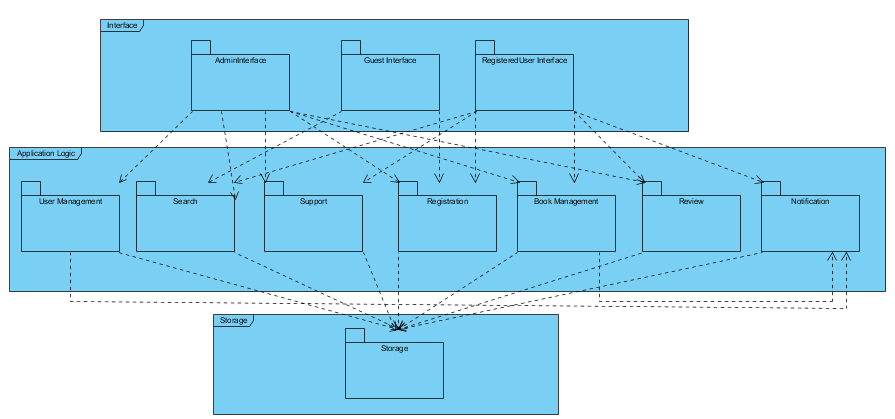
**Support:** Provides function for Ticket System.

**Registration:** Provides the functions related to Registration (sign up, restaurant sign up, sign in, forgot password, sign out)

**Book Management:** Provides the functions which are related to bookings.

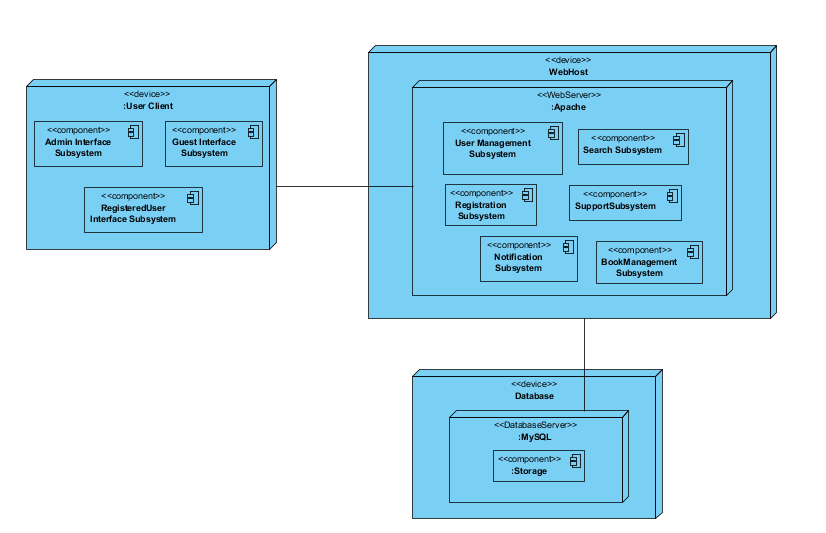
**Review:** Provides the Review System.

**Notification:** Provides the notification function.



## Hardware Software Mapping

We separated the map to three parts because our system will use a three-layered architecture. Interface subsystems are mapped in User Client Hardware which interacts with the user. Application Logic subsystems are mapped in Webhost Hardware. Which takes input / events from User Client and delivers them to database or take data from database to operate. Storage Subsystems are mapped in Database Hardware.



## Persistent Data Management

The persistent data will be stored with 2 ways; in the file system and database system. There is 2 different type of persistent data.

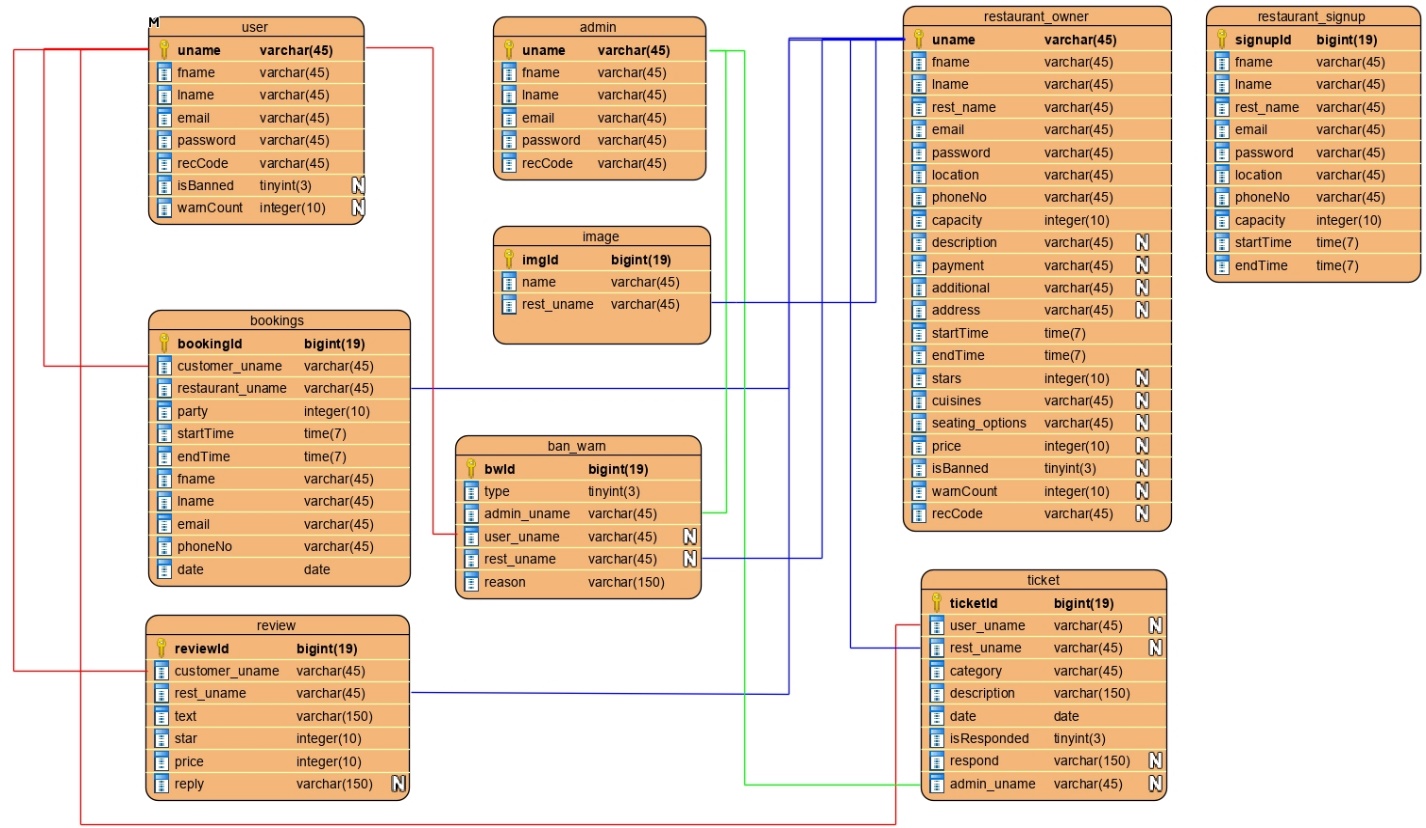
**Static Data:**

Our HTML, PHP, CSS and JavaScript files are static data. And we will store them in the file system.

**Dynamic Data:**

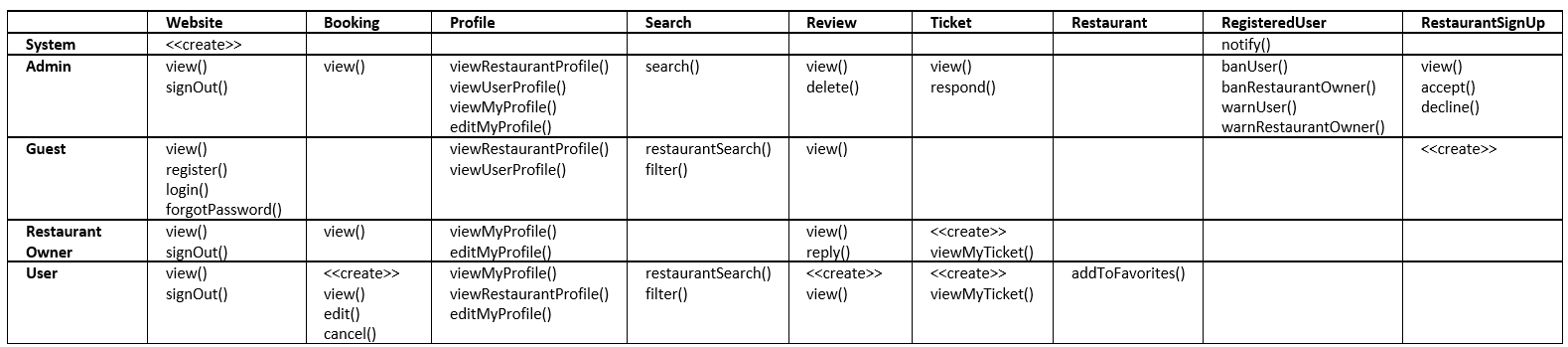
We have a large dynamic data set and we allow the users to perform a lot of operations with these data in the system. So we have to run complex queries for these operations. For that reason, we decided to store all the data created by users and restaurants (like user informations, bookings, reviews etc.) in the relational database. Storing the data in the relational database enables the system to perform complex queries on a large data set.

Also we have images that will be uploaded by restaurant owners. And we have two options for the storing this images. For the relational database, we already have a large data set and adding the images to the database will increase too much the size of the database. We’ll backup the system to protect any failures and increase in the size of the database would make it harder. Storing images in the file system is easier and that’s why, we will store the images in the file system but we will keep the path of the image in the database.



## Access Control and Security

Each user will have a password and username along other fields. These will be stored in the database in encrypted form. Even admins will not be able to see users’ actual passwords. Users will have private data fields and these will not be enable to other users. Restaurant Booking System will use a Logon Authentication Mechanism. Users will enter username and password in order to log in to the system. System will keep unauthorized people from gaining access to recourses which they must not access.



## Global Software Control

Restaurant Booking System uses both event-driven and procedure-driven control flow in different subsystems. Whenever an event occurs, it is dispatched to the appropriate subsystem. For example if a user wants to book a table from a restaurant, whenever he clicks the button this event will be dispatched to BookManagement subsystem and user will be available to perform the functions which BookManagement provides. RBS also uses procedure-driven control flow in various functions. For example a guest wants to sign up and clicks the sign up button, Registration subsystem will show a form to the guest and will wait for him/her to fill the required fields and submit them.

## Boundary Conditions

**Prerequisites**

For the users to successfully enter the website, user must have an internet connection.

**Start-up and shutdown**

**Start-up and login:** When the user enter the website, most popular locations and restaurants will be calculated and all boundary objects in the homepage will be loaded and shown.

When a user logged in to website, system determines which type of user he/she is and redirects the user to related homepage.

**Shut Down:** When the restaurant booking system is shut down, it will ensure any process running at that time will be finished.

**Error Behaviour:**

If any errors originated from user-input occurs, system will not process the following operation and give a warning which notifies the user.

If any network failure occurs, user’s connection to the website will be interrupted.

If the server is unexpectedly terminates backups will be made to make sure damage by server failure is minimal.

# Subsystem Services

Restaurant booking system has multiple subsystem and we have divided into three layers. These are interface, application logic and storage. In the interface layer, there are AdminInterface, GuestInterface and RegisteredUser Interface subsystems. All interface subsystem has their own homepage and in the RegisteredUser interface there are two homepages determined by the user type.

In the application logic layer, there are User management, search, support, registration, book management and notification subsystems.

**User management subsystem provides following services to the Admin:**

banUser()

banRestaurantOwner()

warnUser()

warnRestaurantOwner()

searchUsers()

viewRestaurantSignUp()

acceptRestaurantSignUp()

declineRestaurantSignUp()

**Search subsystem provides following services:**

restaurantSearch()

locationSearch()

filter()

**Support subsystem provides following services:**

sendTicket()

viewTicket()

viewMyTickets()

respondTicket()

viewFAQ()

**Registration subsystem provides following services:**

signUp()

restaurantSignUp()

signIn()

signOut()

forgotPassword()

**Book Management subsystem provides following services:**

book()

editBook()

cancelBook()

view()

dropReview()

reply()

calculateMostPopularLocations()

calculateMostPopularRestaurants()

deleteReview()

**Notification subsystem provides following services:**

notify()

# References

|  |
| --- |
| 1.Bernd Bruegge, Allen H. Dutoit, Object Oriented Software Engineering Using UML, Patterns and Java, 3rd ed., Prentice Hall |

2. Restaurant Booking System RAD Document Final Version.