Purpose

The results of the requirements elicitation and the analysis activities are documented in the Requirements Analysis Document (RAD). This document completely describes the system in terms of functional and nonfunctional requirements and serves as a contractual basis between the client and the developers.

Audience

The audience for the RAD includes the client, the end users, the project manager, and the developers.

Table of Contents

Document History

Rev.	Author	Date	Changes
1	Vadym Khyzhniak	01.08.2022	Added the Introduction, Current system and Proposed system (requirements) parts.
2	Oumaima Regaieg	01.08.2022	Added the system models

1. Introduction

The purpose of the Introduction is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context. The introduction also includes the objectives and success criteria of the project.

1.1 Purpose of the system

This system is developed to provide restaurant reservation service in Munich.

1.2 Scope of the system

Users can find restaurants and book tables in Munich.

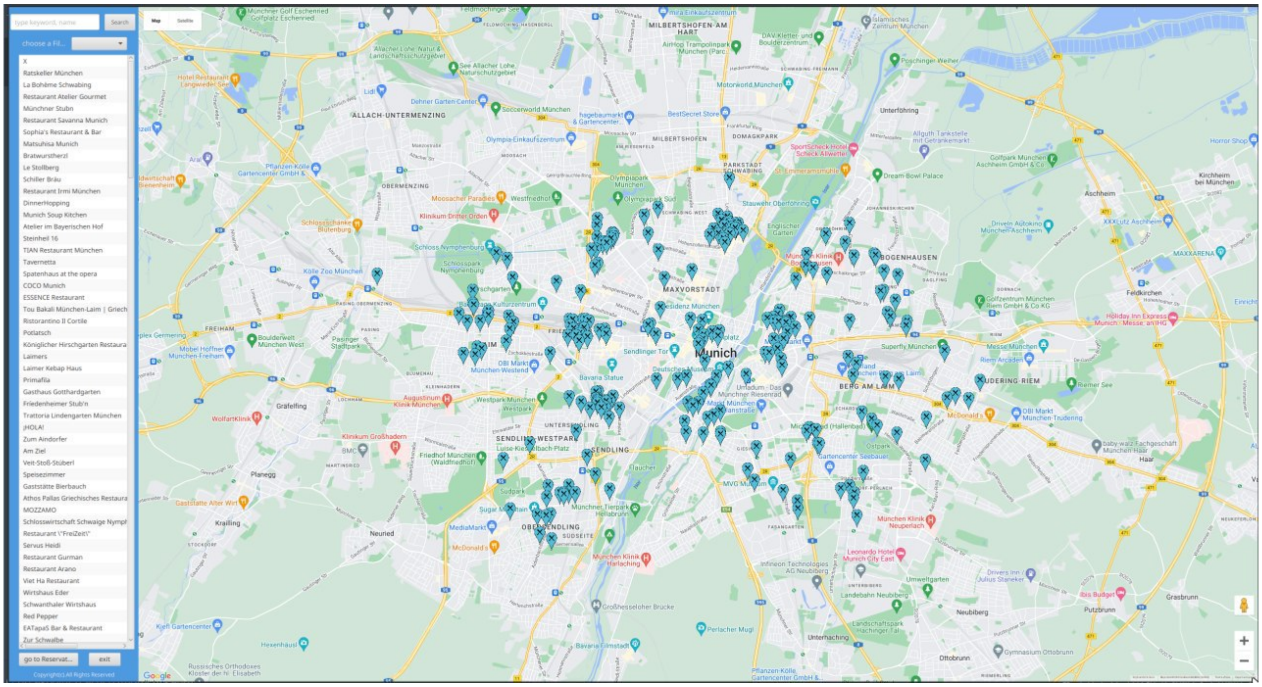
1.3 Objectives and success criteria of the project

Users can search for a restaurant in Munich, check the opening hours, rating, price level, available tables and time slots. They have a possibility to book a table. If they are registered they can see booked tables in their calendar.

1.4 Definitions, acronyms, and abbreviations

1.5 References

1.6 Overview



2. Current system

This section describes the current state of affairs. If the new system will replace an existing system, this section describes the functionality and the problems of the current system.

The system is already in the functional state. All requirements are completed. The restaurant filtering could be also applied to the restaurant markers on the map (currently filters work exclusively for the text search). The table reservation process and reservation confirmation could be more intuitive.

3. Proposed system

The third section documents the requirements elicitation and the analysis model of the new system.

3.1 Overview

The overview presents a functional overview of the system.

Peter lives in Munich and wants to visit a Turkish restaurant on Friday evening with his friend. He opens the Reservation system app, enters as a guest, filters the restaurants by cuisine (Turkish), finds the nearest one on the map, checks its' name by clicking on the marker. Types the name in the search bar, clicks on the found result, clicks on "go to Reservation" button, chooses the date, available table on the table schema, time slot, the number of guests, clicks "Reserve". The reservation is successful.

3.2 Functional requirements

Functional requirements describe the high-level functionality of the system. This section list all functional requirements and additionally presents the dependencies between them.

3.2.1 Search for restaurants

The user can search for restaurants on a list and on a map that displays up to 50 restaurants.

3.2.2 See restaurants details

The user can see pictures, ratings and comments of the restaurant as well as opening times and a link to the website.

3.2.3 Filter search results

He can filter the results by the restaurant type, the prize category, by distance around a certain location, by the average rating and by free time slots for reservations for specified dates and number of visitors.

3.2.4 Reserve table

A user can see the times when he can reserve a table in the chosen restaurant. After clicking on the time, the user sees an overview of all tables in the restaurant. He can choose the exact table the free one in the overview and thus reserve the table for the specified number of visitors.

3.2.5 Save calendar event

When the user reserves a table, an event in the local calendar is created for the reservation.

3.2.6 Confirm reservation

A user is reminded about a reservation one day before the actual date of the reservation and must confirm it until latest 12 hours before the actual date. If the user does not confirm, his reservation is cancelled automatically.

3.2.7 Cancel reservation

A user can cancel his reservation at any time up to two twelve hours before the actual date of the reservation. After the confirmation (see “Confirm reservation”), the user cannot cancel the reservation anymore.

3.3 Nonfunctional requirements

Nonfunctional requirements describe user-level requirements that are not directly related to functionality. This includes usability, reliability, performance, supportability, implementation, interface, operational, packaging, and legal requirements. The section list all these non-functional requirements and additionally presents the dependencies between them.

3.3.1 Usability

The system should be intuitive to use, and the user interface should be easy to understand. Simple interactions should be completed in less than three clicks. Complex interactions should be completed in less than six clicks.

3.3.2 Conformance to guidelines

The design of the system should conform to the typical usability guidelines such as Nielsen's usability heuristics.

3.3.3 Server System

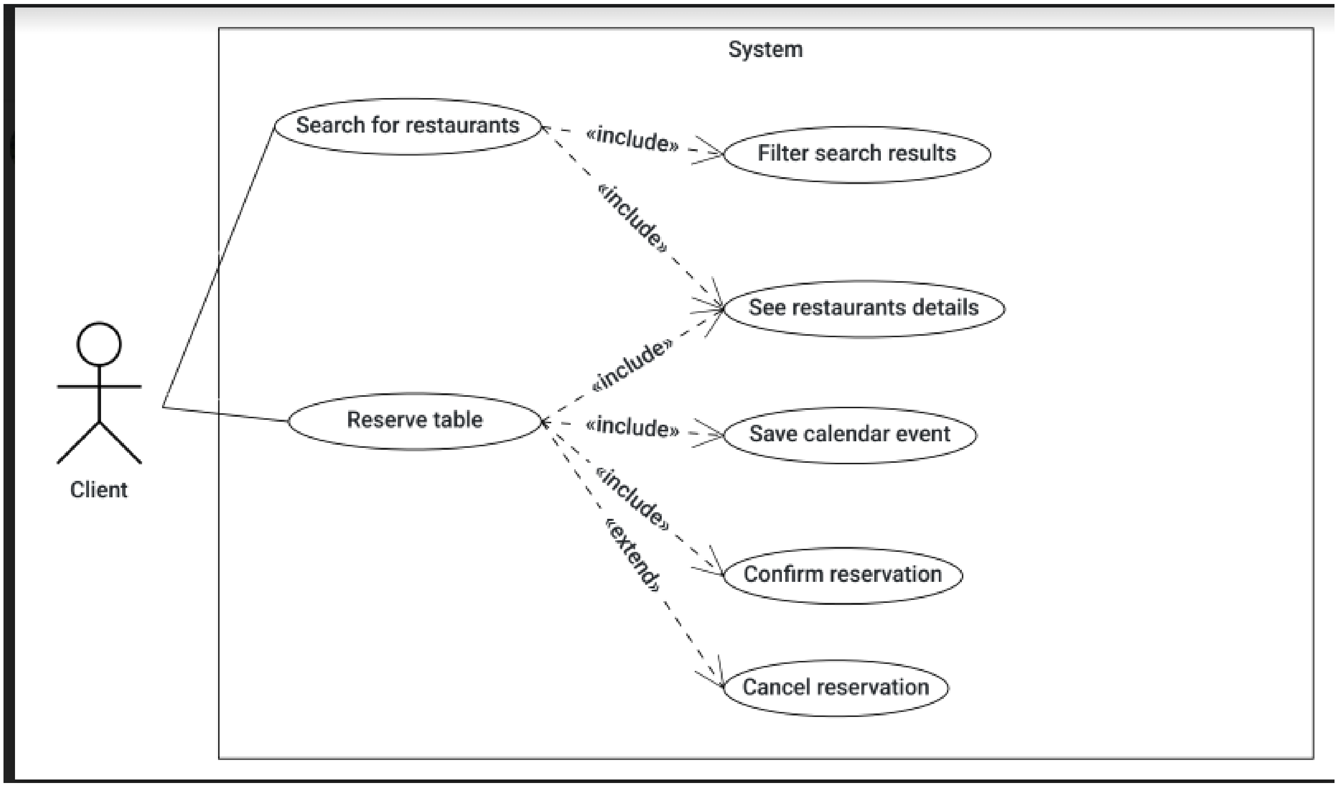
A server subsystem with a couple of services must be used in the system.

3.4 System models

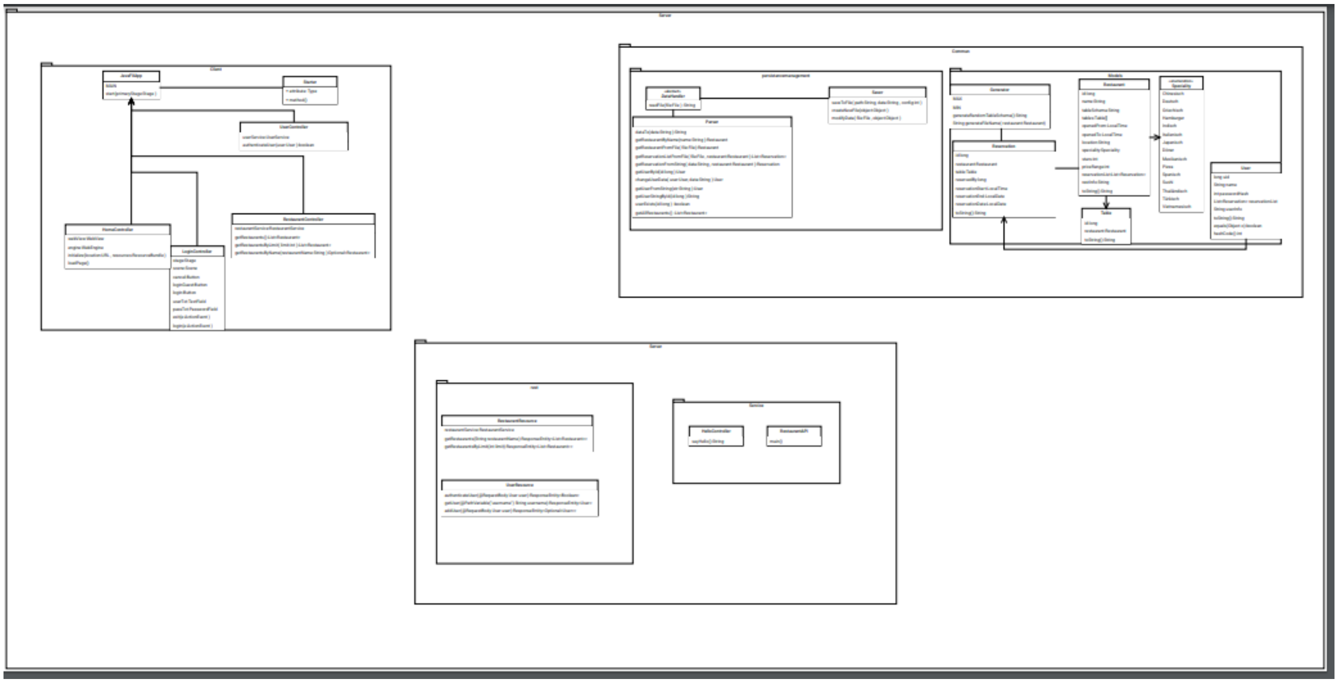
The System models include scenarios, use cases, object model, and dynamic models for the system. This section should contain the complete functional specification, including mock-ups, paper-based prototypes or storyboards illustrating the user interface of the system and navigational paths representing the sequence of screens.

3.4.1 Scenarios

3.4.2 Use case model



3.4.3 Object model


Server
Client
JavaFXApp
MAIN
start(primaryStage:Stage)
Starter
+ attribute: Type
+ method()
UserController
userService:UserService
authenticateUser(user:User):boolean
HomeController
webView:WebView
engine:WebEngine
initialize(location:URL, resources:ResourceBundle)
loadPage()
LoginController
stage:Stage
scene:Scene
cancelButton
loginGuestButton
loginButton
userTxt:TextField
passTxt:PasswordField
exit(a:ActionEvent)
login(a:ActionEvent)
RestaurantController
restaurantService:RestaurantService
getRestaurants():List<Restaurant>
getRestaurantsByLimit(limit:int):List<Restaurant>
getRestaurantsByName(restaurantName:String):Optional<Restaurant>
Common
persistancemanagement
«abstract»
DataHandler
readFile(file:File):String
Saver
saveToFile(path:String, data:String, config:int)
createNewFile(object:Object)
modifyData(file:File, object:Object)
Parser
dataTo(data:String):String
getRestaurantByName(name:String):Restaurant
getRestaurantFromFile(file:File):Restaurant
getReservationListFromFile(file:File, restaurant:Restaurant):List<Reservation>
getReservationFromString(data:String, restaurant:Restaurant):Reservation
getUserById(id:long):User
changeUserData(user:User, data:String):User
getUserFromString(str:String):User
getUserStringById(id:long):String
userExists(id:long):boolean
getAllRestaurants():List<Restaurant>
Models
Generator
MAX
MIN
generateRandomTableSchema():String
String generateFileName(restaurant:Restaurant)
Reservation
id:long
restaurant:Restaurant
table:Table
reservedBy:long
reservationStart:LocalTime
reservationEnd:LocalDate
reservationDate:LocalDate
toString():String
Restaurant
id:long
name:String
tableSchema:String
tables:Table[]
openedFrom:LocalTime
openedTo:LocalTime
location:String
speciality:Speciality
stars:int
priceRange:int
reservationList:List<Reservation>
restInfo:String
toString():String
Table
id:long
restaurant:Restaurant
toString():String
«enumeration»
Speciality
Chinesisch
Deutsch
Griechisch
Hamburger
Indisch
Italienisch
Japanisch
Döner
Mexikanisch
Pizza
Spanisch
Sushi
Thailändisch
Türkisch
Vietnamesisch
User
long uid
String name
int passwordHash
List<Reservation> reservationList
String userInfo
toString():String
equals(Object o):boolean
hashCode():int
Server
rest
RestaurantResource
restaurantService:RestaurantService
getRestaurants(String restaurantName):ResponseEntity<List<Restaurant>>
getRestaurantsByLimit(int limit):ResponseEntity<List<Restaurant>>
UserResource
authenticateUser(@RequestBody User user):ResponseEntity<Boolean>
getUser(@PathVariable("username") String username):ResponseEntity<User>
addUser(@RequestBody User user):ResponseEntity<Optional<User>>
Service
HelloController
sayHello():String
RestaurantAPI
main()

3.4.4 Dynamic model

3.4.5 User interface

4. Glossary

A glossary of important terms used in the project and in the system model ensures consistency in the specification and a common understanding of terms used by the client.