< Rent a Car >

System Design

<4.0>

<24-05-2021>

< Yasin Taş >

< Zahir Mert Tatlı >

< Salah eddine EL Ballouti >

< Okan Üregen>

< Mertcan Özdemir>

< Ali Canalp Cansever>

<Akın Özçıtak>

Prepared for

SOFT3102 Software Engineering Practice



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 2](#_Toc433996778)

[3.1. Overview 3](#_Toc433996779)

[3.2. System Decomposition 3](#_Toc433996780)

[3.3. Hardware Software Mapping 4](#_Toc433996781)

[3.4. Persistent Data Management 4](#_Toc433996782)

[3.5. Access Control and Security 5](#_Toc433996783)

[3.6. Global Software Control 6](#_Toc433996784)

[3.7. Boundary Conditions 7](#_Toc433996785)

[4. Subsystem Services 7](#_Toc433996786)

[5. References 8](#_Toc433996787)

SYSTEM DESIGN DOCUMENT[1]

# Introduction

Purpose of the Rent a Car project is to provide the users, a safe and fast renting service on users’ android phones. The biggest reason behind the project development is to make rent a car service easier to reach anywhere and at any time. Project’s main goal is to satisfy the users with the app and adapt the ‘rent-a-car’ system to the ‘smart world’.

## Purpose of the System

The main purpose of System is to make it easy for people to rent cars online using their mobile devices. The system also creates a safe and fast experience and satisfying interface for the users. By using our system to rent a car, the user no longer needs to call the rental agency and spend too much time on something that can be done fast and easily.

## Design Goals

We aimed to develop a user-friendly interface. For this, attention has been paid to the readability of the texts and the color harmony between the elements on the window. Colors, that are easy on the eyes and not distracting, have been chosen. The design has been improved with some tactics such as shading, border, border-radius and what is called “negative space” in design. Necessary spaces have been left in appropriate places on the page. At the same time, the size of the text and elements (cards, buttons, etc.) was carefully adjusted. Our system is designed so that the user does not waste too much time to perform an action and can take the desired action in a short time, we avoided overwhelming the user with everything placed on the main page, and instead placed everything under categories that feel intuitive and easy to access.

## Definitions, Acronyms, and Abbreviations

**User**: any person who uses the system.

**Admin**: administrator of the system, can add and remove cars.

**RAD:** Requirement Analysis Document.

**GUI:** A GUI or graphical user interface is a form of user interface that allows users to interact with electronic devices through a graphical interface.

**JSON:** or JavaScript Object Notation, is a minimal, readable format for structuring data. It is used primarily to transmit data between a server and web application.

## References

Currently there is no system in place to be replaced by the system we are building.

This is v3 of the product. Adding more functionalities to version 1 & 2 proposed in previous sprints.

# Current Software Architecture

As described in the section before, there is no system to be replaced by the one we are building.

Thus, here is **a survey of current architectures for similar systems:**

There are 4 architectures we considered when we looked at similar systems:

* **Layered architectural style**, because it our system is mobile based, this means we will have calls between the application and the web server, which makes it a runtime dependency relationship between our layers. As We will show in our hardware software mapping (section 3.3), we assumed the general case where the client layer and server layer are running on different machines, which will be the normal case, but that does not mean both layers cannot be running on the same machine, which will be the case during testing. For the hierarchical decomposition both open and closed architectures for the layered architectural style can be a suited solution for that we can go with.
* **Repository Architectural Style,** we considered this style because it is used for database management systems and it is well suited for applications with constantly changing, complex data processing tasks, however the central repository can quickly become a bottleneck, both from a performance aspect and a modifiability aspect, plus it doesn’t provide the level of interactivity we want in our system, which is provided by another Architectural Style.
* **Model View Controller** **architecture** (MVC): a special case of the repository where Model implements the central data structure and Control objects dictate the control flow. subsystems are classified into three different types: Model subsystems maintain domain knowledge, do not depend on any view or controller subsystem 🡪Entity objects. View subsystems display it to the user 🡪Boundary objects. Controller subsystems manage the sequence of interactions with the user 🡪Control objects.

MVC is well suited for interactive systems, especially when multiple views of the same model are needed. MVC can be used for maintaining consistency across distributed data, it does however introduce the same performance bottleneck as for other repository styles, but with the advantage of providing interactivity.

* **Event-driven architecture**  enables minimal coupling, which makes it a good option for modern, distributed application architectures. Event-driven architecture is made up of event producers and event consumers. An event producer detects or senses an event and represents the event as a message.   
  It does not know the consumer of the event, or the outcome of an event.  
  After an event has been detected, it is transmitted from the event producer to the event consumers through event channels, where an event processing platform processes the event asynchronously.  
  Events are captured as they occur from event sources such as Internet of Things (IoT) devices, applications, and networks, allowing event producers and event consumers to share status and response information in real time.

# Proposed Software Architecture

Our system architecture can be classified as an **Event-driven architecture** architecture, since it’s the most used architecture for mobile applications.

During Sprint 2 this architecture proved helpful in developing the necessary functionalities needed for the project.

## Overview

The chosen architecture for this system allows for sharing status and response information in real time, which will make the application/system more interactive for a better user experience.

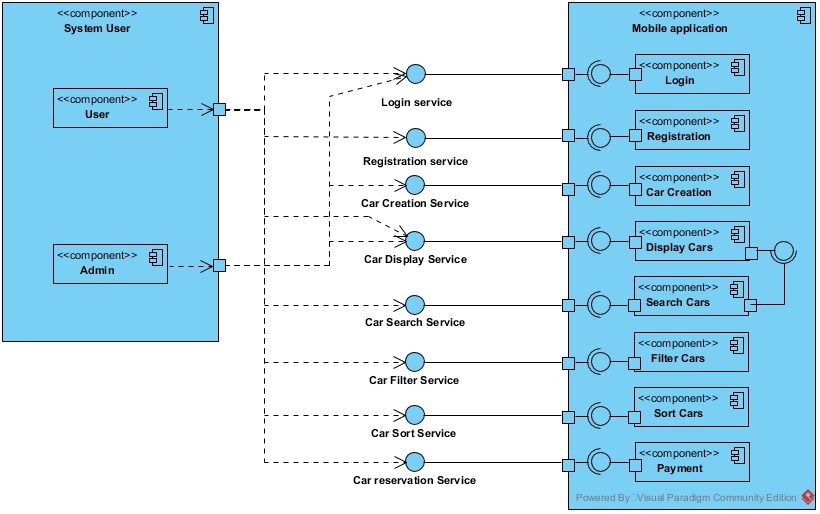
With login subsystem, system users can access the system to get access to the functionalities of the application, the same subsystem provides logout functionality for the users.

The registration subsystem lets the users create an account that they can use to login into the system.

## System Decomposition

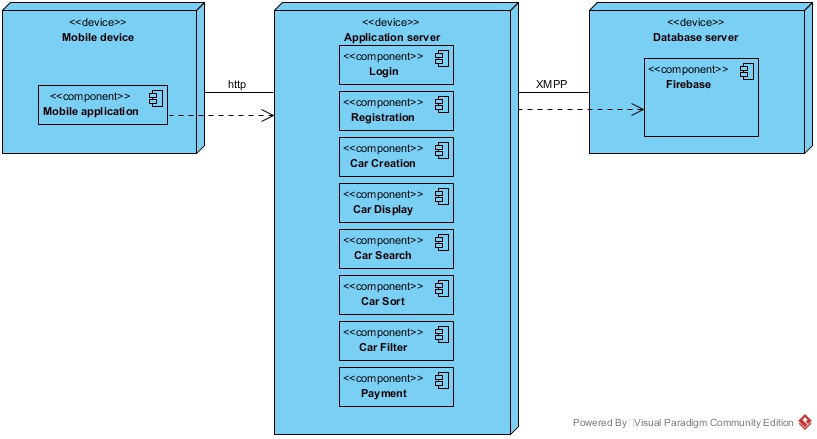
In our System we deal with the complexity of the project by dividing our system into subsystems like login, registration services.

Also, with our selected system architecture, our system can handle the requests easier because every subsystem has their own controller systems. This lets us make the project more organized, maintainable, and readable because the system is decomposed into smaller pieces.



## Hardware Software Mapping

Our system is a mobile application solution and not on-premises solution. Our system makes it possible to rent a car from anywhere you have internet access. Mobile application solutions offer sleek and easy to learn interfaces that allow rental agencies admins to focus on putting the user first instead of struggling with overbearingly complex, and sometimes redundant, software. The user will be able to access to the system, in its full capacity, from anywhere the user has access to a mobile device. The user will always have the latest version.



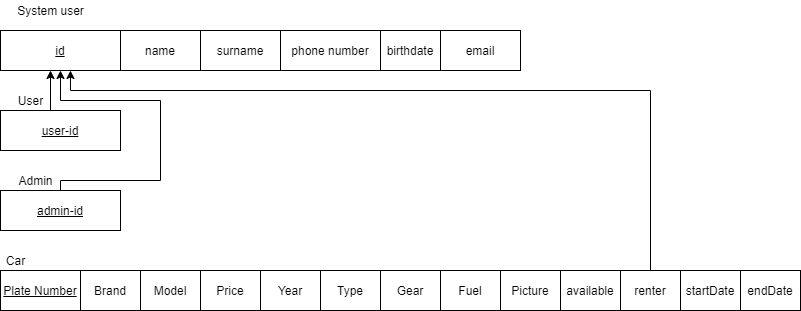
## Persistent Data Management

Persistent data:

* User information: name, surname, phone number, birthdate, email.
* Car information : Plate number, Brand, model, Price, Year, Type, Gear, Fuel type, Picture, availability, current renter, start date, end date.

Data management infrastructure:

The system stores the persistent data related to the users as JSON objects in the firebase database.



## Access Control and Security

Describe the user model of the system in terms of an access matrix. This section also describes security issues, such as the selection of an authentication mechanism, the use of encryption, and the management of keys.

**3.5.1 Access Control**

* Admin has access to all aspects of the system.
* User has access to renting car functionality and display.

|  |  |  |
| --- | --- | --- |
|  | Admin | User |
| Register | No | Yes |
| Login | Yes | Yes |
| Logout | Yes | Yes |
| Create Car | Yes | No |
| Display Car | Yes | Yes |
| Search Car | Yes | Yes |
| Filter Car | Yes | Yes |
| Sort Car | Yes | Yes |
| Car reservation | No | Yes |

**3.5.2 Security:**

The system will not store password as text in our database, it will instead store a hashed version of the password to prevent exposing sensitive data in case of a breach, because storing passwords in text format will make it easy for anyone who gets access to the database all passwords, hashing is a security measure. We are using **bcypt** algorithm to manage the passwords and hashing them, hashing is the process of converting a given password into another value. A hash function (in this case **bcrypt**) is used to generate the new value according to a mathematical algorithm.

Example:

hashUsing**bcypt(**password123)= *$2y$12$vSKoOf8DJElTy4XHXd3U6eosrkiY5mW24UUs8RADAuL6EuUVokqOu*

*we are currently looking into other encryption methods more reliable for mobile apps. We will take a deeper look at this problem in the sprint related to security.*

## Global Software Control

Our System is event driven. Our system should fulfill many users’-initiated requests at the same time. To deal with this situation our system uses event handlers. With these event handlers we can monitor the movement of the user, such as clicking or pressing a key. In other words, our system provides the users the ability to interact with the system at the same time and with different events. When a user interacts with our system the request will be sent only to the relevant subsystem. So other subsystems will not be busy, non-relevant subsystem will just ignore the request and will wait for another request from another or the same user.

For synchronization, the system will synchronize based on three factors: Startup, Authentication and Scheduling. On system startup the users will get a synchronized version of the page; same thing for users that just completed a successful authentication(login), and finally, a scheduled synchronization will be triggered by the system every few minutes (defined by the system admins).

## Boundary Conditions

*Start-up:*

The data that needs to be accessed at startup time is database connection information.

The services that must be registered are domain name service (DNS). In our case we will be using a local emulated machine for testing.

The user interface will show the login page at startup time.

*Shutdown:*

No subsystem can terminate while the rest of the system is running unless it has no effect on the rest of the tunning subsystems.

Only in the case of complete system shutdown are all the subsystems allowed to shut down.

Other subsystems are notified in case one of the other subsystems terminates, if that subsystem was necessary for one of the running subsystems, they will notify the user of the current subsystem that is down and give meaningful feedback.

The updates are communicated to the database in real time, using the agreed-on protocols.

In case of intentional shut down, the users will be notified with a message.

*Error behavior:*

In the case of a communication link fail between server and mobile application, the system gives feedback to the user in the form of a dialog box. Or the system redirects the user to the previous page to the one they are working on or shows an error page depending on the kind of error.

In case of failure (unintentional shut down) all the database information is automatically saved to our local machines(periodically), and upon restarting the system, the information is gathered from the server automatically and the failed communication link is reestablished.

# Subsystem Services

**Log in:**

Provides the login service for all the users of the system, and acts as a gateway to access the other subsystems by giving authentication to the users.

This component is used by **user** component and **admin** component.

**Registration:**

Provides the registration service for all the users of the system, and acts as a gateway to access the **login** subsystem.

This component is used by **user** component.

**Car creation:**

Provides car creation service for admins only, acts as the mechanism for adding new cars into database.

This component is used by **admins only.**

**Car display:**

Provides car display service for all users, this component acts in two ways: display new created cars for the admins and display available cars for the users.

This component is used by **user** component and **Search** component.

**Car search:**

Provides car search service for all users, acts as a mechanism to search for cars with specific criteria. This component uses **Display** Component to display the results of the search.

This component is used by **user** component.

**Car Filter:**

Provides car filter service for all users, this component relies on the results of **search** component to narrow down the number of results according to criteria given by the user.

This component is used by **user** component.

**Car Sort:**

Provides car sort service for all users, this component relies on the results of **search** component to sort the result of the search according to criteria given by the user.

This component is used by **user** component.

**Car reservation:**

Provides car reservation to all end users, this component relies on **login** and **security** component.

This component is used by **user** component.

# References

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.
2. Ian Sommerville (2015). Software Engineering, Pearson tenth edition.
3. <http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/ooa/requirements/IdentifyingURPS.htm>