<Car Service Appointment Application>

Version <2.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <05/06/20> | <2.0> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

2. Non-functional Requirements 4

2.1 Availability 4

2.2 Performance 4

2.3 Security 4

2.4 Testability 4

2.5 Usability 4

3. Design Constraints 4

# Introduction

This document capture the system requirements such as non functional requirements as well as design constraints.

# Non-functional Requirements

## Availability

* Availability is expressed as the ratio of the available system time to the total working time
* Source of stimulus: existence of the system
* Stimulus: software update
* Environment: fully working system
* Artifact: the whole application
* Response: updating the application
* Response measure: the time it takes for the update to be completed
* Make updates efficient by updating smaller parts of the software application

## Performance

* Performance shows the response of the system when performing certain actions for a certain period of time
* Source of stimulus: user
* Stimulus: searching for an appointment
* Environment: the appointments are currently displayed in a certain order
* Artifact: the appointment module
* Response: refreshing the page by displaying only the items that match the search
* Response measure: the time in which the response is given
* Finding a good algorithm for performing the search

## Security

* Security is responsible for reducing the likelihood of malicious or accidental actions, theft or loss of information
* Source of stimulus: malicious user
* Stimulus: attempting to access private information without being authenticated
* Environment: the application is in normal functioning mode
* Artifact: the security/data validation layer
* Response: prevent user from having access to private information
* Response measure: the consistency of the blocking measures in response to different attacking methods
* Password encryption, encapsulation

## Testability

* Testability shows how the system performs according to certain criteria
* Source of stimulus: user
* Stimulus: entering wrong data formats
* Environment: the application is in normal functioning mode
* Artifact: unit tests
* Response: validate data and respond accordingly
* Response measure: the wideness of cases covered by the tests
* Creating tests for as many as possible use case scenarios

## Usability

* Usability is defined by the efficiency of the interaction between the user and the system
* Source of stimulus: User
* Stimulus: finding the register page
* Environment: the application is in normal functioning mode
* Artifact: UI
* Response: page searched is rendered
* Response measure: the amount of actions needed in order to reach the goal (i.e. number of clicks)
* Making every action accessible within at most 2 clicks

# Design Constraints

[This section needs to indicate any design constraints on the system being built. Design constraints represent design decisions that have been mandated and must be adhered to. Examples include software languages, software process requirements, prescribed use of developmental tools, architectural and design constraints, purchased components, class libraries, and so on.]

The development tools chosen for this application initially were the Django framework (using Python) and JavaScript for building the UI. The disadvantages of being constrained to basing an event management problem to an MVT design pattern later changed the decision to using a layered client-server architecture. The backend runs on .NET Core being implemented with Entity Framework core, storing the data using SQL Server.

The advantages of using C# can be seen specially when building the data access layer by making use of Linq. Linq is a set of technologies with query capabilities integrated directly into C#. Query expressions are written in a declarative query syntax. By using query syntax, you can perform filtering, ordering, and grouping operations on data sources with a minimum of code.

Another significant library used is JWT (JSON Web Tokens) for generating the token necessary for performing the user authentication part. JSON Web Token (JWT) is an open standard ([RFC 7519](https://tools.ietf.org/html/rfc7519)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

The client side is built with Angular 9. The advantages of angular are many but include cross platform capabilities, being compatible with different web browsers but also with different operating systems.

Regarding speed and performance, angular apps load quickly thanks to the Router component, which delivers code splitting, such that users only load code required for the view they requested

This application was built with the help of Angular CLI, which offers command line tools for fast building and fast component/test generation.

The application follows the REST API principles :

* **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
* **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
* **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
* **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

Source: <https://restfulapi.net/>