

# **Smart Garage**

# Car Service Management System

# Project Description

**Smart Garage** is a web application tailored for auto repair shop owners to efficiently handle day-to-day operations. Users can use the system as employees or customers.

For new customers, an administrator generates a profile within the system, and login credentials are sent to the customer's email. The profile includes a username (customer's email address) and a randomly generated password. Upon email delivery, customers gain access to their personal information through the web UI.

Customers must be able to view a comprehensive list of cars, including manufacturer and model, along with a list of available services such as oil change and filter replacement, each with its associated price. The system maintains a customer list with contact details (name, phone, email, etc.)

Employees must be able to link specific cars to customers.

Each car must be uniquely identified by a registration plate with a vehicle identification number.

**Smart Garage** must keep a history of all services performed on customers' cars. The application must be able to provide detailed reports for each visit, showing all services performed and the total cost, in the default currency or an alternative one, chosen by the employee or customer. Additionally, the software could generate a PDF report summarizing all services for a specific car and the corresponding total price.

# Functional Requirements

## Entities

* Each **user** must have a username, password, email, phone number.
* The username must be unique and have between 2 and 20 symbols.
* Password must contain at least 8 symbols and should contain a capital letter, digit, and special symbol (+, -, \*, ^, etc.)
* Email must be a valid email and unique in the system.
* Phone number must be 10 digits and unique in the system.
* Each **vehicle** must have a license plate, VIN, year of creation, model, and brand.
* The license plate must be a valid [Bulgarian license plate](https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Bulgaria#:~:text=Standard%20Bulgarian%20vehicle%20registration%20plates,it%2C%20the%20country%20code%20for).
* The vehicle identification number must be 17 characters long string.
* The year of creation must be a positive whole number larger than 1886.
* The model and brand must be between 2 and 50 symbols.
* Each **service** must have a price.
* The price must be a non-negative number.

## Public Part

The public part must be accessible without authentication for anonymous users.

The public part provides the Login form (must) used to authenticate a user using username and password.

There must be an option to invoke the “forgotten password” option.

## Private part

Accessible only if the user is authenticated.

Customers must be able to see the list of all their services.

* There must be a way to filter them by vehicle *or*/*and* date.
* There must be a way to see a detailed report of a visit to the shop.
  + The report must include the customer info, the vehicle info, every service that was done to the vehicle and the total price. The customer must be able to choose the currency of the report. (Use a third-party service to convert the price).
  + There could be a way to generate a PDF report for a given visit to the shop.
  + There could be a way to generate one report that includes multiple visits.
* There could be an indicator of the status of the service (not started, in progress, ready for pickup…).

There should be an option to change the password.

## Administrative part

Accessible to employees only.

There must be a section in the application, dedicated to the **vehicles**:

* Employees must be able to browse or update all vehicles linked to customers.
* Employees must be able to filter the vehicles by owner.
* Employees must be able to create a new vehicle for a customer.
  + If the model/make does not exist in the application’s database, it must be created.

There must be a section in the application, dedicated to the **services** that the shop offers:

* Employees must be able to browse, create, delete, or update a service.
* Employees must be able to filter the services by name *or*/*and* price.

There must be a section in the application, dedicated to **customers**.

* Employees must be able to browse, delete, or update a customer’s profile.
* Employees must be able to filter the customers by name, email, phone number, vehicle (model or make) **or** visits in range (a visit between two dates).
* Employees must be able to filter the customers by name, email, phone number, vehicle (model or make) **and** visits in range (a visit between two dates).
* Employees must be able to sort the customers by name **or** visit date (latest or oldest on top).
* Employees must be able to add a new visit to the shop by a customer who does not have a profile yet.
  + The visit is registered and then the customer receives an email with automatically generated login information.
* Employees must be able to see a detailed report of a visit to the shop.
  + The report must include the customer info, the vehicle info, every service that was done to the vehicle and the total price. The employee must be able to choose the currency of the generated report (Use a third-party service to convert the price).
* Employees should be able to send a “reset password” link to a customer via email.
* Employees could be able to generate a PDF report for a given service. The report can be downloaded or sent to the customer via email.

## Other Optional features

**Make an Enquiry** – the customer fills in the online form with first & last name, email address, postcode, license plate and free text with an issue explanation. A service receptionist receives the request and returns a response with approximate repair time and price. In the admin panel, the user request and the receptionist’s response are displayed together with the time of creation of both.

**Remote Visit Repair** – A customer can request remote visit repair by filling in an online form with their first and last name, email address, location, license plate, and free text with an issue explanation. The service has a single team that responds to remote visit repairs. If the team is free, the customer receives an offer. The offer must contain the estimated time of arrival of the service team. The time of arrival depends on the location and must be calculated with an external API (see Appendix for more information).

**Payment Methods** – the customer or employee can choose between cash, bank, or credit card payment when he makes a car visit. If the payment is via bank transfer, he receives an email with a proforma invoice after the visit is closed. If the payment is via credit card, the customer receives an email with a payment link.

**Apply to Mechanic Job** – A user is required to register before he can apply for a job. After filling in the online form (First, Last name, email, mobile phone and postcode), the user receives an email with his application. The HR team receives the application. In their admin panel they can see each application – each one can have the following status (new, denial, job interview, cancellation, job offer, hired). At each step, the HR team fills in an online form to be sent to the applicant.

**Visit Rating –** each visit has a repair history which includes the mechanic who worked on the car. Only the user who owns the car can leave a review with a comment.

**Loyalty Program Discount** – loyal customers who have made certain visits receive an automatic discount. There are three levels of loyalty – 9 visits (car enthusiast), 18 visits (car fan), 27 visits (car fanatic). Their loyalty status can be visible in the customer profile.

**Spare Parts** – when making a visit the customer can choose whether he would like the original or other spare parts. After selecting spare parts depending on the brand and model of the car approximate repair time is displayed to the customer.

**Service Visit Calendar** – the service cannot have more than 6 visits per day. If you try to make more, a suggestion for the next available date appears.

**All the models!** - you can consume another programmer’s API to get enough car models and brands for your application.  
**Easter eggs** – Creativity is always welcome and appreciated. Find a way to add something fun and/or interesting, adding an Easter egg or two to your project will add some variety.

## REST API

To provide other developers with your service, you need to develop a REST API. It should leverage HTTP as a transport protocol and clear text JSON for the request and response payloads.

A great API is nothing without great documentation. The documentation holds the information required to successfully consume and integrate with an API. You must use [Swagger](https://swagger.io/) to document yours.

The REST API provides the following capabilities:

1. Users

* CRUD Operations (must)
* Employees must be able to search a user by username, email, phone, or car’s license plate/VIN (must)

1. Vehicles

* CRUD Operations (must)
* Search a vehicle by license plate or VIN (must)
* Employee must be able to search vehicles by owner’s phone number (must)
* Filter and sort vehicles by model, brand, year of creation (must)

1. Services

* CRUD Operations (must)
* Search by name (must)
* Search by price range (must)
* Filter and sort by name and price (must)

# Use Cases

## A walk-in customer

A new customer walks into the shop. He has never visited before. He explains what the issue with his vehicle is and an employee creates a new vehicle registration (it includes the brand, model, year, VIN, and license plate). When the registration is created the employee creates a new service order that includes the newly created vehicle, as well as all the service procedures that need to be performed (oil change, brake fluid change, etc.). The customer receives in the email that he has provided an automatically generated invoice, containing the total sum that he must pay along with every service procedure that will be performed on his vehicle. He receives another email, that contains information about his automatically generated login information.

## Forgotten password

A customer has forgotten their password. They select the “Forgot password?” option and enter their email. The system checks if there is a registered user with that email. If there is one, he receives an email with a link to a page where he can enter a new password. The link should be accessible for a limited time only (say, an hour) and if accessed once, it should not be possible to access it again.

## Report request

*Note: Applicable only if PDF report generation is implemented.*

A customer lost the copy of a report that you sent him after a visit to the shop a year ago. He logs in to the application, filters the list of visits by date and vehicle and goes on to generate a PDF report. He wants it to be in USD, instead of the default currency, say BGN. He receives the report on his email with the correct price, converted to USD.

# Technical Requirements

## General

* Follow [OOP](https://en.wikipedia.org/wiki/Object-oriented_programming) principles when coding
* Follow [KISS](https://en.wikipedia.org/wiki/KISS_principle), [SOLID](https://en.wikipedia.org/wiki/SOLID), [DRY](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) principles when coding
* Follow REST API design [best practices](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy) when designing the REST API (see Appendix)
* Use tiered project structure (separate the application in layers)
* The service layer (i.e., "business" functionality) must have at least 80% unit test code coverage
* Follow [BDD](https://en.wikipedia.org/wiki/Behavior-driven_development) when writing unit tests
* You should implement proper exception handling and propagation
* Try to think ahead. When developing something, think – “How hard would it be to change/modify this later?”

## Database

The data of the application must be stored in a relational database. You need to identify the core domain objects and model their relationships accordingly. Database structure should avoid data duplication and empty data (normalize your database).

## Git

Commits in the GitHub repository should give a good overview of how the project was developed, which features were created first and the people who contributed. Contributions from all team members must be evident through the git commit history! The repository must contain the complete application source code and any scripts (database scripts, for example).

Provide a link to a GitHub repository with the following information in the README.md file:

* + Project description
  + Link to the Swagger documentation (must)
  + Instructions how to setup and run the project locally
* Images of the database relations (must)

## Optional Requirements

Besides all requirements marked as should and could, here are some more *optional* requirements:

* Integrate your project with a Continuous Integration server (e.g., GitLab’s own) and configure your unit tests to run on each commit to your master branch
* Host your application's backend in a public hosting provider of your choice (e.g., AWS, Azure, Heroku)
* Use branches while working with Git

# Teamwork Guidelines

Please see the Teamwork Guidelines document.

# Appendix

* + [Guidelines for designing good REST API](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy)
  + [Guidelines for URL encoding](http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/)
  + [Git commits - an effective style guide](https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn)
  + How to Write a Git Commit Message

## Microsoft Bing Maps External Service

We will use this external service to help us calculate the travel duration between two locations. (if you decide to implement the Remote Visit Repair)

Note: We’re using this service because it’s free. It doesn’t require any payment information to be used, the registration process is straight forward as well as the generation and usage of API Key.

Because of the service limitations, for addresses outside of the US, we can only use travelMode=driving. We can’t use walking or transit. The point is to get familiar with consuming a REST service, understanding its domain and do some data transformations.

## API Key

Each team needs to register at least one account and get a free API key, which will allow them to make HTTP calls to the REST service. Each API key is limited in the number of requests etc. so it’s not advisable to share your API key with another team as it may lead to locking or completely disabling the API key from Bing Maps. The API key is passed to every HTTP request in “key” query parameter.

## Integration Guidelines

### Locations

The first API endpoint that needs to be called is “Locations” and we will use it to find a Location by address. (official [docs](https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address)). We will use the structured URL form, which specifies the location query parameters as part of the URL path:

[*http://dev.virtualearth.net/REST/v1/Locations/{countryRegion}/{adminDistrict}/{postalCode}/{locality}/{addressLine}?key=<YOUR\_API\_KEY*](http://dev.virtualearth.net/REST/v1/Locations/{countryRegion}/{adminDistrict}/{postalCode}/{locality}/{addressLine}?key=<YOUR_API_KEY)*>*

For example, the structured URL query for Telerik Academy’s Location will look like:

*http://dev.virtualearth.net/REST/v1/Locations/BG/Sofia%*[*20City/Mladost/1729/Alexandar%20Malinov%2031?key*](http://dev.virtualearth.net/REST/v1/Locations/BG/Sofia/Mladost/1729/Alexandar%20Malinov%2031?key)[*=*](http://dev.virtualearth.net/REST/v1/Locations/BG/Sofia/Mladost/1729/Alexandar%20Malinov%2031?key=AhqpRyyyhbWVwrrxOVnvjKJuIfcPRcBWe2YS-mL4BXAVXuUH3lfx_QilDP50vOJO)*<YOUR\_API\_KEY>*

The result contains a collection of “geocodePoints”. If there’s more than one Geo Point returned, we want the one of usageType “route”.

Note: You can use the lat/long values returned as a result in any mapping service (e.g. Google Maps) just to test if your results are correct.

### Distance Matrix

The next API endpoint that we will use is the Route’s API distance matrix. which can calculate the time to travel from point A (lat, long) to point B (lat, long) for us. [Official docs](https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix). For example, the URL to calculate the distance between The National Place of Culture and Telerik Academy will look like:

[*https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?origins=42.685428619384766,23.318979263305664&destinations=42.6508241,23.3790428&travelMode=driving&key=<YOUR\_API\_KEY*](https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?origins=42.685428619384766,23.318979263305664&destinations=42.6508241,23.3790428&travelMode=driving&key=<YOUR_API_KEY)*>*

How to make a request to an external API?

Use [HttpClient](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/http-requests?view=aspnetcore-6.0) or [RestSharp](https://restsharp.dev/) (easier maybe) that sends a request and receives a response.

* Construct the request URI with a "moving" part - the search string
* The response will be in JSON format, study the properties and map the json object to a C# object, so you can work easily with the properties (title, pageid, etc.)

# Legend

* Must – Implement these first.
* Should – if you have time left, try to implement these.
* Could – give these a go only if you are ready with everything else.