**Car service**

**Description:**

The Car Service system serves primarily for drivers, but also for the employees and Car Service administration. In total there are 3 different type of users. Every type has own access permissions accordingly with hierarchy Client – Employee – Administrator, where the last has the most permissions.

Service provides two different systems for clients: detail shop and service reservation system. First system is the system where client can order details for their cars. Clients can track order information. Clients will have a notification when detail is delivered. Second system is a reservation system for any car service that client need: wheel repair, window reinstallation, painting etc. Clients can track service status too. Also, they can see which employee provide service to their car, start and end of working process.

For employees system provides regular service (for clients) as well as special employees’ service with higher level of access rights. Besides detail order and service reservation, employees (mechanics in our case) can choose the work they will do, specify their hours of work and change order status. Users with administration permissions can manage accounts and access level for other users.

System restrictions are based on pretty common things as “There can’t be two details/categories/services with same name”, “Only employees and administration can manage details/categories/services” etc.

**Functional requests:**

* F1: There are 3 different type of users: Admin, Employee and Client.
* F2: All the user types can order details from service shop.
* F3: All the user types can order reserve service for their car.
* F4: All the user types can track their orders and service status. For example: *ORDERED*, *ON THE WAY* or *DELIVERED*.
* F5: All the user types should have a notification when detail is delivered or when the service is provided.
* F6: Employees can choose the work they want to do themselves.
* F7: Admins can manage accounts.
* F8: Admins can change access level of any user.

**How to use:**

* Run MySQL server
* Create table “autoservice” on your MySQL server
* Run the application in compiler (*for ex. IntelliJ IDEA*)
* Controllers tests are in special postman file. Import them to the postman.
* For request sending in most cases you need to put token, which you’ll get after login to headers, with “Bearer “ *(space included)* before token.

**Conclusion:**

When we planned our project, we had no idea that it would be so laborious. The main part of our work was the correct implementation of the "Service" and "Controller" classes.

We used "Date Transfer Object" type of classes, as we consider this to be the most convenient way to convert information in queries for the further use in the program. Due to the large number of small, but insanely annoying problems, we have quite well understood the basics of Spring Booth and Spring Security, and we also realized the benefits of using "Layered architecture".

As were mentioned above, the main problems were waiting for us at the Spring level. There have been many problems with authorization, translation of the JSON format to JAVA format (for example LocalDateTime), problems with the database (invalid default value in database) and a huge number of small flaws in the development of the application.

Despite these difficulties, we consider the project to be successfully completed. We have gained a lot of experience in the programming areas that this project affects, especially after solving all the errors.

Developers: **Miron Grishchenko**, **Nazar Grigorenko**.