VERICU'
TUNING

Service “La Vericu”

a JavaFX desktop App

“ia zi vărule, ce ii facem?”

Table of contents

----

1. Application summary

“Vericu Service and tuning” is a desktop application that eases communication between customers and the mechanics of a car service. The application has two types of users: customers and mechanics. The application allows customers to see the status of their car’s visit to the shop, as well as contact information of the service. Respectively, the mechanics can add a car by introducing the basic information of a car and after that he can change the status of the appointment or delete the car from the database. The application has been developed in JavaFX and the database controller PostgreSQL.

2. User guide

When starting the app, the first thing appearing on the screen is the log-in page (fig. 1). Here you can log in or if you don’t have an account, you can press the “Sign Up” button then create an account (fig. 2).

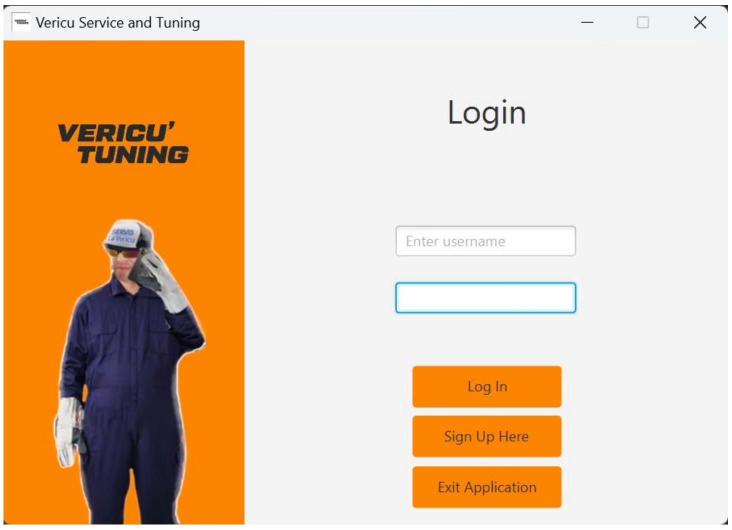


fig. 1

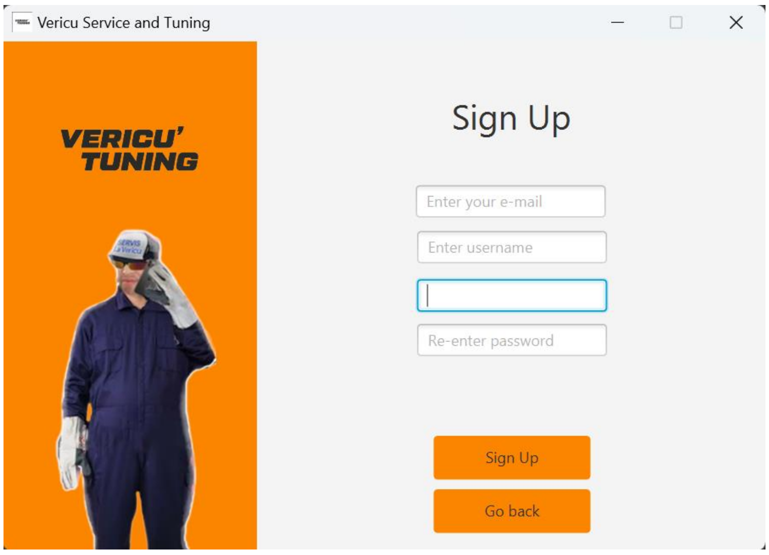
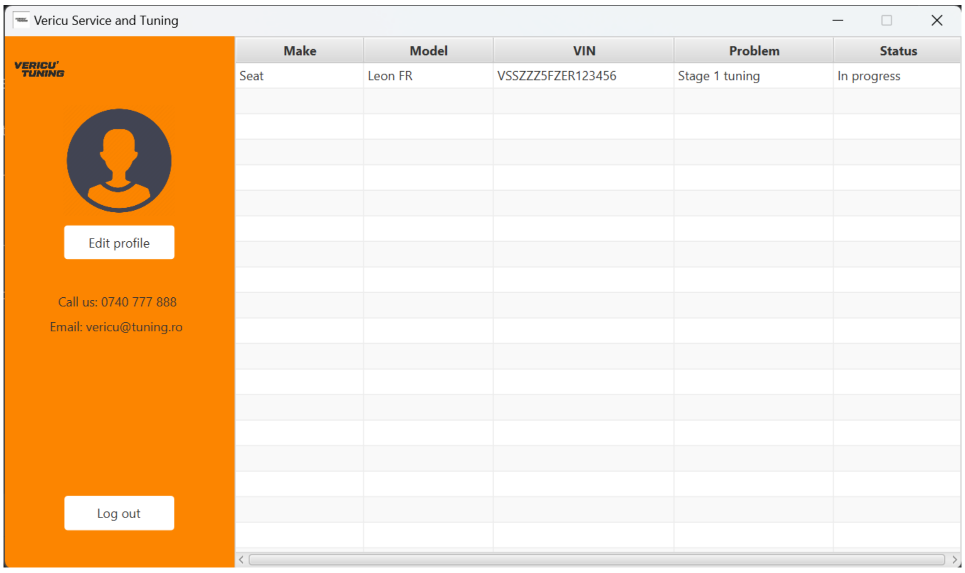


fig. 2

Once you log in, depending on the type of user, one of two interfaces will appear: customer page or mechanic page.

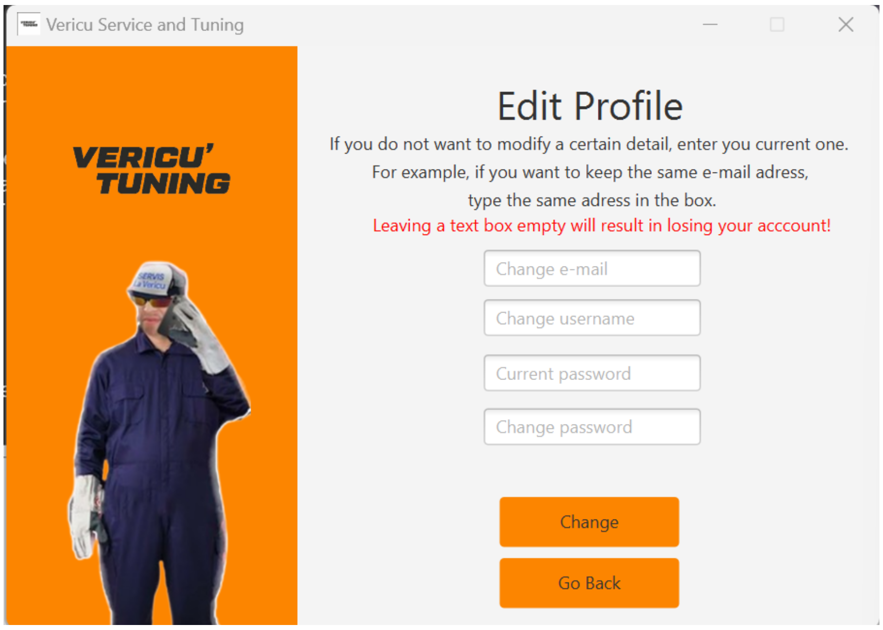
2.1 Customer page


Vericu Service and Tuning

Make	Model	VIN	Problem	Status
Seat	Leon FR	VSSZZZ5FZER123456	Stage 1 tuning	In progress

fig. 3

This is the customer page (fig. 3). Here the customer can see his car's details and the service status. The "Log out" button logs the user out and brings him back to the Log in page. There is also an "Edit profile" button that leads to a new page that lets the user modify their info such as log-in credentials (fig. 4).


Vericu Service and Tuning

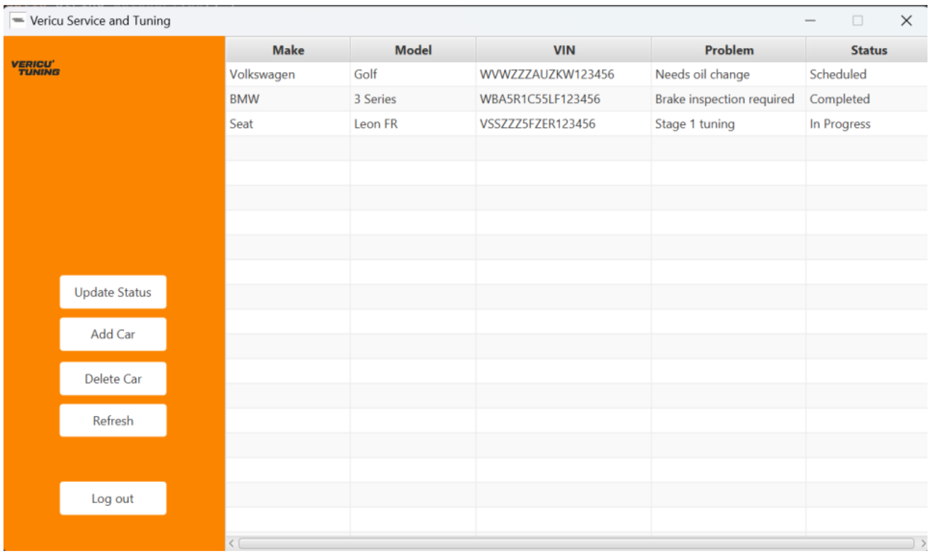
Edit Profile

If you do not want to modify a certain detail, enter you current one.
For example, if you want to keep the same e-mail adress, type the same adress in the box.
Leaving a text box empty will result in losing your acccount!

fig. 4

2.2 Mechanic page

The mechanic page (fig. 5) looks a lot like the customer's page but has more functionality, such as adding a new car, changing the status of the service, refreshing the database of the cars, and finally removing a car from the database, marking a service fully completed.



Make	Model	VIN	Problem	Status
Volkswagen	Golf	WVWZZZAUZKW123456	Needs oil change	Scheduled
BMW	3 Series	WBA5R1C55LF123456	Brake inspection required	Completed
Seat	Leon FR	VSSZZZ5FZER123456	Stage 1 tuning	In Progress

fig. 5

Being logged in as a mechanic, the user can see all the cars currently in use. If the user wants to update the status of a car, he simply clicks on the car that he wants to modify, then clicks on the "Update Status" button, then a pop up with a selection box will appear (fig. 6).

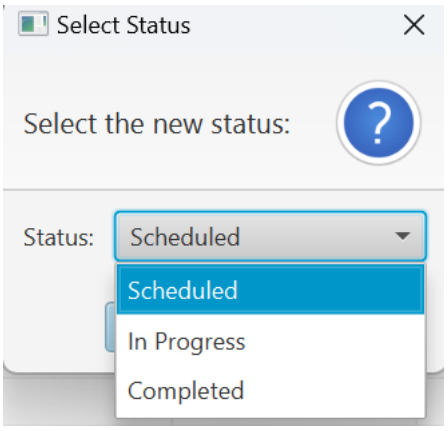


fig. 6

After that, the status will be modified to which selection it was made.

Moving on, the "Add Car" button lets the user add a new car to the database by entering the information about the vehicle as shown in the next picture (fig. 7).

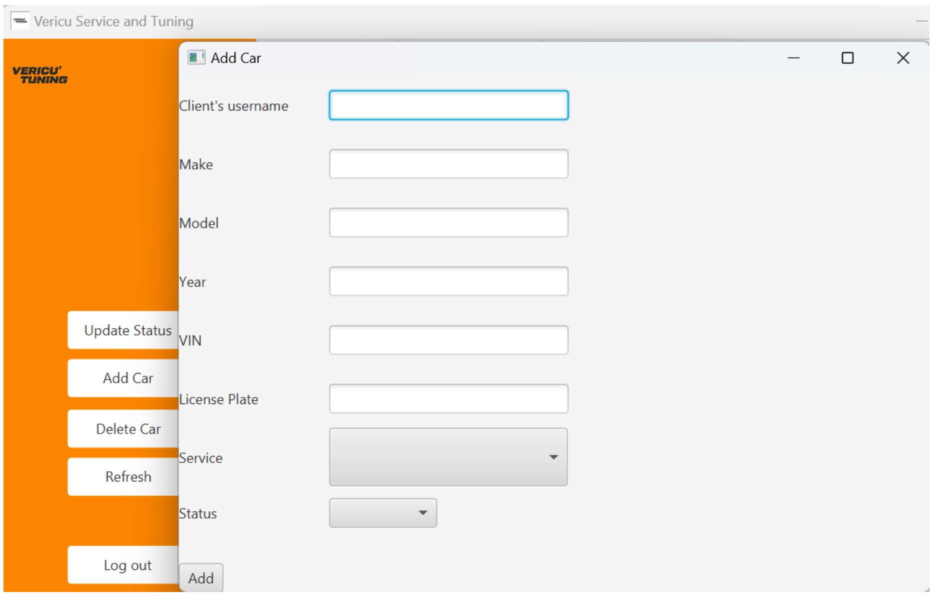


fig. 7

Here, the user must type in the client’s username, the car’s make, model, year of production, VIN, license plate and then select the service and the status needed for the car.

Next, the “Delete Car” button will delete the car from the database, assuming that the car has been collected back by the owner after completion of the service. The “Refresh” button refreshes the table so that the new car added to the database can be seen and modified as needed.

Finally, the “Log out” button logs the user out, bringing him back to the log in page.

3. Database

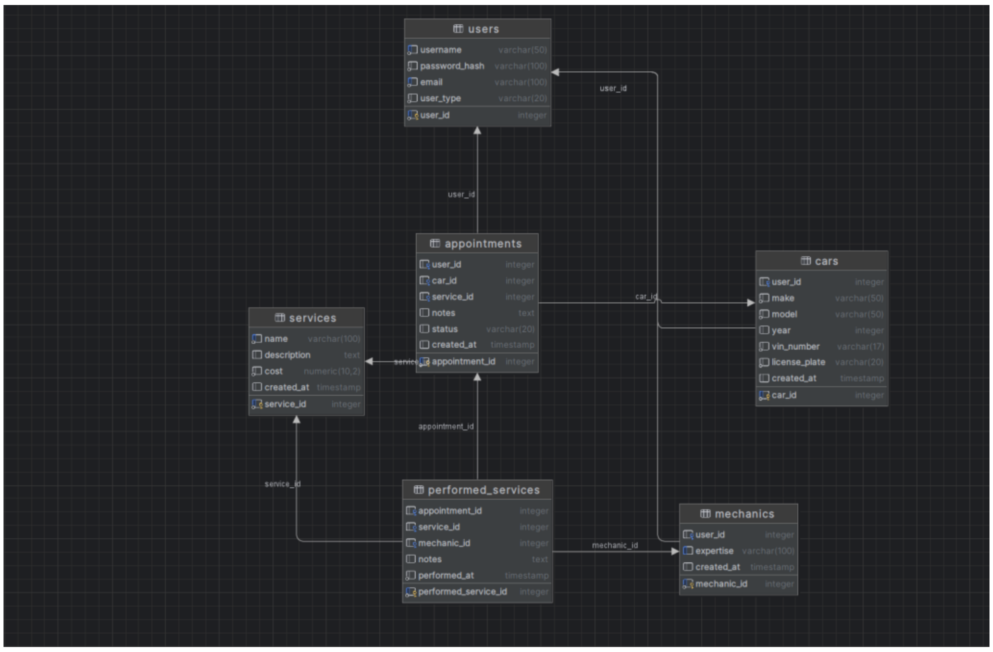


fig. 8

The database (fig. 8) consists of 6 tables: users, cars, mechanics, performed_services, services and appointments. Each table has their unique constraints ranging from the user id to be unique to the VIN being a legitimate and correct VIN number. The database was created using DataGrip and the database manager PostgreSQL.

4. Java programming

As mentioned before, the application has been written in Java 21 coding language. The project features multiple Java Classes and an Enumeration Class, each class being associated with a .fxml file for each UI element.

The connection with the PostgreSQL database was made using JDBC and a connector file specific for PostgreSQL. This database manager has been chosen for accessible and complex functions such as CRUD functions, allowing easy data maneuverability.

The User Interface was made SceneBuilder, which is a front-end program that lets users create JavaFX files with more ease and with a user-friendly interface.