Ministry of Education and Science of Russian Federation

Federal Autonomous Educational Institution of Higher Professional Education

**«NATIONAL RESEARCH TOMSK POLYTECHNIC UNIVERSITY»**

Department Engineering School of Information Technology and Robotics Direction 090404 «Software engineering»

Cathedra Software Engineering

**Personal car repair book history**

**Term project**

Course Participants: \_ M.A. Vasin

(group) (date) (signature)

Scientific Consultant: V.Y. Polishchuk

(date) (signature)

Tomsk – 2019

CONTENTS

[INTRODUCTION 3](#_Toc535993507)

[Working process 4](#_Toc535993508)

[Describing data and creating a database 4](#_Toc535993509)

[Backend implementation 5](#_Toc535993510)

[Frontend implementation 6](#_Toc535993511)

[Results 7](#_Toc535993512)

[CONCLUSION 13](#_Toc535993513)

[Appendix 1 15](#_Toc535993514)

# INTRODUCTION

Data visualization is a general term that describes any effort to help people understand the significance of data by placing it in a visual context. Patterns, trends and correlations that might be undetected in text-based data can be exposed and recognized easier with data visualization software.

In current research the process of visualization for history of repair by car is described. Nowadays people tend to buy more and more cars, and sure, they have to serve their.

This scientific investigation covers the process of collecting data and implementation of a web application. This application is aimed to provide detailed information about how many money were paid for car’s repair.

# Working process

## Describing data and creating a database

To describe data better – describe process of service a car. We bay some components in car shop, after that we repair it self-employed or call to repair service.

On the basis of the data obtained, a database of 7 related tables was created: cars, repairs, repairWorks, persons, components, carShops, repairStantions.

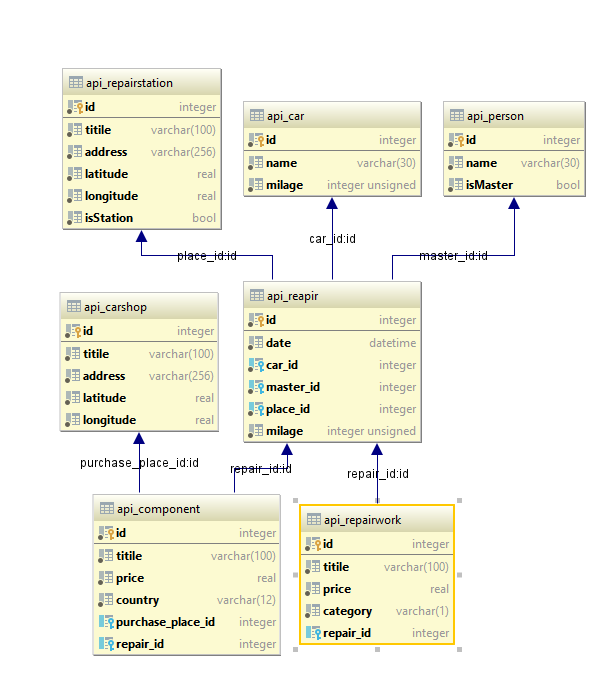


Figure 1 –Schema of implemented database

Creations, populations and all queries are implemented using the django’s ORM with SQLite as database witch locally located in the project.

## Backend implementation

Backend is a Python's WSGI application implemented on the Django REST framework. API is as follows:

Table 1 – Application Programming Interface (*HTTP GET requests only*)

|  |  |
| --- | --- |
| Path | Return |
| /api/cars | Get list of car marks |
| /api/cars/<int:id> | Get detailed information about a specific mark (brand) by id |
| /api/repairs | Get list of repairs |
| /api/repairs/<int:id> | Get detailed information about a specific repair by id |
| /api/repairs/<int:car> | Get list of car repairs |
| /api/works/ | Get list of all repairs |
| /api/works/<int:repair> | Get list of repair’s works |
| /api/works/<int:car> | Get list of all repair’s works for car |
| /api/components | Get list of components |
| /api/components/<int:repair> | Get list of components by repair |

Also there’re some basic end-point of api for other entities, but these don’t use in application.

Models are described at Appendix 1.

## Frontend implementation

To implement the front-end part of the project, a variety of different tools were used. Nodejs and its built-in npm package installer were used to create and develop a frontend project.

The basis of the project is the framework VueJs. Vue.js (commonly referred to as Vue; pronounced /vjuː/, like view) is an open-source JavaScript framework for building user interfaces and single-page applications. Vue.js features an incrementally adoptable architecture that focuses on declarative rendering and component composition. Advanced features required for complex applications such as routing, state management and build tooling are offered via officially maintained supporting libraries and packages.

For the realization of the map, the library Leaflet was used. For other charts – v-line library.

* 1. Integrated Development Environment

For the development of the entire project, the following Integrated Development Environments (IDE) were used: Jetbrains PyCharm professional edition (student license) to create Django project, Jetbrains WebStorm to implement front-end project and DataDrip for continuous access and tracking of the database’s state.

# Results

Figure 4 shows the page with the choice cars:

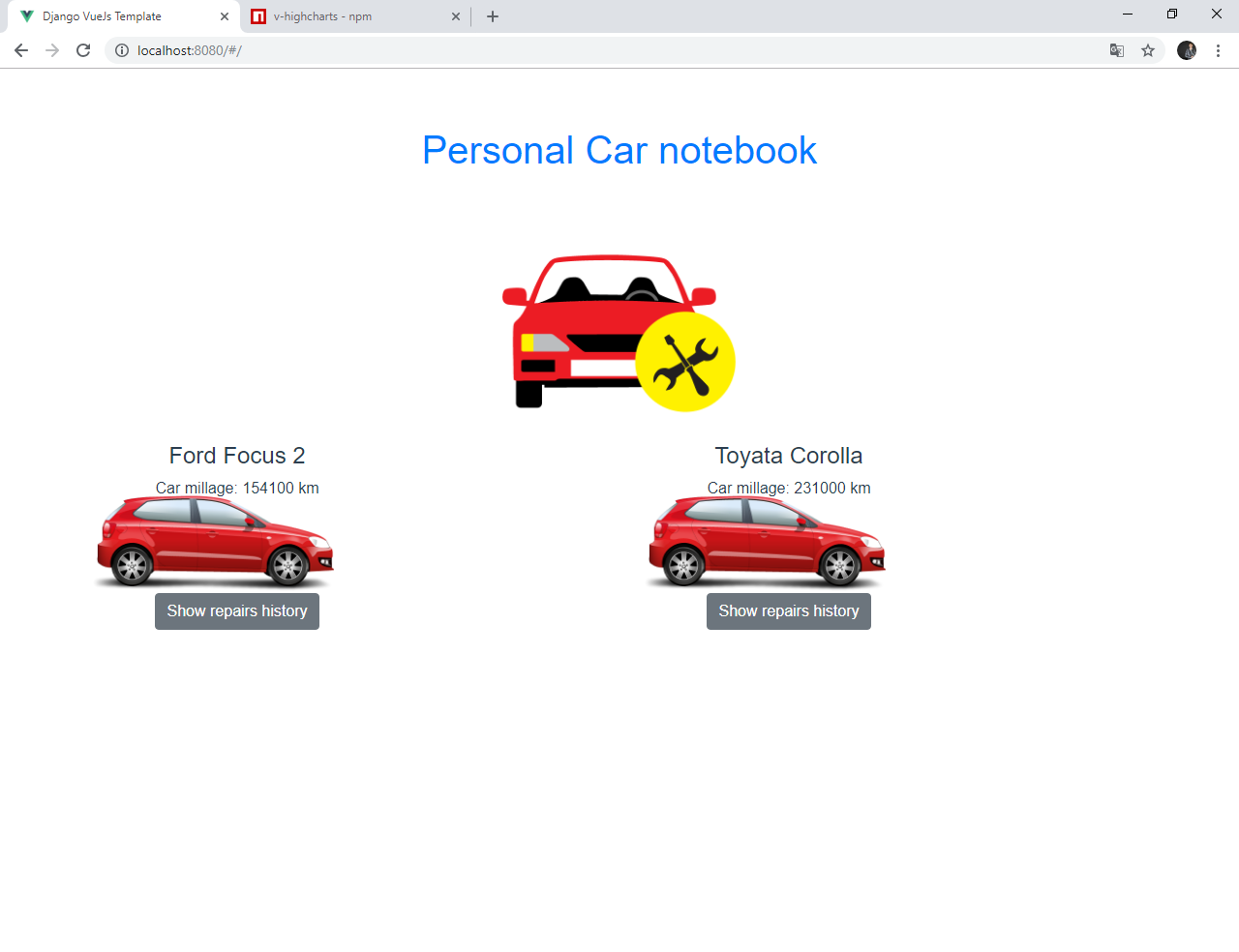


Figure 4 – User’s cars

As you can see – cars are present as cards components and after clicking on any one, a page draws repair list and basic statistics charts. Repair data is presented in tabular form. We can sort the data by values in the columns, and line-series chart with summary costs. Figure 5 shows an example of this page.

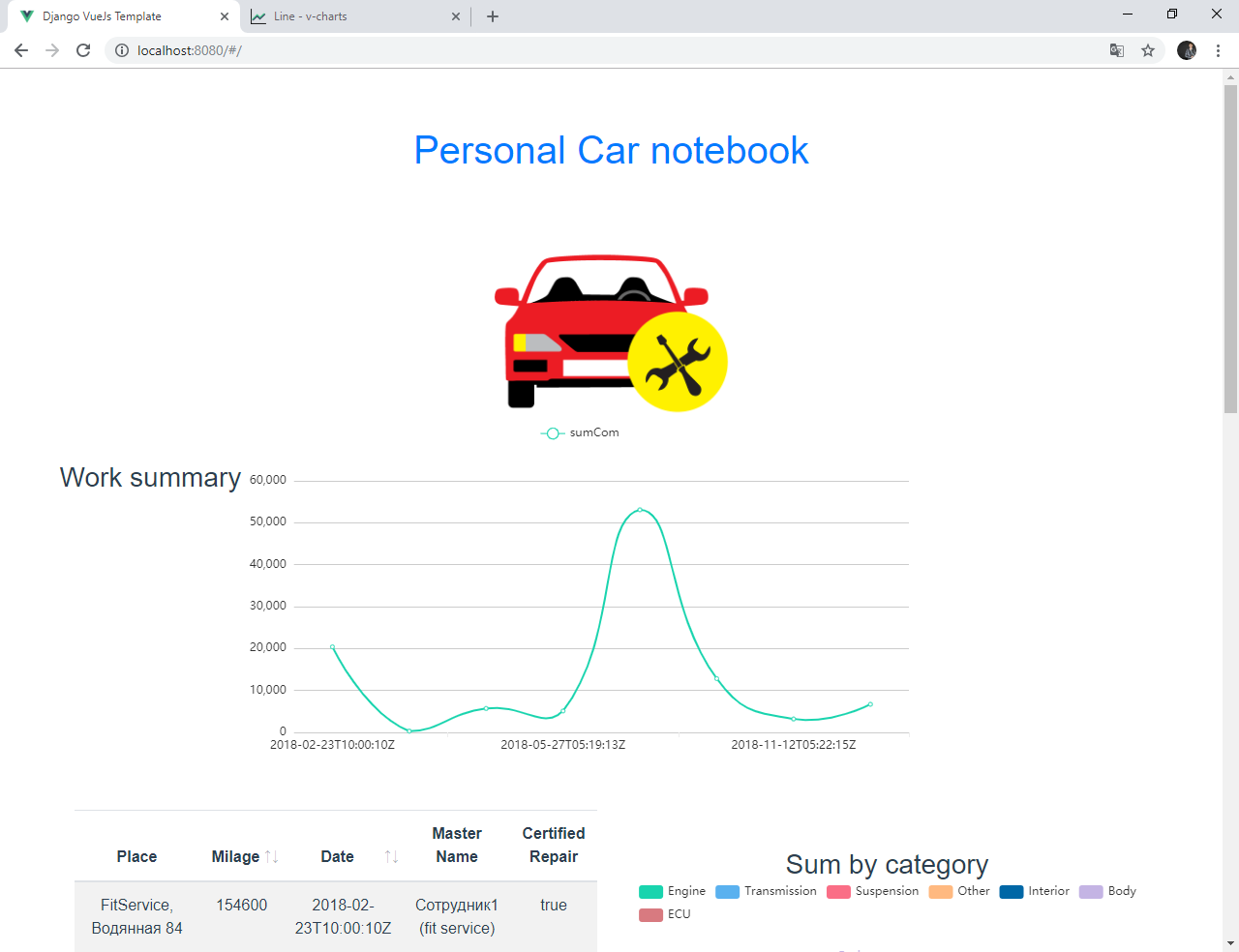


Figure 5 – Repairs page: top part

Also we can see piecharts about numbers of repairs by category (like engine, transmission, etc) by chosen car for all time. Result is present on Figure 6.

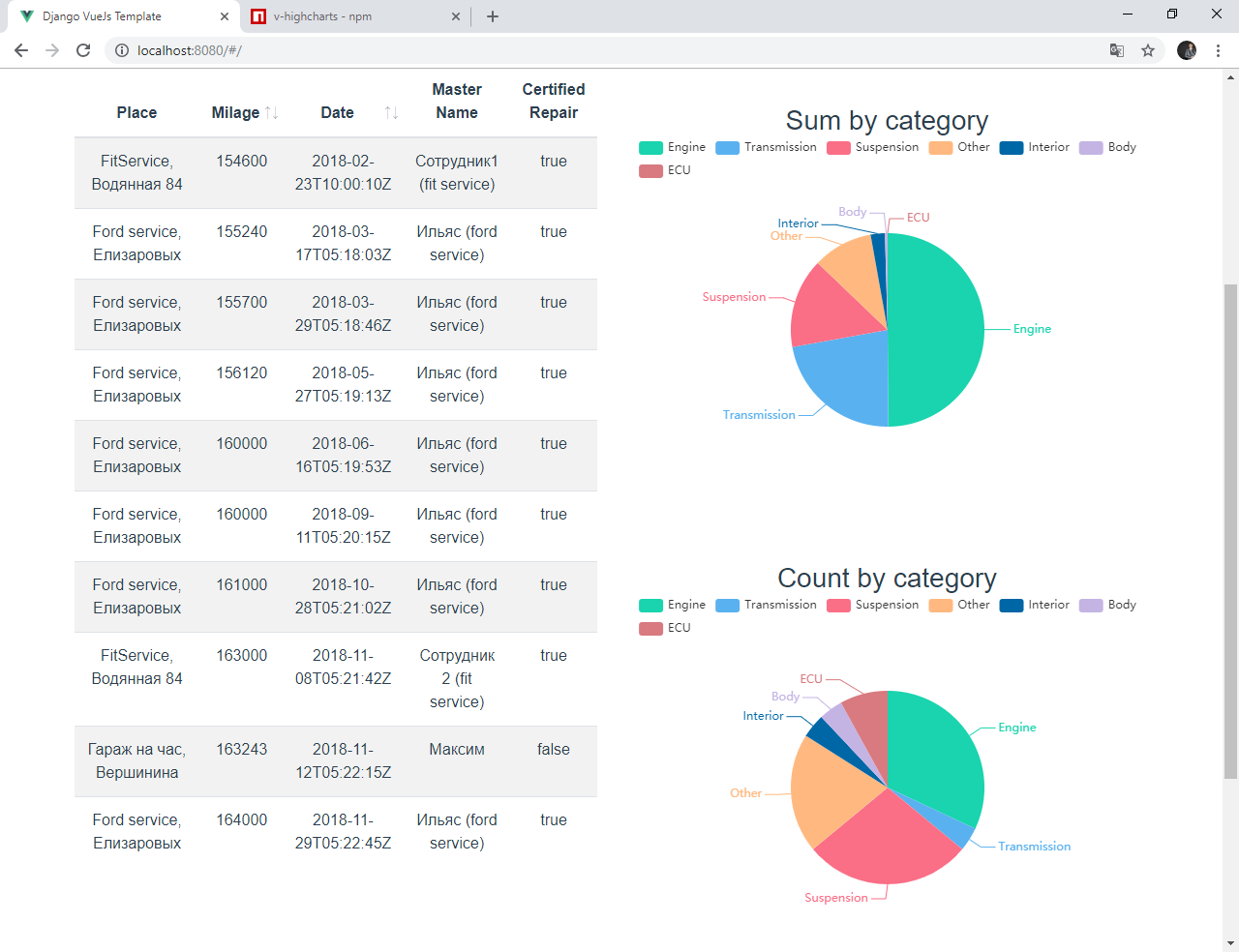


Figure 6 – Repairs page: bottom part

And if take a look on Figure 7 – we can see maps with all service stations, where car were served.

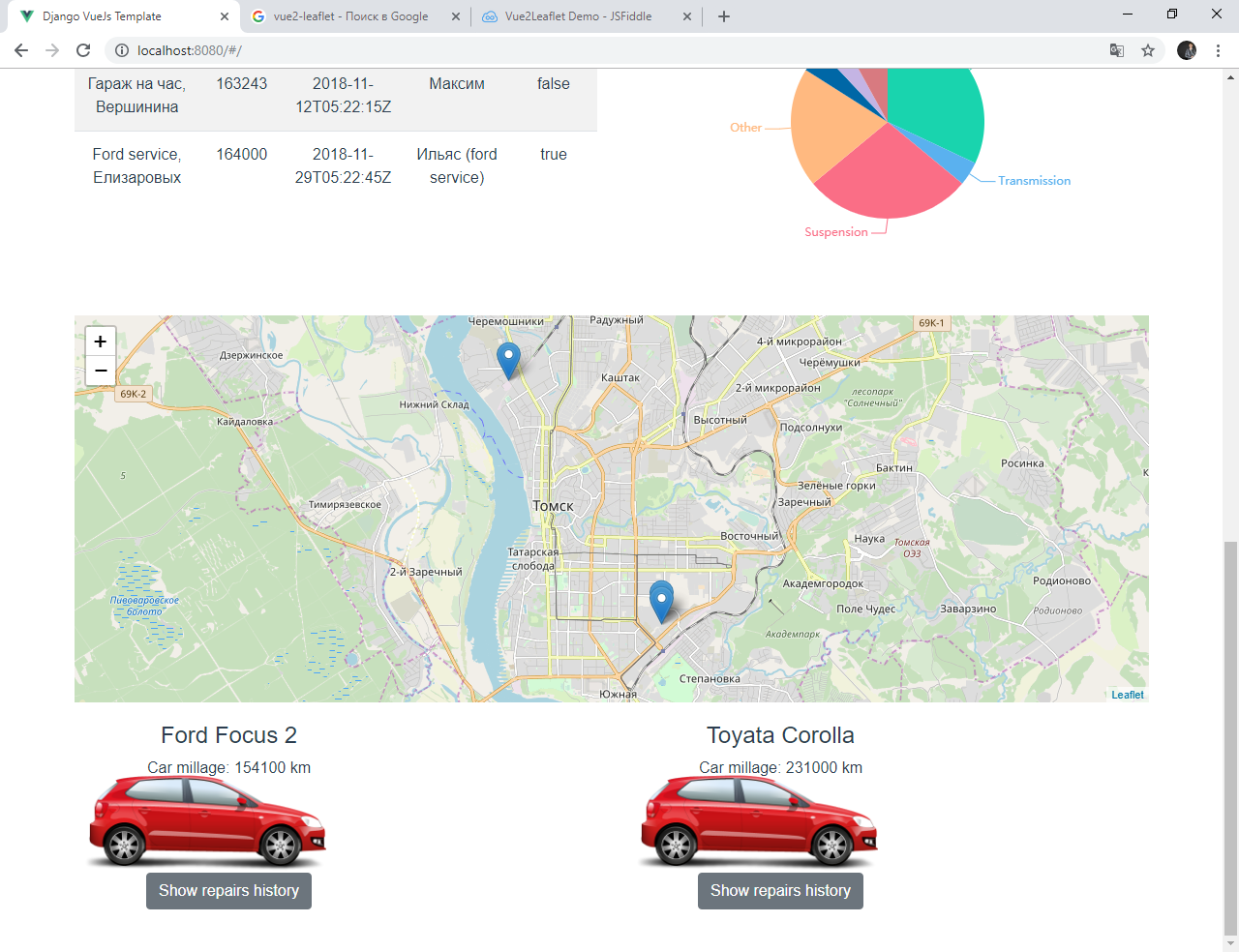


Figure 7 – Repairs page: bottom part 2

After clicking on the row with “repair”, a page with a detailed repair’s description opens (Figure 8-10).

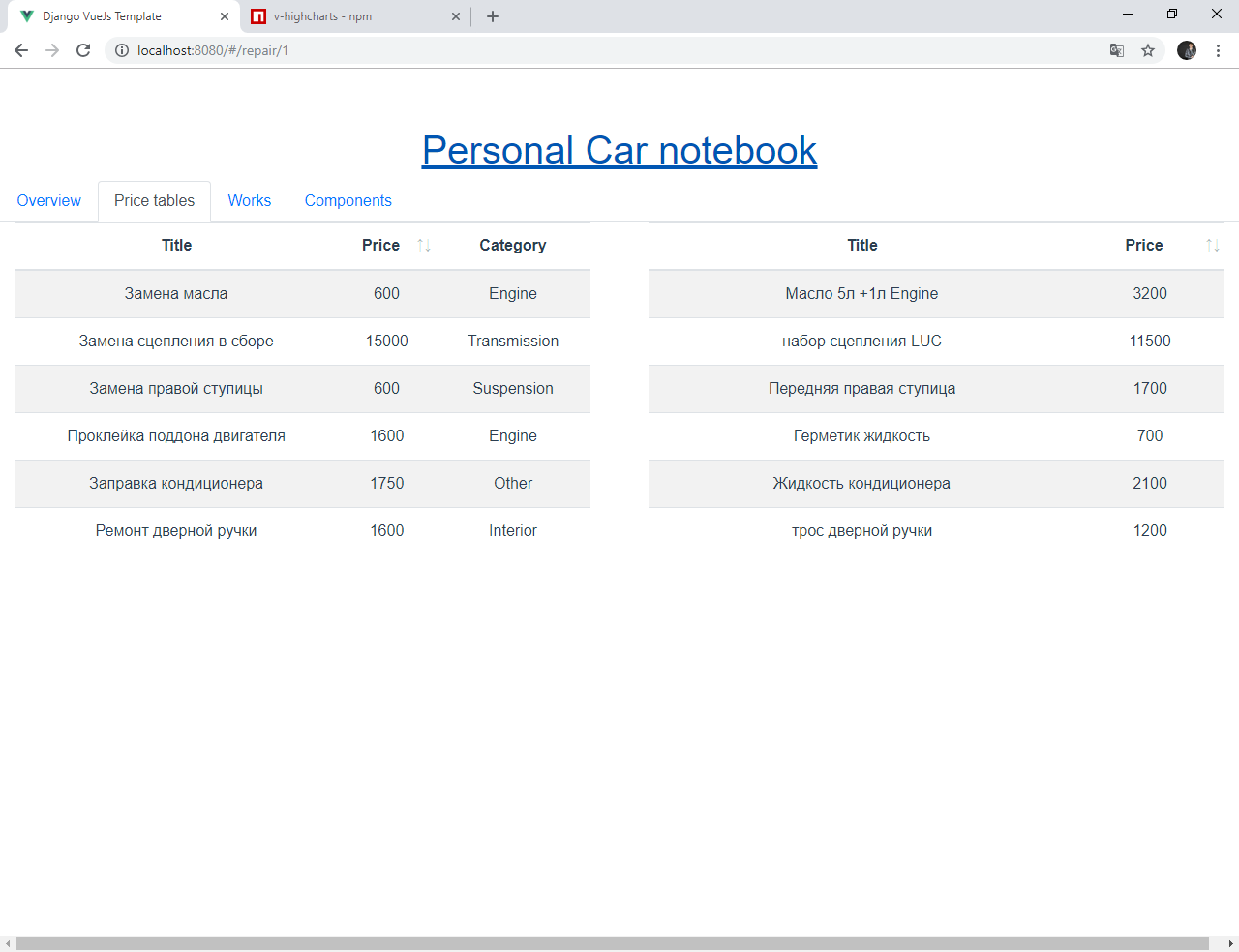


Figure 8 – Repair details: price tables

On Figure 6 we see all costs by works and components in tables view.

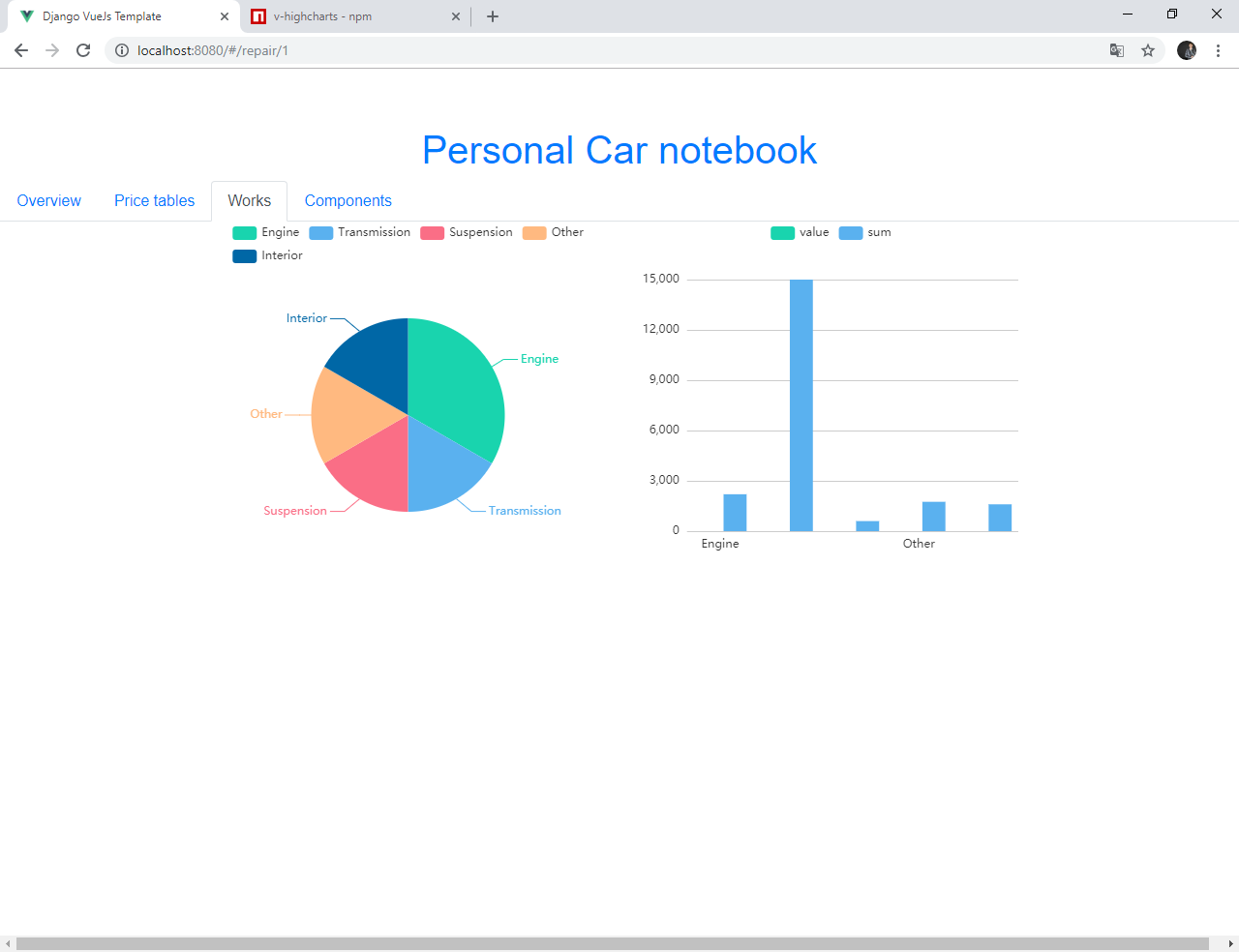


Figure 9 – Repair details: work’s category

When we click on “Works” tab we see pichart grouped by categories and show us how many works were performed for every category. Bar charts is showing costs by every category.

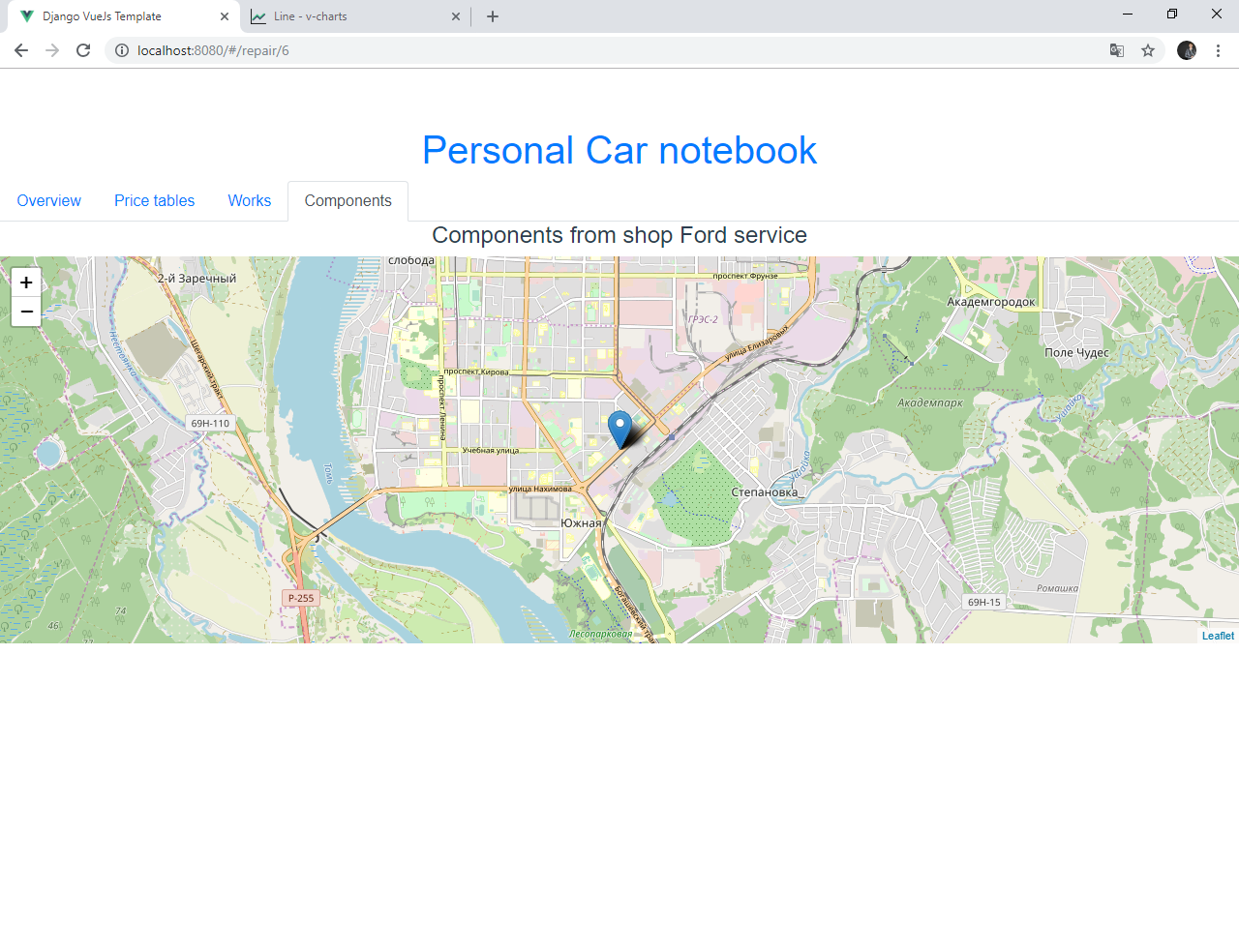


Figure 10 – Repair details: maps with car shops

And if we want to see where components were bought – there’s a map with markers. On figure 10 we can see “Ford service” in г.Томск, Елизаровых 18.

# CONCLUSION

In current research had been successfully created a web application for storing and visualizing data about the car repair history. All information is presented in various ways: in the form of tables, graphs, diagrams, distribution on the map.

Moreover, current application can be used for own goals – to see how many economics resources yours car is wasted.

REFERENCES

1. Django documentation [electronic resource]. – URL: [https://docs.djangoproject.com/en/2.1/](http://flask.pocoo.org/) (Date of access: 12.01.2019)
2. Документация Django на русском [electronic resource]. – URL: [https://djbook.ru/](http://flask.pocoo.org/) (Date of access: 12.01.2019)
3. Vue js The Progressive JavaScript Framework [electronic resource]. – URL: https://vuejs.org/ (Date of access: 12.01.2019)
4. Leaflet - a JavaScript library for interactive maps [electronic resource]. – URL: https://leafletjs.com/ (Date of access: 12.01.2019)
5. V-charts.js | Open source HTML5 Charts for your website [electronic resource].

* URL: https://v-charts.js.org/#/en/line (Date of access: 12.01.2019)

# Appendix 1

**class** Car(models.Model):  
 name = models.CharField(max\_length=30)  
 milage = models.PositiveIntegerField()  
  
 **def** \_\_str\_\_(self):  
 **return** self.name  
  
**class** Person(models.Model):  
 name = models.CharField(max\_length=30)  
 isMaster = models.BooleanField(default=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.name  
  
**class** CarShop(models.Model):  
 titile = models.CharField(max\_length=100)  
 address = models.CharField(max\_length=256)  
 latitude = models.FloatField()  
 longitude = models.FloatField()  
  
 **def** \_\_str\_\_(self):  
 **return** self.titile + **', '** + self.address  
  
  
**class** RepairStation(models.Model):  
 titile = models.CharField(max\_length=100)  
 address = models.CharField(max\_length=256)  
 latitude = models.FloatField()  
 longitude = models.FloatField()  
 isStation = models.BooleanField(default=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.titile + **', '** + self.address  
  
**class** Reapir(models.Model):  
 date = models.DateTimeField(default=timezone.now)  
 milage = models.PositiveIntegerField()  
 car = models.ForeignKey(Car, on\_delete=models.CASCADE, null=**False**)  
 place = models.ForeignKey(RepairStation, on\_delete=models.CASCADE, null=**False**)  
 master = models.ForeignKey(Person, on\_delete=models.CASCADE, null=**False**)  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.place) + **' at:'** + str(self.date)  
  
**class** Component(models.Model):  
 repair = models.ForeignKey(Reapir, on\_delete=models.CASCADE, null=**True**)  
 titile = models.CharField(max\_length=100)  
 price = models.FloatField()  
 country = models.CharField(max\_length=12)  
 purchase\_place = models.ForeignKey(CarShop, on\_delete=models.CASCADE, null=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.titile  
  
**class** RepairWork(models.Model):  
 WORK\_ENUM = (  
 (**'E'**, **'Engine'**),  
 (**'T'**, **'Transmission'**),  
 (**'U'**, **'ECU'**),  
 (**'B'**, **'Body'**),  
 (**'I'**, **'Interior'**),  
 (**'O'**, **'Other'**),  
 (**'S'**, **'Suspension'**),  
 )  
 repair = models.ForeignKey(Reapir, on\_delete=models.CASCADE, null=**True**)  
 titile = models.CharField(max\_length=100)  
 price = models.FloatField()  
 category = models.CharField(max\_length=1, choices=WORK\_ENUM)  
  
 **def** \_\_str\_\_(self):  
 **return** self.category + **': '** + self.titile