**分类号：TP311.1 U D C：D10621-408-(2020)3181-0**

**密 级：公 开 编 号：2018121032**

**成都信息工程大学**

**学位论文**

**基于vue的汽车资讯网站的设计与实现**

|  |  |
| --- | --- |
| **论文作者姓名：** | **向峰** |
| **申请学位专业：** | **网络工程** |
| **申请学位类别：** | **工学学士** |
| **指导教师姓名（职称）：** | **王燚（教授）** |
| **论文提交日期：** | **2022年06月15日** |

**汽车资讯网站的设计与实现**

**摘 要**

逐渐提高的生产消费水平大大提高了人们的出行需求，自行车太慢，公交车太挤，于是乎人们更多地倾向于购买属于自己的汽车来满足自己与家人的出行需求。但是大多数汽车4S店都位于城郊处，而且线下看车非常不方便，需要占用大量的时间前往4S店进行选车，即使是购买二手车，在二手车交易店里还可能被中间商赚差价，可谓是十分的不划算了。这种情况下，我们就急需一款产品提供优质高效实惠的汽车资讯购买服务。

而guloti汽车资讯网站则恰恰满足了高效实惠轻量这一特点，可以让用户选得舒心，买的放心。guloti汽车资讯网站基于vue2+express进行开发，通过基于websocket的socketio进行消息双向通信，Elementui作为ui库提供样式，前后端都采用js作为开发语言，采用mysql5.6做为数据库提供数据源支持，阿里云轻量级服务器部署的tomcat提供资源提供服务。本系统目前采用客户端、商家端、管理端三合一模式，一套系统，三方使用，轻便高效。本系统实现了账户的注册、登，客户端的资讯展示、新车的多条件查询搜索、车辆详情、新车的购买入库、询问卖家、二手车的挂售信息设置、出售、购买，车友会的品牌选择、新帖的发布、新帖详情查看、评论的发布，个人信息页面的头像上传、信息设置、注销、申请成为商家，商家可以进行新车信息上传发布、修改车辆信息、删除车辆信息，系统管理员可以进行新帖的发布，路由与侧边栏采用依据权限动态生成。

guloti汽车网功能完善并将持续更新，用户体验较好，经测试，已开发功能使用情况良好，基本达到开发目的。

**关键词：**汽车；vue+express；socketio；

**Design and Implementation of Automobile Information Website**

**Abstract**

The gradual improvement of production and consumption level has greatly increased people's travel demand. Bicycles are too slow and buses are too crowded, so people tend to buy their own cars to meet the travel demand of themselves and their families. However, most 4S shops are located in the suburbs of the city, and it is very inconvenient to look at cars offline, so it takes a lot of time to go to 4S shops for car selection. Even if you buy second-hand cars, middlemen may earn price differences in second-hand car dealers, which is very uneconomical. In this case, we are in urgent need of a product to provide high-quality, efficient and affordable car information purchase services.

Guloti automobile information website is precisely to meet the characteristics of efficient, affordable and lightweight, can let users choose comfortable, buy at ease. Guloti automobile information website is developed based on VUe2 + Express, two-way message communication is carried out through Socketio based on Websocket, Elementui provides style as UI library, JS is used as development language at both front and back ends, and mysql5.6 is used as database to provide data source support. Ali Cloud lightweight server deployment of Tomcat to provide resources to provide services. The system currently uses the client, business terminal, management terminal three in one mode, a set of systems, three parties to use, portable and efficient. This system has realized the account registration, boarding, client information display, new car multi-condition query search, vehicle details, new car purchase into the warehouse, asking sellers, second-hand car sales information setting, sale, purchase, car club brand selection, new post release, new post details view, comment release, Personal information page of the profile picture upload, information Settings, logout, application to become a business, business can upload and publish new car information, modify vehicle information, delete vehicle information, system administrator can publish new posts, routes and the sidebar based on permissions dynamically generated.

Guloti auto network has perfect functions and will continue to update, with good user experience. After testing, the developed functions are in good use, basically achieving the development purpose.

**Key words:**The carinformation;Vue;Express；Socketio；

目录

[1.引言 6](#_Toc11791)

[1.1课题背景 6](#_Toc8570)

[1.1.1 线上汽车市场 6](#_Toc6247)

[1.2国内外研究现状 6](#_Toc10047)

[1.3本课题研究的意义 6](#_Toc10824)

[1.4本课题研究的内容 6](#_Toc13101)

[2. 相关技术概述 6](#_Toc16498)

[2.1开发技术分析介绍 6](#_Toc18168)

[2.1.1 Vue2 6](#_Toc30169)

[2.1.2 Express 6](#_Toc31119)

[2.1.3 js 6](#_Toc30287)

[2.1.4 Mysql 7](#_Toc32150)

[2.1.5 Element ui 7](#_Toc28577)

[2.1.6 Vue-cli3 7](#_Toc16526)

[2.1.7 webpack 7](#_Toc23686)

[2.2开发与运行环境分析介绍 7](#_Toc4241)

[2.2.1 Chorme V8 7](#_Toc10)

[2.2.2 nodeJS 7](#_Toc25730)

[2.2.3 tomcat 7](#_Toc23905)

[3.系统设计 7](#_Toc798)

[3.1数据库设计 7](#_Toc5958)

[3.1.1 车库表 7](#_Toc26464)

[3.1.2 品牌表 7](#_Toc15183)

[3.1.3 汽车表 7](#_Toc29954)

[3.1.4 汽车资讯表 7](#_Toc24793)

[3.1.5 汽车订单表 7](#_Toc26160)

[3.1.6 聊天记录表 7](#_Toc134)

[3.1.7 车友会分类表 7](#_Toc25508)

[3.1.8 评论表 7](#_Toc20568)

[3.1.9 帖子表 7](#_Toc705)

[3.1.10 用户信息表 7](#_Toc4588)

[3.2 用户权限设计 7](#_Toc359)

[3.2.1 未登录用户 7](#_Toc27504)

[3.2.2 普通用户 7](#_Toc5531)

[3.2.3 商家用户 7](#_Toc26936)

[3.2.4 超级管理员 7](#_Toc5942)

[3.2.5 动态生成路由 7](#_Toc29993)

[3.3功能模块设计 7](#_Toc15413)

[3.3.1 登录注册模块 7](#_Toc14932)

[3.3.2 新车选购模块 7](#_Toc9313)

[3.3.3 二手车交易模块 7](#_Toc24172)

[3.3.4 车库模块 7](#_Toc5992)

[3.3.5 车友会模块 7](#_Toc2054)

[3.3.6 车辆管理模块 7](#_Toc12455)

[3.3.7 首页资讯模块 7](#_Toc8132)

[3.3.8 资讯管理模块 8](#_Toc20004)

[3.3.9 个人中心模块 8](#_Toc27600)

[4.汽车资讯网站的开发与实现 8](#_Toc1172)

[4.1 页面请求响应 8](#_Toc24673)

[4.1.1 登录验证 8](#_Toc10338)

[4.1.2 错误拦截 8](#_Toc5397)

[4.2 代理设置 8](#_Toc15373)

[4.3 动态路由生成 8](#_Toc3682)

[4.4 后端结构 8](#_Toc22105)

[4.4.1 项目入口 8](#_Toc17276)

[4.4.2 路由层 8](#_Toc26409)

[4.4.3 控制层 8](#_Toc19540)

[4.4.4 socket服务端 8](#_Toc14325)

[4.4.5 其他工具 8](#_Toc16538)

[4.5 登录注册模块 8](#_Toc11586)

[4.5.1 注册 8](#_Toc22517)

[4.5.2 登录 8](#_Toc14469)

[4.6新车与二手车选购模块 8](#_Toc7371)

[4.6.1汽车筛选 8](#_Toc9804)

[4.6.2交易 8](#_Toc20030)

[4.7 车库模块 8](#_Toc4788)

[4.7.1汽车挂售 8](#_Toc13152)

[4.7.2 放弃挂售 8](#_Toc24166)

[4.8 车友会模块 8](#_Toc28833)

[4.8.1 帖子发布 8](#_Toc4282)

[4.8.2 评论发布 8](#_Toc14848)

[4.9 首页资讯 8](#_Toc14255)

[4.10 资讯管理 8](#_Toc20368)

[4.10.1资讯新增/删除 8](#_Toc7379)

[4.10.2 资讯修改 8](#_Toc24392)

[4.11 汽车管理 8](#_Toc7771)

[4.11.1 汽车发布/删除 8](#_Toc32318)

[4.11.2汽车信息修改 8](#_Toc13808)

[4.12 个人中心模块 8](#_Toc15541)

[4.12.1个人信息修改 9](#_Toc17158)

[4.12.2 头像上传 9](#_Toc13225)

[4.12.3 注销 9](#_Toc7341)

[5 系统功能展示 9](#_Toc20295)

[5.1 客户端 9](#_Toc31960)

[5.2 商家端 9](#_Toc19588)

[5.3 超级管理员 9](#_Toc28247)

[6 功能总结 9](#_Toc18569)

[结 语 9](#_Toc25697)

[参考文献 9](#_Toc5440)

[致 谢 9](#_Toc10843)

[声 明 9](#_Toc5398)

1.引言

1.1课题背景

人们生活物质水平的提高，让车房都成了一大消费热点，而消费的主要对象主要是30-40岁工薪阶层，因为这一年龄层的人群大多已经有了相对稳定的工作收入，对汽车的消费热情较高同时需求也比较大。

* + 1. 线下汽车市场

从现实情况出发，各大城市往往会开办一些汽车展，以供人们近距离的了解各品牌汽车。同时各类汽车实体店也有着配套的汽车咨询服务以及过户，保险，维修等服务。线下汽车市场最大的好处就在与为用户提供了一站式服务，能够让用户更加真切得观看，试驾到自己有意向的汽车。同时线下交易也能一手交钱一手交货，买的更加放心，避免了一些不必要的损失。

* + 1. 线上汽车市场

但是，线下汽车市场就真的这么好吗？也不尽然，无论是展览会还是4S店，巨大的人力，店铺，场地成本，实际上这些成本还是会变成服务费还是要摊到交易之中的，同时，线下看车买车有巨大的局限性，或许一天下来也看不了几家4S店，十分费心费时费力。因此也就应运而生了各类汽车网站，让用户线上看车买车。所谓新车二手车直卖网站也是多如牛毛，但实际上这些网站上存在大量的“二房东”，用户根本无法与卖家直接沟通，既不安全也不实惠。

1.2本课题研究的意义

因此经过调查，我决定设计一款能够让用户在方便看车的同时，能够直接与商家或者二手车卖家对话的网站。简单便捷的注册方式，高效的选车工具，用户能够快捷的选择到自己喜欢的车辆，网站风格简约，没有花里胡哨的广告，提升用户体验。更可以与卖家一对一通信，拿到第一手汽车资料，买的放心。不仅如此，用户还可以从车友会其他用户的口中了解到更多关于汽车的信息，相互参谋，避免非理性消费。同时网站也为商家提供管理界面，让商家能够及时快捷的对汽车进行管理。

1.3本课题研究的内容

经过需求分析显示，本网站需要实现三个客户端，包含9大模块，主要涉及用户选购买卖汽车，车友会，汽车管理，资讯发布展示等。同时是将客户端，商家端，管理员端三端合一，根据权限获取各自能够访问的页面，便捷的同时提升网页各模块的复用性。

1. 相关技术概述

2.1开发技术分析介绍

* + 1. Vue2

Vue是用于构建用户界面的渐进式JavaScript框架，通过组件化模式提高了代码的可复用性，同时让代码更加便于维护。声明式的编码方式与虚拟dom再加上diff算法，让开发人员在不需要重复操作真实DOM节点的情况下能够多次复用DOM节点，提高开发效率，也让开发人员将开发关注重心注重于数据层面而非页面的显示上。

插值语法，指令语法，以极低的成本就可以让数据在页面上展示。而完成一个vue页面的搭建，仅仅需要学会了解以下几个知识点。

生命周期：vue的8个生命周期函数，分别让页面在不同时段执行用户所需要的功能，例如mounted阶段可以用于发送ajax请求，beforeDestroy阶段可以用于清除登录信息，清除定时器等。

Data:用于保存当前vue页面初始化时的变量，这些在data中定义的变量可以在html代码中直接展示，也可以在Js中调用，让变量能够在html中动态变化。

Computed:计算属性，根据data数据发生变化就会重新解析模板，也可以直接读取。

* + 1. Express

Express 基于 Node.js 的Web应用程序开发框架，可以为网页和APP提供后台服务，Express框架建立在node.js内置的http模块上。

Express框架有一个重要的特点就是中间件，即处理HTTP请求的函数，通过next()将请求传递给下一个中间件，它每次接受req,res,next三个参数，分别代表请求，相应和下一个中间件。

Express 4.0版本后开始，Express的路由器功能成了一个单独的组件Express.Router。它好像小型的express应用程序一样，有自己的use、get、param和route方法。Express的路由器可以通过解析路由路径以处理不同的http请求。

* + 1. Mysql

mysql是关系形势数据库的典型代表，是当前最流行的数据库之一，由MySQLAB公司开发，mysql将数据存在于不同的表中，通过字段，主键，外键等定义各种表内与表间的关系。mysql使用的是最标准的sql语言，体积小，速度快，开放源码，也因此成为了大多数中小型网站热衷的数据库。mysql还有针对不同规模应用的三种架构，分别是单点，复制，集群，可以针对性的提供自己的功能。

* + 1. Element ui

Element ui是当前最火的ui库之一，他不依赖于vue，确实和vue配合的最紧密最方便的ui库。它由饿了么前端团队开发，ui风格简约实用，提供大量提前写好的属性以及模板以供开发者使用，极大地提高了开发效率，他的组件化也是非常契合vue的组件化思维，让vue项目开发者无需大量重复写一些小组件，而将更加关注数据的处理。

element提供全局引入和按需引入两种方式，根据用户不同需求进行使用。

* + 1. Vue-cli

Vue-cli是基于vue进行开发的项目脚手架，可以快速进行vue项目的快速搭建。Vue cli可以将vue生态中的各个工具进行标准化管理，因为它默认内置webpack工具，且有自带合理的包管理配置，可以扩展其他插件以保证插件与项目的平稳衔接。

在安装vue-cli之后，仅仅通过简单的命令行即可完成vue项目的创建，再创建过程中可通过可视化的方式对vue项目进行预设引入模块配置。

* + 1. Webpack

webpack是一种模块打包工具，使用webpack可以管理项目中模块之间的依赖关系，可以将各模块之间的文件进行打包生成web页面所需要的html,css,js以及其他静态资源文件，让开发过程更加高效。webpack会自动将各模块之间的调用关系进行分析，优化并生成最后的静态资源。

Webpack将整个项目当成整体，从给定的主文件开始寻找项目中所有的依赖文件，通过loaders来进行处理，将这些文件编译转化打包成为浏览器可以识别的js文件。除了loaders，webpack还可以使用plugin来对打包进行优化压缩，用以生成体积更小的bundle文件，使页面加载时更快。

1. 系统设计

3.1需求概述

网站开发所要达成的目的是方便客户对于汽车的选购，同时需要满足客户能够多方面的获取汽车讯息，因此需要设计汽车新闻，交流工具，车友会，以及汽车选购市场等功能，要求界面简介，流程简单且安全。提供二手车市场，避免线下二手车市场的复杂情况，用户可以自主提交二手车发售消息，为客户提供直接交流的平台。

管理层面，超级管理员和商户可以在自己权限范围内对汽车资讯，新车信息进行管理，因此要设计相应的管理系统。在提交数据过程中，前台需要对于数据进行验证，符合规范的情况下才允许数据上传。

由于涉及消息发布，有上传文件的需求，因此需要搭建在线服务器，用阿里云轻量化服务器，部署Tomcat服务，提供在线资源。

3.2开发运行环境

操作系统要求：centos 7.9;

开发操作系统:window 10

运行环境：nodejs14

服务器要求:tomcat 7

开发工具:visual studio code

数据库:mysql 5.6及以上

3.1数据库设计

* + 1. 车库表

车库表信息：车库表id，汽车表id，用户表id，车牌号，里程数，订单id，售出状态（在库/在售/已售），定价

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| barn | id |  | int(11) | PRI |
| barn | carid | 汽车id | int(11) |  |
| barn | userid | 用户id | int(11) |  |
| barn | car\_number | 车牌号 | varchar(11) |  |
| barn | km | 里程数 | int(11) |  |
| barn | orderid | 所属订单，表示是哪个订单完成此车购买的 | int(11) |  |
| barn | sellstate | 该汽车售出状态,默认为 '在库'，有'在库','在售','已售‘等状态 | varchar(255) |  |
| barn | pricing | 当前车辆定价，可由车主决定，在库状态默认为0，单位w | int(11) |  |

* + 1. 品牌表

品牌表信息：品牌表id，品牌名，品牌Logo图片链接，品牌首字母

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| brand | id |  | int(11) | PRI |
| brand | brand\_name | 品牌名称 | varchar(255) |  |
| brand | brand\_logo | 品牌logo | varchar(255) |  |
| brand | brand\_intial | 品牌首字母 | varchar(255) |  |

* + 1. 汽车表

汽车表信息：汽车表id，汽车名字，车型（轿车/SUV/MPV/跑车），品牌表id，品牌名，车辆简介，车辆缩略图链接，车辆展示图列表地址，价格下限，价格上限，产地，生产日期，是否为新车（0为false,1为true），公里数，发动机型号，变速箱型号，汽车评分，参与汽车评分人数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| car | id |  | int(11) | PRI |
| car | car\_name | 汽车名字 | varchar(255) |  |
| car | car\_type | 车型 | varchar(255) |  |
| car | car\_brandid | 品牌号 | varchar(255) |  |
| car | car\_brand | 品牌名 | varchar(255) |  |
| car | car\_introduce | 车辆简介 | text |  |
| car | car\_showimg | 车首展图地址 | varchar(255) |  |
| car | car\_exhibition\_list | 车展览图列表地址，以$分隔 | text |  |
| car | car\_price\_low | 起始价格 | int(11) |  |
| car | car\_price\_high | 顶配价格 | int(11) |  |
| car | car\_manufacture\_addr | 产地 | varchar(255) |  |
| car | car\_manufacture\_date | 生产日期 | varchar(255) |  |
| car | isnew | 是否是新车 | tinyint(255) |  |
| car | kmruned | 已跑公里数，新车默认为0 | int(255) |  |
| car | car\_engine | 发动机型号 | varchar(255) |  |
| car | car\_at | 变速箱型号 | varchar(255) |  |
| car | car\_score | 汽车评分 | double(255,0) |  |
| car | car\_score\_nums | 参与汽车评分的人数 | int(255) |  |

* + 1. 汽车资讯表

汽车资讯信息：汽车资讯id，资讯名称，资讯缩略图链接，资讯类型（文字资讯/视频资讯），视频资讯链接，文字资讯内容，发布日期，播放次数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| car\_news | id |  | int(11) | PRI |
| car\_news | news\_title | 资讯名称 | varchar(255) |  |
| car\_news | news\_headimg\_url | 资讯缩略图url | varchar(255) |  |
| car\_news | news\_content\_type | 资讯类型,文字资讯/视频资讯 | varchar(255) |  |
| car\_news | news\_video\_url | 视频资讯url | varchar(255) |  |
| car\_news | news\_text | 文字资讯内容 | text |  |
| car\_news | news\_time | 发布日期 | varchar(255) |  |
| car\_news | playtimes | 播放次数 | int(11) |  |

* + 1. 汽车订单表

汽车订单表信息：汽车订单表id，汽车表id，用户表id，购买时间，订单类型（0为新车，1为二手车），卖家id（新车为品牌id，二手车为上任车主id），交易额

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| car\_order | id |  | int(11) | PRI |
| car\_order | carid | 汽车id | int(11) |  |
| car\_order | userid | 用户id | int(11) |  |
| car\_order | order\_time | 购买时间 | varchar(255) |  |
| car\_order | order\_type | 订单类型：0为新车，1为2手车 | int(255) |  |
| car\_order | sell\_id | 卖家id,新车为brandid,二手车为上一任车主id | int(11) |  |
| car\_order | sellprice | 交易额 | int(10) |  |

* + 1. 聊天记录表

聊天记录表信息：聊天记录表id，发送方用户id，接收方用户Id,接收方是否已读消息（0为false，1位true），消息内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| chatlist | id | 聊天表id | int(11) | PRI |
| chatlist | from\_userid | 发送信息的用户id | int(11) |  |
| chatlist | to\_userid | 接收信息的用户id | int(11) |  |
| chatlist | readed | to方是否已读信息,0为否，1为是 | int(11) |  |
| chatlist | content | 评论内容 | text |  |

* + 1. 车友会分类表

车友会分类表：车友会表id，所属品牌id

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| club | id | 车友会id | int(11) | PRI |
| club | brandid | 所属品牌id | int(11) |  |

* + 1. 评论表

评论表信息：评论表Id,评论类型（车友会帖子评论/资讯评论），评论人id，评论内容，评论时间

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| comment | id |  | int(11) | PRI |
| comment | comment\_type | 'invitation'车友会评论，'car\_news'资讯评论 | varchar(255) |  |
| comment | from\_id | 根据comment\_type选择是那种内容的评论id | int(11) |  |
| comment | userid | 评论人id | int(11) |  |
| comment | comment\_content | 评论内容 | text |  |
| comment | comment\_time | 评论时间 | varchar(255) |  |

* + 1. 帖子表

帖子表信息：帖子id，所属车友会id，发帖人，帖子简介，帖子详细内容，发帖时间

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| invitation | id | 帖子id | int(11) | PRI |
| invitation | clubid | 所属车友会id | int(11) |  |
| invitation | userid | 发帖人 | int(11) |  |
| invitation | brief\_context | 帖子简介 | varchar(255) |  |
| invitation | detail\_context | 帖子详细内容 | text |  |
| invitation | time | 发帖时间 | varchar(255) |  |

* + 1. 用户信息表

用户表信息：用户表id，账号，密码，账户名，头像，余额，socket连接号，用户身份（normal/store/admin，分别对应普通用户/商户/超级管理员），账户所属公司id（普通用户为-1/商户为品牌id/超级管理员为0），是否在线（1为true，0为false）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 代码 | 名称 | 数据类型 | 主键 |
| user | id |  | int(11) | PRI |
| user | account | 账号 | varchar(255) |  |
| user | password | 密码 | varchar(255) |  |
| user | name | 账户名 | varchar(255) |  |
| user | headimg | 头像 | varchar(255) |  |
| user | balance | 余额，单位w | int(255) |  |
| user | socketid | socketid | varchar(255) |  |
| user | usertype | 用户身份:normal/store/admin | varchar(255) |  |
| user | companyid | 账户所属公司的id:普通用户为-1；公司服务号为公司或品牌id  所属企业的Id：普通用户为-1；员工为企业的brand\_id;超管为0 | int(11) |  |
| user | isonline | 是否在线,1为true,0为false | tinyint(1) |  |

3.2 用户权限设计

根据用户身份分配权限，不同的权限对应可以进入不同的网页。

3.2.1 未登录用户

用户刚进网页时处于未登录状态，只能访问选车，资讯，车友会页面，但无法进行任何提交，购买操作。一旦进行上述操作会被拦截。

3.2.2 普通用户

用户注册后初始身份即为普通用户（normal），普通用户可以访问选车，资讯，车友会，车库等界面，能够进行发帖，评论，购买汽车，管理车库等操作。普通用户在个人信息中心可以申请成为商家用户。

3.2.3 商家用户

商家用户无法进入选车，车库等界面，但允许访问车辆管理系统，能够对车辆进行新增上传，以及现有车辆的编辑。

3.2.4 超级管理员

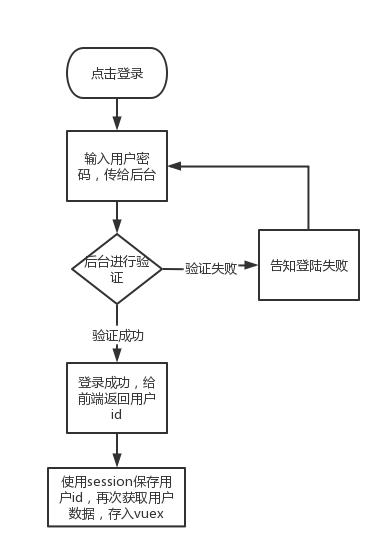
超级管理员可以进入任意界面，较其他权限可以访问资讯管理界面，能够对资讯进行上传发布编辑。

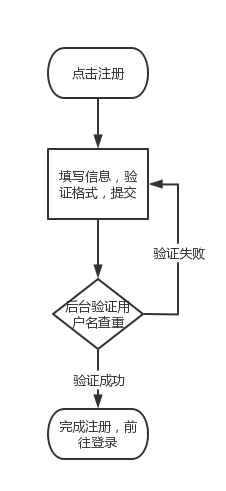
3.2.5 动态生成路由

本网站采用的是客户端，管理端，商家端三合一模式，许多功能不能跨越权限操作，因此需要通过权限验证，生成各自可访问的路由列表。

3.3功能模块设计

3.3.1 登录注册模块

登录：

注册：

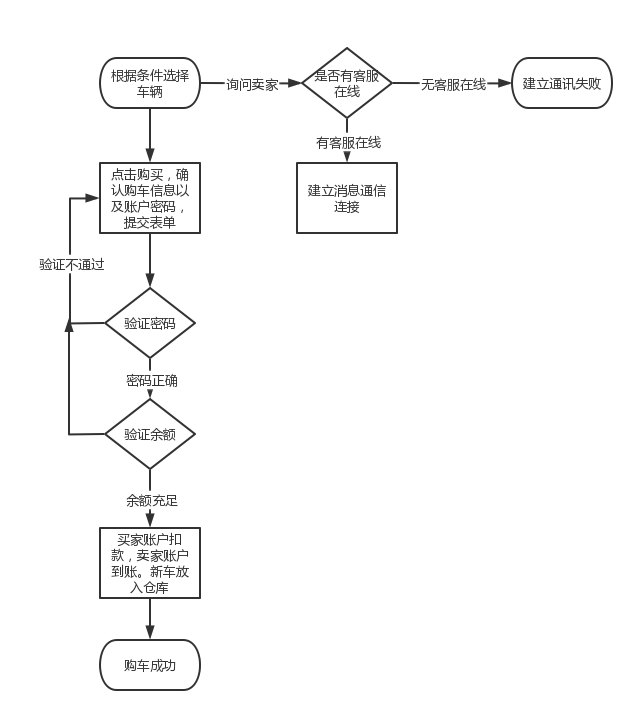
3.3.2 新车/二手车选购模块

车辆选购模块包含三个小模块，分别是选车，与商家交谈，购买付款。

选车环节为满足客户需求，设置了多条件选择器，可以通过品牌，车型，价格区间，或者是输入名字等多个条件进行筛选，以此提供出符合用户条件的车辆列表以供用户选购。

为了让用户得以了解到信息能够清楚明了，只靠页面展示的部分是远远不够的。因此我们为用户提供与商家发消息的平台，只要相应商家有客服在线的情况下，就可以随时进行咨询，一旦建立了通讯，即使客服或用户已经下线，发送过来未读的消息也会在下次通讯时展示出来。

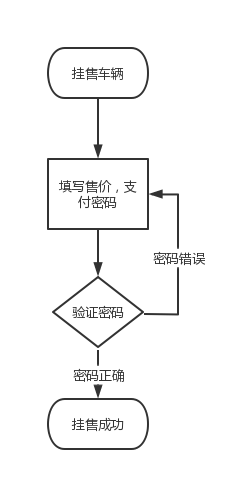
点击购买车辆后，会出现表单，其中会有车辆相关信息以供用户确认，用户需要输入支付密码并提交表单。表单会在后台以此对支付密码，账户余额进行验证，验证通过之后就会进行金额转移的流程。新车完成购买之后，会新增订单记录，并将新车入库。二手车完成购买之后，除了新增订单记录，车库中二手车的所属人信息会进行相应更新。



3.3.4 车库模块

个人车库中主要涉及已购车辆的展示，个人车辆的挂售/取消挂售的功能。

当车主需要将自己的车辆卖出时，会出现表单让用户填写售价以及个人支付密码等信息。填写完毕之后，后台对车主密码进行验证，验证通过即将车辆添加到二手车市场。个人车库中的该车也将处于挂售状态，车主可随时取消挂售。

在二手车市场中的车被购买之后，车辆所属人会被修改成买家。

3.3.5 车友会模块

车友会首页会提供所有品牌车友会的入口。选择一个车友会进入之后，会展示车友会近期发表的帖子的简述信息，点击查看全文可以浏览帖子的具体内容。每个帖子提供评论功能，也可以查看近期评论并能够发布个人评论。

发布帖子功能采用的是富文本文本框，可以往里面直接添加图片，并可以设置字体。

3.3.6 车辆管理模块

车辆管理模块必须是商家才能够访问，包含了新车上传，车辆信息编辑以及删除功能。

新车上传以及编辑功能必须给提供车辆名字，展示图，售价，相关参数等信息，否则无法上传或保存信息成功。

3.3.7 首页资讯模块

资讯模块用于首页展示，包含轮播图以及资讯列表。新闻资讯主要分为文字资讯以及视频资讯两类。

3.3.8 资讯管理模块

资讯管理模块由超级管理员进入。主要涉及资讯上传，编辑，删除等功能。按照要求填写相应字段即可进行资讯的上传与保存。需上传标题，展览图，文本等消息。

3.3.9 个人中心模块

个人中心页面主要包含信息的更改，头像上传，个人所属公司的更改。用户在此处可以选择的本地文件上传头像，但头像必须是jpg格式且有大小限制。用户可以选择个人身份，可以申请成为某品牌客服。

4.汽车资讯网站的开发与实现

4.1 页面请求响应

4.1.1 登录验证

因为类似于发布评论，交易等大量功能需要用户处于在线状态，因此对于用户的请求设置了请求拦截以验证用户登录状态，主要是通过验证session中存储的myid键。只有当用户信息已经存在时，请求才会真正向后端发出。

核心代码如下：

request.interceptors.request.use(config => {

    if (!sessionStorage.getItem('myid')) {

        return Promise.reject({

            code: 20001,

            message: '请先完成登录'

        })

    } else {

        return config

    }

}, err => {

    return Promise.reject(err)

})

4.1.2 错误拦截

无论是成功还是失败请求都在响应对象里定义了三个属性表示相关信息，code表示自定义响应码，message表示响应信息，data表示返回的数据

当后台校验数据需要报告验证失败或者请求失败时，需要返回自定义错误信息。定义code:20000为正确信息，code:20001为错误信息。当响应拦截器获取到code:20001的信息时，就会返回promise.reject对象，告诉前台失败信息。

核心代码如下：

request.interceptors.response.use(response => {

    if (response.data.code != 20000) {

        return Promise.reject(response.data)

    } else {

        return response //拦截处理响应结果，直接返回需要的数据

    }

}, err => {

    console.log(err)

    return Promise.reject(err)

})

4.2 代理/打包设置

设置反向代理是为了避免开发时的跨域问题，以及相关websocket通讯的允许问题。可以根据请求baseurl替换成自己希望的目标请求地址。

核心代码如下：

      '/loc': {

        pathRewrite: {

          '^/loc': ''

        },

        target: "http://localhost:3201",

        ws: true, // 是否开启 websokets

        secure: false, // 是否安全，https 为 true，http 为 false

        changeOrigin: true,

      },

打包设置是为了将前端项目生成网页可是识别的html，js，css等静态资源文件，并且使用了plugin的插件对打包进行优化，将生成的静态资源文件进一步压缩，提高页面的加载速度，减小打包文件的大小。

打包核心代码如下：

  configureWebpack: {

    plugins: [

      new CompressionPlugin({

        test: /\.(js|css)?$/i, // 哪些文件要压缩

        filename: '[path].gz[query]',　// 压缩后的文件名

        algorithm: 'gzip',　// 使用gzip压缩

        minRatio: 1,　// 压缩率小于1才会压缩

        deleteOriginalAssets: false // 删除未压缩的文件，谨慎设置，如果希望提供非gzip的资源，可不设置或者设置为false

      })

    ]

  }

4.3 动态路由生成

本系统将客户端，管理端，商家端进行整合，涉及权限问题，因此都需要根据不同身份权限来对路由进行处理，让用户无法跨权限操作访问。每个用户在数据库中都有一个身份字段，路由文件中使用路由元数据中的limits存储能够访问该网页的身份。

每次当网页载入时，会根据登录用户身份与limits中的身份限制过滤出用户可以访问的路由列表并保存到vuex数据仓库中。

核心代码：

Routerjs:

       meta: {

          title: '首页',

          sidebar: true,

          require\_login: false,

          icon: 'el-icon-menu',

          limits: ['admin', 'store', 'normal']

        }

用于过滤路由列表的函数:

export function filtrouters() {

  let myroutes = routes.options.routes[0].children

  if (store.getters.getme) {

    let usertype = store.getters.getme.usertype

    myroutes = myroutes.filter(function (item) {

      return item.meta.sidebar && item.meta.limits.includes(usertype)

    })

  } else {

    myroutes = myroutes.filter(function (item) {

      return item.meta.sidebar && !item.meta.require\_login

    })

  }

  store.commit('setmyroutes', myroutes)

  console.log(store.getters.getmyroutes);

}

过滤路由函数会在页面初始化和登录时调用，生成可访问的路由列表。

4.4 后端结构

4.4.1 项目入口

项目后端采用express框架，入口文件为app.js，在app.js中主要是对全局资源的引入，包括全局用到的中间件，请求入口。App.js中需要对请求路由进行注册分配，将不同的请求根据路径分发到不同的路由层。另外还需要引入post请求的处理中间件，因为express默认没有对post请求参数的接收方法，因此需要引入额外引入。

4.4.2 路由层

路由层负责接收分配的请求，并将请求传递给控制层。

4.4.3 控制层

控制层是真正处理请求的地方，路由层分配的请求都会对应控制层的一个方法。参数req，res分别是请求体，响应体，可以获得参数，进行操作，然后返回前端所需要的数据。

4.4.4 socket服务端

引入全局socketio-server工具，在前端初次连接服务端时，服务端的socket会为前端分配socketid来标识用户。前后端的socket连接主要是通过socket.on和socket.emit来进行侦听和触发，从而完成双向通信。

4.4.5 其他工具

主要包括数据库连接和sql语句工具等，在控制层可以快速调用这些已经封装好的js文件中的方法。

4.5 登录注册模块

4.5.1 注册

注册功能需要用户提交用户名密码等信息，前端进行验证后提交至后端。后端会在查重之后进行数据库的信息提交。也并非所有信息都需要用户提交，比如头像，在初次注册时会设置默认头像。

查重实现：

    async cnki(table, columnName, columnval) {

        var str = `select \* from ${table} where ${columnName} = '${columnval}'`

        var res = await sqlQuery(str)

        return res.length > 0 ? true : false

    }

4.5.2 登录

输入账户名密码，后端进行验证。验证完成之后即可返回用户信息。前端会将用户id保存至session中，做持久化存储。将其余的用户信息保存如vuex的store中，方便随时调用。同时触发socket事件，更新数据库中该用户被分配的socketid，并设置登录状态，每次关闭页面或者主动注销时就会清除vuex中的信息和session中的userid。在session中保存userid的作用就在于，当页面刷新，就会根据sessionid重新给后端发出请求，重新获得用户数据。

用户信息保存实现：

            sessionStorage.setItem(

                  "myid",

                  JSON.stringify(res.data.data.userid)

                );

                socket.emit("setsocket", sessionStorage.getItem("myid"));

                getUser(sessionStorage.getItem("myid")).then((res) => {

                  this.$store.commit("setme", res.data.data);

                }).then(() => {

                  filtrouters()

                });

更新数据库信息:

socket.on('setsocket', async function (userid) {

            console.log(socket.id + '登录了');

            //修改数据库登录信息(socketid,isonline)

            //设置数据库更新语句

            let sqlStr = "update user set socketid=?,isonline=? where id=?"

            let result = await sqlQuery(sqlStr, [socket.id, 1, userid])

        })

4.6新车与二手车选购模块

4.6.1汽车筛选

在选购页面，默认按照价格降序展示车辆。但根据多条件选择器可以选择出用户需要的车辆。多条件选择器首先可以通过首字母选择品牌，还有车型，价格，名字来进行选择。我通过一个数组来保存用户设置的条件，分别是品牌，价格，车型，名字。这些条件都有默认值，每一次点击条件都会将这个条件数组完整的传回后端，后端根据条件进行拼接sql语句完成查询。

后端筛选实现：

 var str = `select \* from car where 1=1`

            var term = ''

            if (req.query.myscreen[0] != -1) {

                term += ` and car\_brandid = '${req.query.myscreen[0]}'`

            }

            if (req.query.myscreen[1] != -1) {

                term += ` and car\_price\_low >= ${req.query.myscreen[1]} and car\_price\_high <= ${req.query.myscreen[2]}`

            }

            if (req.query.myscreen[3] != -1) {

                term += ` and car\_type = '${req.query.myscreen[3]}'`

            }

            if (req.query.searchstr != '') {

                term += ` and car\_name like '%${req.query.searchstr}%'`

            }

            term += ` order by id desc limit ${req.query.pagenum},${req.query.reqnum}`

            var sqlres = await sqlQuery(str + term)

4.6.2交易

交易模块需要用户提交购车表单，其中包含用户支付密码以及选购的车辆信息。信息传回后端会进行支付密码以及余额的验证。全都验证通过后完成整个交易流程。

//获取车辆信息

            let carsql = await sqlQuery(`select \* from car where id=${req.body.carid}`)

            let car = carsql[0]

            //支付密码验证

            let pwdcheck = await modusers.pwdcheck(req.body.userid, req.body.paypwd)

            if (!pwdcheck.check) {

                return res.json({

                    code: 20001,

                    message: "密码验证失败",

                })

            }

            //账户余额验证

            let balancecheck = await modusers.balancecheck(car.car\_price\_low, req.body.userid)

            if (!balancecheck.check) {

                return res.json({

                    code: 20001,

                    message: "账户余额不足",

                })

            }

            //账户扣费

            modusers.deduct(car.car\_price\_low, req.body.userid)

            //购买信息计入订单表

            //会返回一个插入sql对象,其中inserId为插入行的id

            var ordersql = await sqlQuery(`insert into car\_order (carid,userid,order\_time,order\_type,sell\_id)

            values (${car.id},${req.body.userid},'${new Date()}',0,${car.car\_brandid})`)

            var car\_number = "川E 456E06"

            //汽车入库

            var barnsql = await sqlQuery(`insert into barn (carid,userid,car\_number,km,orderid)

            values (${car.id},${req.body.userid},'${car\_number}',0,${ordersql.insertId})`)

            return res.json({

                code: 20000,

                message: "购买成功",

                data: {

                    car\_number: car\_number,

                    orderid: ordersql.insertId,

                    car: car

                }

            })

4.7 车库模块

在库的车辆提供出售服务。汽车有在库/挂售状态。用户若要卖出车辆，需要设置定价以及输入自己的支付密码，完成验证后，即可将车辆挂售。挂售的车辆会出现在二手车市场。后台对挂售的主要操作是，更改车辆状态。

在其他用户购买此车完成支付后，当完成交易，会进行金额的转移和车辆所属人的更新。

4.8 车友会模块

帖子与评论的实现类似，帖子使用了富文本，意味着用户可以在输入框中贴图，帖子与评论的提交会顺带提交用户信息，以及发布帖子评论的位置，后台就会进行插入数据的操作，记录发布者，发布时间以及内容。帖子还会区分为车友会评论还是资讯评论。

4.9 资讯管理

资讯管理主要是由超级管理员负责。有新增，编辑，删除三个功能。引用了element的表单组件进行快捷展示。新增或者编辑时对每个字段都有相应的规则进行文本校验，只有当校验列表都符合要求，才能向后端发出请求。比如，资讯的展示图必须要求是jpg格式，且有大小要求限制，每一个资讯都要提供资讯名，展示图，类型，内容四个部分。

4.10 汽车管理

汽车的管理主要是由商户进行负责，发布人在新增/编辑车辆时同样需要提供车名，展示图等相关信息，发布时不但会将表单内容提交至后台，同时会将发布人所属的公司id一起提交，按照公司id来决定车辆所属品牌，因此汽车管理模块只有商户能够进入，因为另外的身份没有公司id。

4.11 个人中心模块

4.12.1 头像上传

头像上传主要涉及的知识点是图片的上传。前端的上传主要是使用的element的图片上传组件。在获取到文件之后，将需要保存的地址一起发送给后端。后端有文件上传接口，在获取到邮件之后根据保存地址来保存图片。在后台完成文件的保存之后会告诉前端文件的真实保存路径，前端再次发出请求完成用户信息修改，也就是将用户的头像地址进行更新。

文件上传接口实现：

    async uploadfile(req, res, next) {

        try {

            // console.log(req.body.path);

            var des\_file = `/www/server/tomcat/webapps/ROOT/guloti/GulotiResource/upload/${req.body.path}`

            // res.json(req.body)

            // var des\_file = "./upload/" + req.files[0].originalname;

            fs.readFile(req.files[0].path, function (err, data) {

                fs.writeFile(des\_file, data, function (err) {

                    if (err) {

                        console.log(err);

                        fs.unlinkSync(req.files[0].path);

                    } else {

                        var response = {

                            message: 'File uploaded successfully',

                            url: `http://47.108.230.246:8080/guloti/GulotiResource/upload/${req.body.path}`

                        };

                        // console.log(response);

                        fs.unlinkSync(req.files[0].path);

                        res.json(response)

                    }

                });

            });

        } catch (error) {

            next(error)

        }

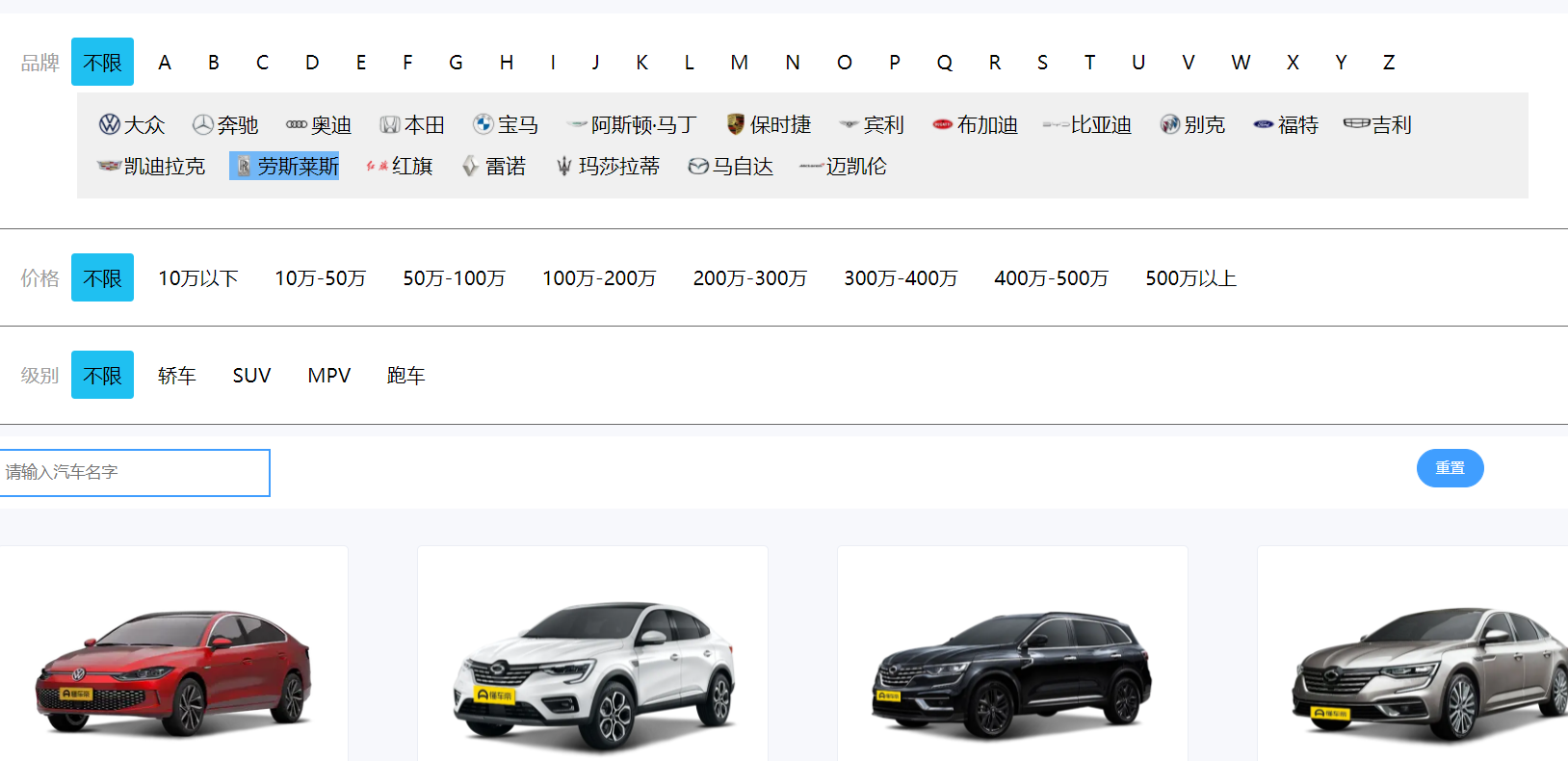
    }

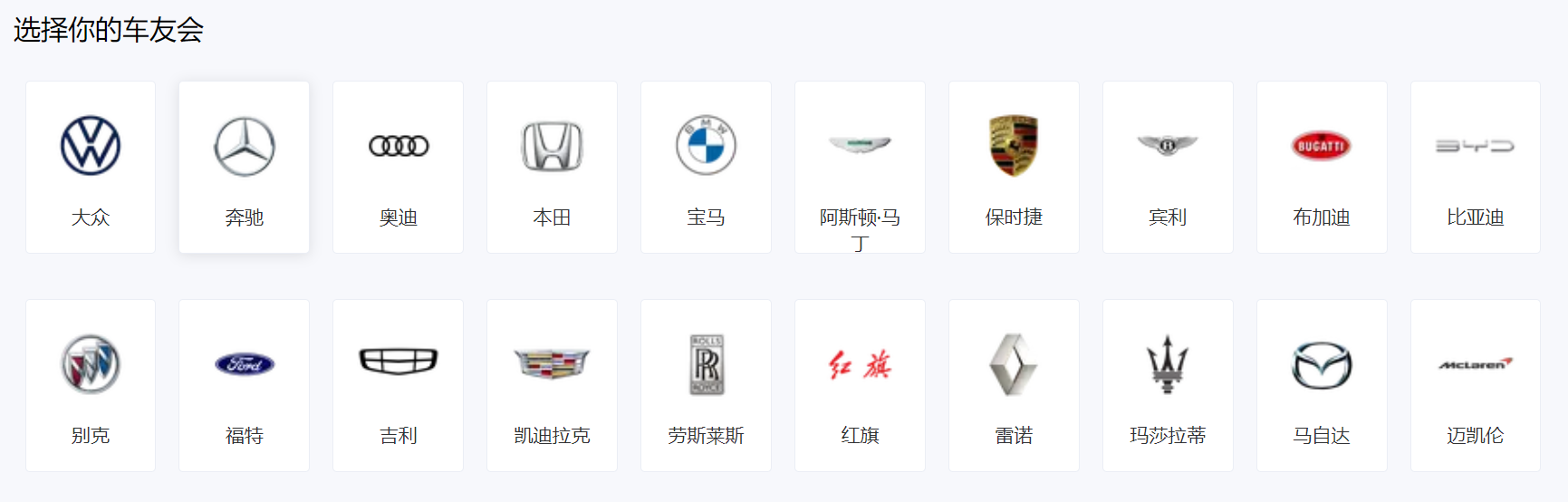
4.12.2 注销

注销功能的流程与关闭页面的轮流一样，根据session中的用户id清除用户数据库中的socketid，清除vuex里的用户信息，最后完成退出页面完成注销。

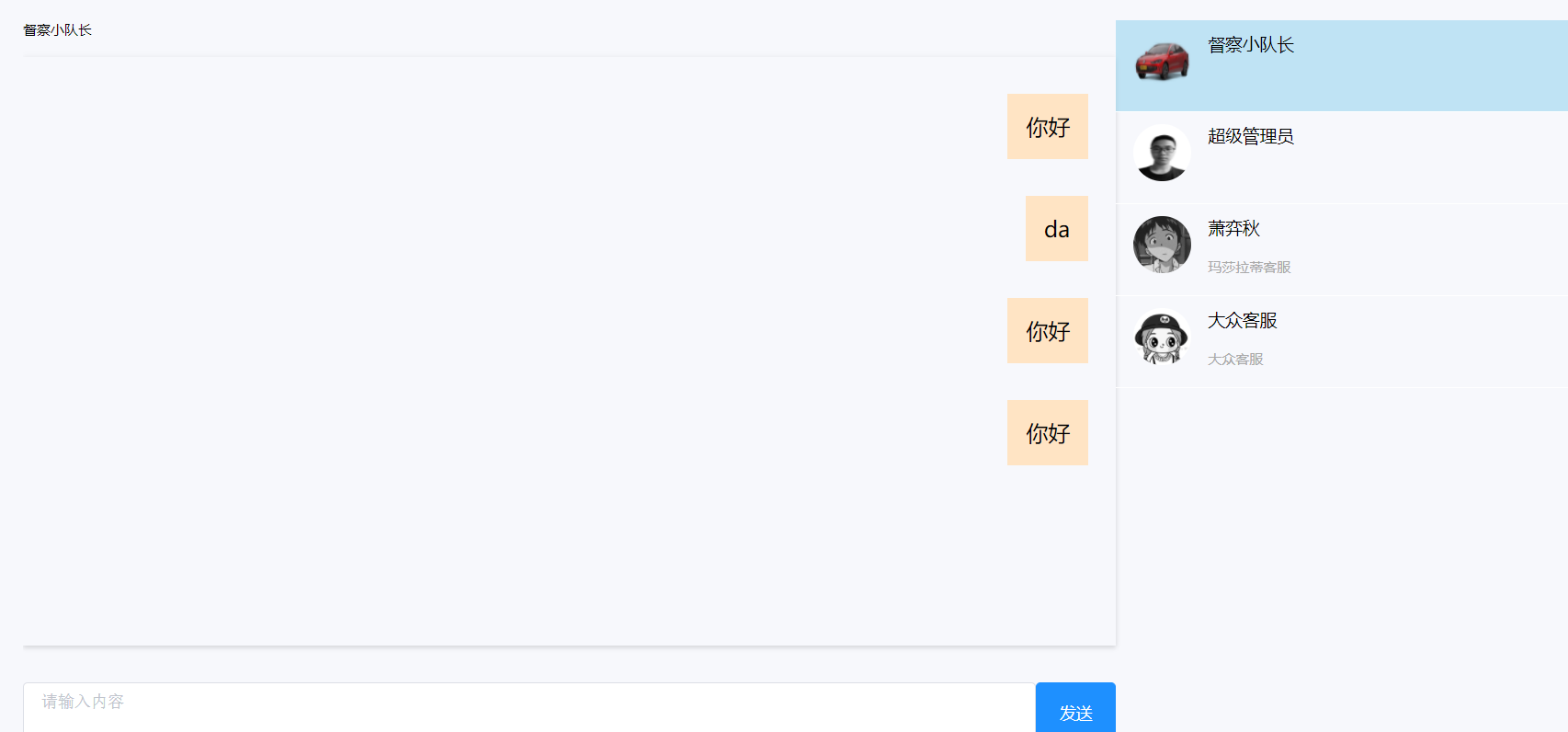
5功能展示



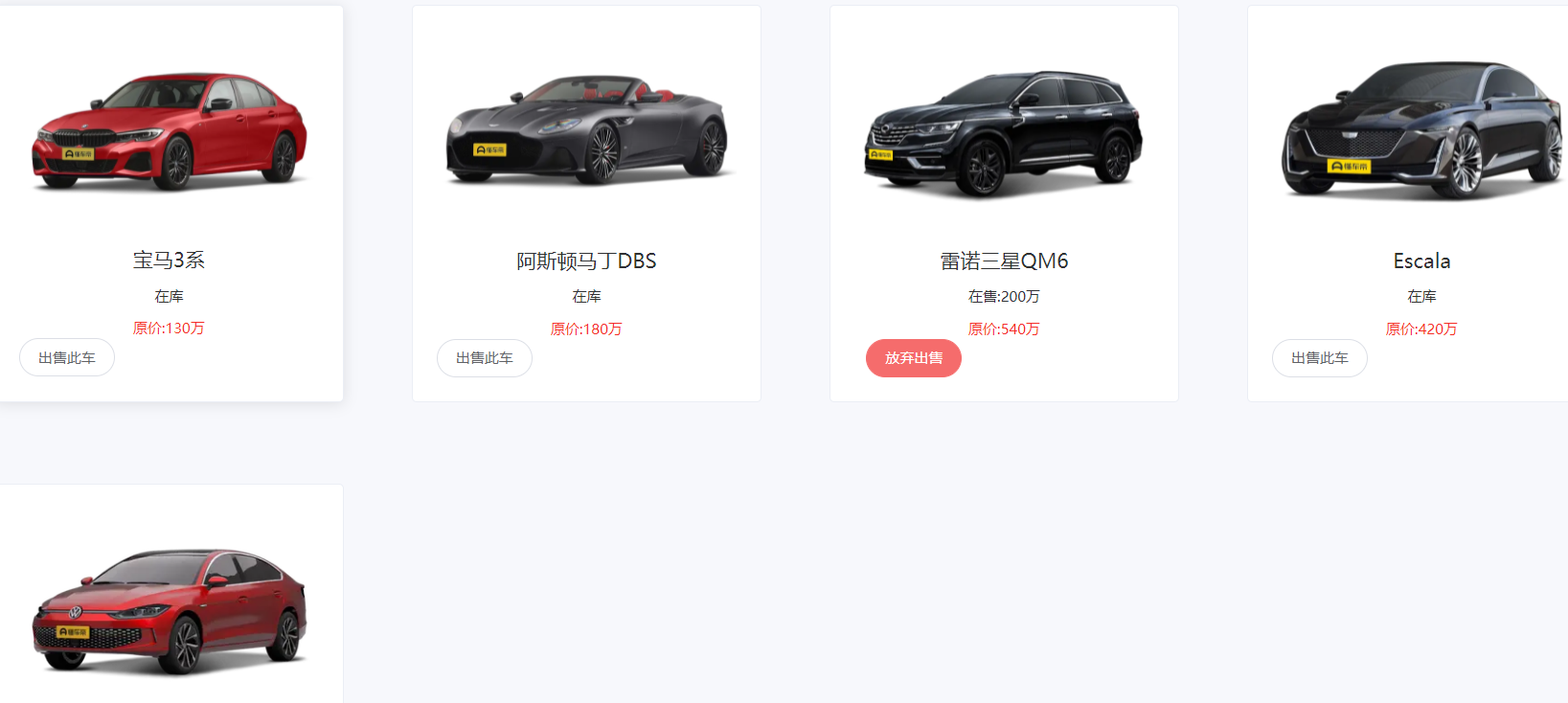
















6 功能总结

结 语

参考文献

致 谢

声 明