Lab2 实验报告

周训哲 20307110315　　柏露 20307130208
张智雄 20307130152　　陈实立 20307110078

2023 年 4 月 6 日

分工

周训哲负责后端部分：最开始参与数据的设计工作，并初拟了一个 db 的表；完成了用户登录后展示商店的后端到前端的数据传输，商店主页的后端到前端的数据传输，管理员查看和批准请求的前后端的数据传输，以及商户申请开店的数据格式审核和数据写入数据库的工作；最后还负责测试前后端联合的结果，并及时修复前后端联合的 bug，以及提出部分网页设计改进。

张智雄负责后端部分：完成了登陆，管理员登陆，注册以及数据库设计。

陈实立负责前端部分：完成了用户登录后展示商店的页面，商店的详情页面，以及管理员查看和批准请求的界面；还完成了前端向后端发送数据以及从接收数据的工作，同时负责维护用户登录后在前端存储的信息。

柏露负责前端部分：完成了登录，管理员登录，注册以及开店申请的页面以及表单的设计。

实验设计

一、数据库设计

1. 用户表

用户表有 id_num(身份证号)，username(用户名)，phonenumber(手机号)，email(电子邮件)，password(密码)，is_shop(是否为商户) 几个属性。其中用户名和邮箱用变长字符串存储，is_shop 用 bool 类型，其余用定长字符串存储，id_num 为主键。

2. 管理员表

管理员表仅有 username(用户名) 和 password(密码) 两项组成，username 为主键，将普通用户和管理员分两个表是为了减少冗余。

3. 商店表

商店由 shop_id(商店编号),kind(商品类型),shopname(店铺名称),id_num(店主身份证号)，intro(店铺简介)，address(店铺地址)，register_capital(注册资本)，register_date(注册日期)，is_open(是否被批准开店)，is_check(管理员是否处理)。其中 shop_id 为主键，id_num 为外键用于关联用户表

4. 商品表

商品表由 good_id(商品编号), shop_id(商店编号)，goodname(商品名)，price(价格)，amount(数量)。其中商品编号为主键，shop_id 为外键用于关联商店表

二、后端功能设计

1. 字段验证

为了保证后端数据库的安全，首先设计了 checker 模块用于验证收到的各个字段是否合法，主要使用 python 正则表达式进行校验。

2. 功能设计

(1) 注册功能 注册功能将从前端收到的字段进行校验之后插入数据库即可，根据成功失败返回给前端相应信息。

(2) 登陆功能 登陆功能将前端收到的账号密码与数据库中的账号密码进行比对，这里使用 sha256 算法同时进行加盐计算用于保存密码。若成功生成一个 token 返回给前端用于维护登陆状态，若失败则返回对应信息。

(3) 管理员登陆功能 管理员登陆功能和普通用户登陆功能一样。

(4) 商户开店 使用 ‘request.get_json()‘获取前端传输过来的数据，并且在后端使用 ‘json‘格式进行存储，首先判断当前申请开店的用户是否是商户，调用 ‘json_data[’is_shop’]‘，如果不是则返回错误信息。然后对填写信息进行逐个判断输入格式是否合规。由于申请开店时输入内容基本都是前端传入后端的字符串，故主要使用 ‘re.match()‘进行判断。最后使用数据库将用户信息进行存储。

(5) 管理员查看开店申请 主要就是相当于实现商店的展示功能，筛选所有商店信息并将信息打包成 json 数据传输给前端。

(6) 管理员批准开店请求 获取前端传输过来的批准/拒绝信息，以及当前处理的商店的主键信息，首先对当前商店进行查询，如果发现审批状态为已经审批，则报错，以防止重复审批。否则识别审批信息，对是否允许开店的元数据进行更新，并且更新是否审批的元数据。

(7) 主页商店展示 类似管理员查看开店申请，主页展示的商店信息只包含已经审批并且通过的所有商店的信息，最后返回这些商店的所有数据到前端。

(8) 商店主页 使用 ‘GET‘请求获取当前网页发出的请求，然后使用 ‘request.args.to_dict()‘将请求的信息转换为数据，并且作为商店的索引信息进行数据库筛选，最后返回该商店的所有信息至前端。

三、路由设计

我们将整个应用设计为单页面应用，为了更好的统一页面布局和复用代码，我们采用层次化路由的方式：

```
13  const routes = [
14    {
15      path: '/',
16      component: Layout,
17      children:[
18        { path: '/', component: Home },
19        { path: '/shop/:id', component: ShopPage },
20        { path: '/admin', component: AdminPage },
21
22      ]
23    },
24    { path: '/login', component: LoginView },
25    { path: '/signin', component: SigninView },
26    { path: '/adminlogin', component:
      AdminLoginView },
27    { path: '/setupshop', component:
      SetupShopApplyView },
28  ]
```

其中，主页，商店详情页，管理员审批页面为 2 级路由，他们都采用了 Layout 组件的布局，并复用了导航栏的代码，提高了页面的一致性，并减少了重复代码。

四、less 的自动化导入

为了复用前端的 css 代码，我们选择对常用的样式进行包装，并利用 style-resoures-loader 插件来完成自动注入到每个 less 文件或者 vue 组件中 style 标签中（无需每次手动引入）。

```
  pluginOptions: {
    'style-resources-loader': {
      preProcessor: 'less',
      patterns: [
        // 自动引入 less
        path.join(__dirname, './src/assets/styles/variables.less'),
        path.join(__dirname, './src/assets/styles/mixins.less')
      ]
    }
  },
```

五、登录状态的维护

我们使用 vuex 对用户的登录状态进行维护，并将该状态持久化保存（以免用户每次登录需要重新输入密码）。持久化部分我们使用 vuex-persistedstate 插件。

```
import { createStore } from 'vuex'
// 导入 vuex 持久化存储
import createPersistedState from "vuex-persistedstate";

import user from "@/store/user";

export default createStore({
  modules: {
    user,
  },
  plugins: [
    // 配置持久化数据存储
    createPersistedState({
      key: "Web_Project/frontend/vue3",
      paths: ["user"],
    }),
  ],
});

```

对于 vuex 存储的修改，我们全部采用同步的 mutation 来实现，并对用到的功能进行了封装。

```
19    mutations: {
20      setUser(state, payload) {
21        state.profile = payload;
22      },
23      setToken(state, payload = "") {
24        state.profile.token = payload;
25      },
26      // 设置重定向地址
27      setRedirectURL(state, payload = "") {
28        state.redirectURL = payload;
29      },
30    },
```

其中，主页，商店详情页，管理员审批页面为 2 级路由，他们都采用了 Layout 组件的布局，并复用了导航栏的代码，提高了页面的一致性，并减少了重复代码。

六、请求工具

请求工具我们使用 axios ，我们在 axios 的基础上对我们可能用到的功能进行了进一步的封装（详情见 request.js ）。通过我们的封装，我们可以方便地实现请求的发送，并可以实现在请求头中的 Authorization 字段携带 token。

七、业务逻辑设计

登录功能

- 登录：

1. 将用户名和密码发送给后端

2. 根据后端返回的结果进行判断：

- 成功：将返回的用户信息存储在 vuex 中，跳转到主页，提示登录成功

- 失败：提示登录失败

- 请求发生异常，提示请求异常
- 退出登录：
清空 vuex 中存储的用户信息，并跳转到登录界面

管理员批准开店申请

1. 将商店的 id 发送给后端
2. 根据后端返回的结果进行判断，成功或失败进行相应的处理
3. 重新请求申请列表并刷新界面

注册，申请开店等逻辑较为简单，不做详细介绍

注册功能

注册的表单由相应的需要的信息的表单项构成，除了用户的类型外的表单项由下拉框组成，其他的表单项都是输入框。

对于注册表单用 formSignin 来储存用户输入的数据，用 ruleSignin 来验证用户的输入是否合法，在使用时需要将 formSignin 绑定到表单的 model 上，将 ruleSignin 绑定在 rules 上。表单的每一项的 prop 对应相应的规则，表单项中具体的容器用 v-model 绑定 formSignin 中对应的项。

表单的验证主要靠正则表达式和长度限制完成。对于用户名，电话，邮箱以及身份证的验证使用了网上的正则表达式。密码的验证的正则表达式先让纯数字，字母和特殊字符的密码不可用然后允许有数字，字母和特殊字符的密码达到验证包含两类字符的密码的目的。对于确认密码的验证使用了自己写的函数来验证，先判断是否为空，在判断是否和第一次的密码相同。

表单最下面是提交按钮，该按钮绑定了提交注册信息的函数，该函数先判断所有的表单项是否合法，若合法则提交信息，否则拒绝提交。

登录功能

登录的表单较为简单，复用了注册中用于验证了用户名和密码的规则，在表单的左下角有切换普通用户登录和管理员登录的按钮。

申请开店功能

申请开店的表单较为简单，没有复杂的验证规则，使用了表单中用于提交文本和时间的容器。对于注册资金的验证使用了单独的函数，该函数将字符串转换为浮点数进行比较。

问题及解决方案

普通用户登录

在测试过程中，发现对于管理员来说，可以实现登录功能，但是对于普通用户来说，则出现登录之后的状态仍然和未登录状态相同，与前端同学协调研讨之后发现后端数据可以成功传入前端，但是前端的数据没有传入后端，协商后前端同学修改以后解决该问题。

开店失败

对于开店过程中填写表单后，发现即使数据类型填写无误，后端仍然无法通过，进行调试后发现前后端之间传输的时间类型不同，后端对时间数据进行截取之后解决该问题。

商店主页展示

在商店主页展示环节，出现当前商店信息无法获取的问题，原因是前端同学一开始没有将商店主键放入对象数据中，而是在网页请求中，在协调之后，采用 ‘request.args.to_dict()‘之后能够成功获取前端信息。

对于 flask 框架蓝图的使用有问题

开始在网上查询到 flask 模块化使用蓝图进行开发，但在最后的测试中却出现找不到对应路由的状况，最后查询了 flask 官方文档才得以解决

前后端连接问题

前后端连接时遇到了跨域问题，导致后端一直无法收到前端传来的数据。后来通过浏览器的开发者模式和查询 flask 官方文档才解决这一问题。

前端技术细节问题

出现警告 ‘[Vue 3 - inject() can only be used inside setup or functional components]‘ ，而且 vuex 的 ‘useStore()‘ 相关的方法产生异常

出错原因 ‘useStore‘ 和 ‘useRouter‘ 只能在 setup 中使用

错误代码：

```
<script setup>
import { ref, reactive } from 'vue'
import { loginByAccountAndPassword } from "@/api/user";
import { useRouter } from "vue-router";
import { useStore } from "vuex";
import Message from "@/components/library/Message";

const submitLoginForm = (formEl) => {
        if (!formEl) return
        formEl.validate((valid) => {
                if (valid) {
                        console.log('submit!')
                        // here is the problem
                        //-------------------------
                        const router = useRouter();
                        const store = useStore();
                        //-------------------------
                        ...

```

改正：

```
<script setup>
import { ref, reactive } from 'vue'
import { loginByAccountAndPassword } from "@/api/user";
import { useRouter } from "vue-router";
import { useStore } from "vuex";
import Message from "@/components/library/Message";

// put it in the component directly
//-------------------------
const router = useRouter();
const store = useStore();
//-------------------------

const submitLoginForm = (formEl) => {
        if (!formEl) return
        formEl.validate((valid) => {
                if (valid) {
                        console.log('submit!')
                        ...
```

需要在 html 中 ‘@click‘ 绑定一个函数，结果告诉我说 ‘[Function is declared but its value is never read‘ ，错误代码如下：

```
export default {
        name: "AdminPage",
        setup() {
                const shopList = ref([])
                getOpenRequest().then(data => {
                        shopList.value = data.result
                })
```

```
                // this function
                const approveOpenShopReq = (shop_id) => {
                        approveOpenShop({shop_id, approval:'true'})
                        getOpenRequest().then(data => {
                                shopList.value = data.result
                        })
                }
                // return { shopList } wrong line
                return { shopList, approveOpenShopReq } wrong line
        },
};
```

‘setup‘ 中定义的函数需要 ‘return‘ 出来
‘setup‘ 中使用路由参数

```
<script>
import { ref } from 'vue'
import { getShopDetails } from '@/api/home'
import { useRoute } from 'vue-router'
export default {
        name: "ShopPage",
        components: {
        },
        setup() {
                const shop = ref([])
                const route = useRoute()
                // route.params.id 就是路由中的参数 id
                // 对应 js 中的 { path: '/shop/:id', component: ShopPage },
                getShopDetails(route.params.id).then(data => {
                        shop.value = data
                })
```

```
17
18                  return { shop };
19          },
20  };
21  </script>
```

后端将返回结果传给前端时，约定的 json 对象变为字符串。

错误原因：返回结果中含有 json 对象的嵌套，后端对两层 json 对象都使用了 json.dumps，前端收到结果后会去掉最外层的 json 对象，但是内层 json 对象仍然是字符串的形式。

解决方法：内层的 json 对象不需要调用 json.dumps 即可。

代码检查结果以及测试截屏

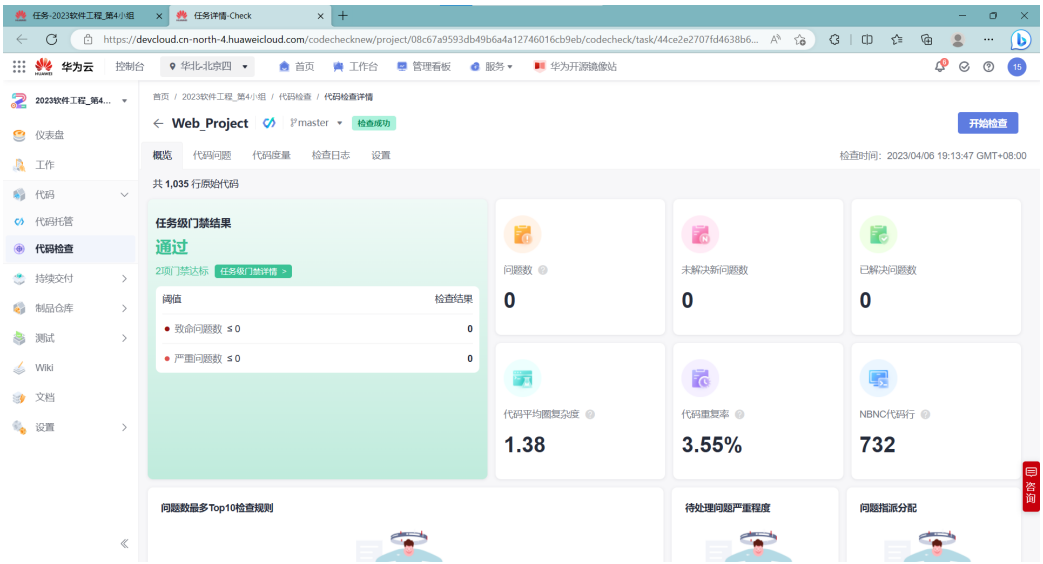

图 1: 代码检查结果

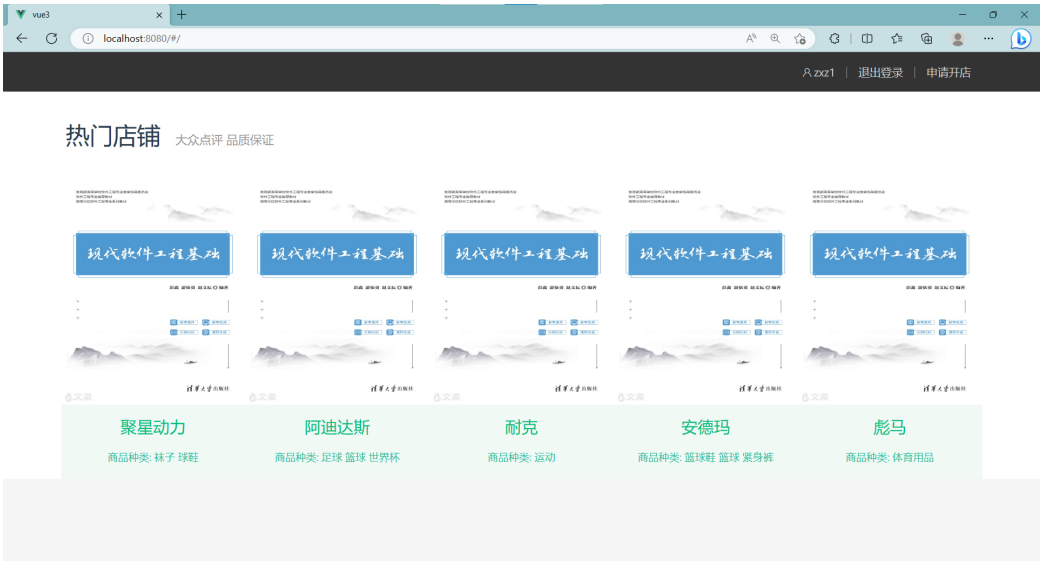

图 2: 主页

图 3: 商店页面展示

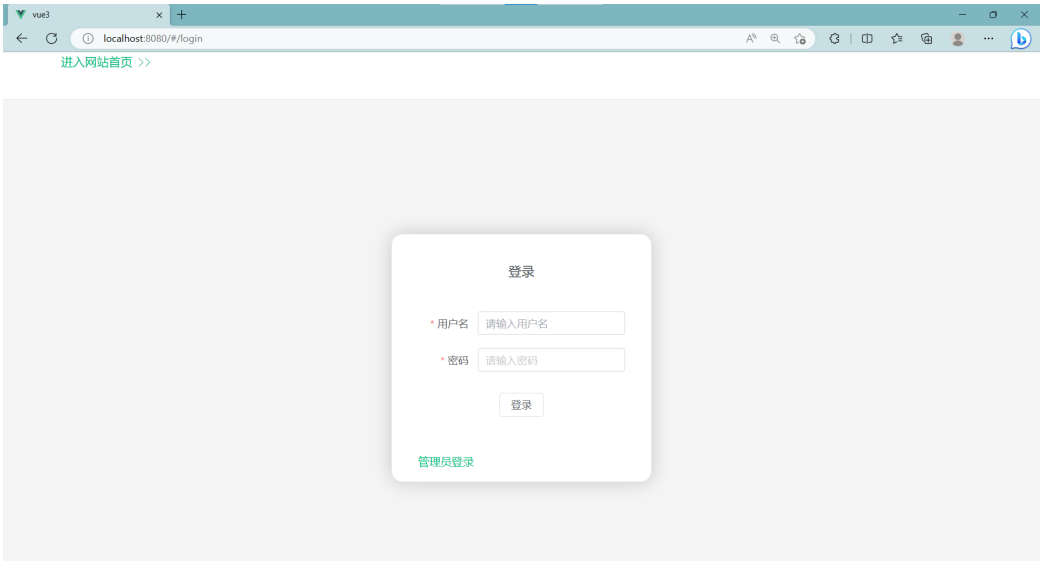

图 4: 登录界面

图 5: 注册界面

图 6: 开店界面

图 7: 管理员查看开店申请界面

心得体会

周训哲　这是我第一次接触 flask 框架模型，也是我第一次接触前后端分离的 web 应用，对于我来说首先需要熟悉 flask 框架，同时需要了解前后端分离的数据传输这些花掉了大部分的实验时间。

同时这也是我第一次团队协同合作完成代码任务。我们团队的沟通并不高效，首先是对前后端协调的 api 接口的网站应用不够熟练，导致实验过程中多次修改 api 接口，使得前后端协调并不流畅。我们应该在开始写代码前集中讨论，制定接口，在开发的过程中尽量不修改接口。

张智雄　通过这次实验，我了解了 flask 框架的使用，mysql 数据库的使用。同时学会了如何使用一些 api 工具进行合作开发和测试。

当然在实验过程中也遇到了许多问题，主要是成员之间沟通不充分，不及时导致合作效率很低，甚至前后端之间的 api 接口频繁修改，通过这次实验我也学习到了该怎样才能进行有效的沟通，高效的团队合作。

陈实立　这是第一次软工实验，也是我第一次接触前端代码，对于我来说前端的工作有一定难度，我也花了比较长的时间来完成这次实验。

我们团队的沟通并不高效，在实验进行的过程中接口发生了多次修改，这拉低了我们团队的开发效率。我们应该在开始写代码前集中讨论，制定接口，在开发的过程中尽量不修改接口。

此外，我们前端的完成度也不够高，希望可以在后面的实验中有更高的代码完成度。

柏露　这是第一次软工实验，也是我第一次接触前端代码，对于我来说前端的工作有一定难度，我也花了比较长的时间来完成这次实验。

我们团队的沟通并不高效，在实验进行的过程中接口发生了多次修改，这拉低了我们团队的开发效率。我们应该在开始写代码前集中讨论，制定接口，在开发的过程中尽量不修改接口。

此外，我们前端的完成度也不够高，希望可以在后面的实验中有更高的代码完成度。