

## 仿 www.booking.com 网站

### 一、概论

#### 1.1. 课题目标:

在大学生涯的最后一次小组实训中，我们组选择了模仿 [www.booking.com](http://www.booking.com) 网站作为课题。在这次实践中，我首次接触并成功完成了一个前后端分离的 Web 应用项目，深刻体会到了前后端协作的魅力。我搭建了复杂的前端 Vue 界面，定义了 api 交互接口以及后台逻辑，并设计了数据库表结构。

作为项目组内唯一的推荐生，我的任务主要分为两点：

(1) 任务 1: 担任组长，确保团队高效完成实训的全部内容。

(2) 任务 2: 尽自己所能承担项目大部分内容，以争取更多时间给组员备考研究生入学。

为了完成上述两点任务，我设定了以下目标，以掌握相关技能和知识：

- ❖ 数据库层面：学习 MySQL 数据库操作，掌握 JDBC 连接数据库的方法，并了解阿里云 OSS 的使用。
- ❖ 后端层面：熟悉 JavaSE 编程语言，掌握 SpringBoot 3.1.5 框架和 MyBatis 持久性框架，Maven 项目管理工具，以便高效地实现后端逻辑。
- ❖ 前端层面：学习 Node.js，熟悉 Vue2.0 框架，Vue CLI 脚手架，Element-ui 桌面端组件库，同时深入了解 HTML、CSS、JavaScript 等前端基础知识。
- ❖ 理论层面：了解和应用后端接口设计的三层结构，包括控制层、服务层和数据层，以及数据库设计的相关知识。
- ❖ 工程实践：学习并应用软件工程的最佳实践，包括编写 API 接口文档、代码版本控制（如 Git）、团队协作等，以确保项目的高质量完成。

通过不断学习和实践，我致力于成为一个全面的开发者，同时通过团队协作，确保项目的顺利进行。我相信这次实训将为我的软件开发和团队管理能力提供宝贵的经验。

#### 1.2. 任务分析:

在我所负责的方面，主要面临如下任务：

- 任务 1: 分析和理解 Booking.com 网站的 HomeView 主界面。
- 任务 2: 分析和理解 Booking.com 网站的注册及登录功能的实现。
- 任务 3: 开发前端界面，利用组件化设计的思想，将 HomeView 主页面拆分为 13 个组件，分别命名为: Comp1, Comp2, Comp3, ...Comp13, 并实现用户交互功能。
- 任务 4: 开发注册和登录功能，利用 Vue.js 框架自带的 Router 路由以及后端程序实现界面跳转、用户注册、用户登录等功能。

- 任务 5: 开发后端界面, 基于 SpringBoot 3.1.5 框架和 MyBatis 持久性框架实现响应前端请求数据、数据库链接、用户数据插入数据库、图片持久化存储等功能
- 任务 6: 编写接口请求 api 文档, 实验报告和其他相关文档, 记录实验过程、设计决策、实现细节和测试结果。

### 1.3. 内容描述:

(1) 在任务 1 “分析和理解 Booking.com 网站的 HomeView 主界面。”中, 主要包含如下内容:

- 使用浏览器开发者工具、查看源代码, 分析页面结构和 CSS 样式, 理解前端实现细节。
- 详细研究 Booking.com 网站的主界面, 了解其布局、样式、交互设计等, 确定需要模仿的主要界面和功能元素。

(2) 在任务 2 “分析和理解 Booking.com 网站的注册及登录功能的实现。”中, 主要包含如下内容:

- 深入研究 Booking.com 的用户注册和登录功能, 包括界面设计、数据验证、错误处理等。
- 通过注册和登录流程, 网络请求、设计前端代码和后端接口, 理解整个流程。

(3) 在任务 3 “开发前端界面, 利用组件化设计的思想, 将 HomeView 主页面拆分为 13 个组件, 分别命名为: Comp1, Comp2, Comp3, ...Comp13, 并实现用户交互功能。”中, 主要包含如下内容:

- 利用 Vue.js 框架, 将 HomeView 主页面拆分为 13 个组件, 分别命名为 Comp1 至 Comp13。
- 使用 Vue 组件系统, 根据功能和 UI 元素将页面拆分, 确保每个组件负责单一的功能, 方便维护和复用。

(4) 在任务 4 “开发注册和登录功能, 利用 Vue.js 框架自带的 Router 路由以及后端程序实现界面跳转、用户注册、用户登录等功能。”中, 主要包含如下内容:

- 在每个组件中实现用户交互功能, 确保页面具有良好的用户体验。
- 使用 Vue.js 的事件处理机制、状态管理 (Vuex) 等功能, 实现用户的交互需求。
- 使用 Vue.js 框架的 Router 路由实现用户注册、登录等功能, 确保页面跳转和用户认证的正常流程。

(5) 在任务 5 “开发后端界面, 基于 SpringBoot 3.1.5 框架和 MyBatis 持久性框架实现响应前端请求数据、数据库链接、用户数据插入数据库、图片持久化存储等功能”中, 主要包含如下内容:

- 使用 SpringBoot 和 MyBatis 框架实现后端逻辑，响应前端请求并进行数据库操作，包括用户数据插入和图片存储。
- 创建相应的 Controller 处理前端请求，配置 MyBatis 连接数据库，实现数据库操作和文件上传等功能。

（6）在任务 6 “编写接口请求 api 文档，实验报告和其他相关文档，记录实验过程、设计决策、实现细节和测试结果。”中，主要包含如下内容：

- API 接口文档：描述前后端接口的使用方法、参数、返回结果等。
- 实验报告：见本文。
- 其他文档：如数据库设计文档、前端组件文档等，以便后续维护和团队协作。

## 二、设计与实现

### 2.1.程序的整体架构

本项目为“前后端分离”架构，故代码中整体由“前端代码”和“后端代码”代码组成。其中前端代码使用“Vue.js 手脚架”生成代码基本框架，后端代码使用“IDEA”生成 Maven 结构的后端代码框架。

（1）前端代码架构：

- views 目录： 存放页面级别的 Vue 组件，通常每个组件对应一个页面。

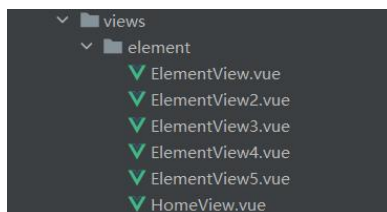


图 1 views 目录

- components 目录： 存放 Vue 组件文件，主要包括主界面 HomeView 的各个 Comp 组件以及子组件 SubComp。

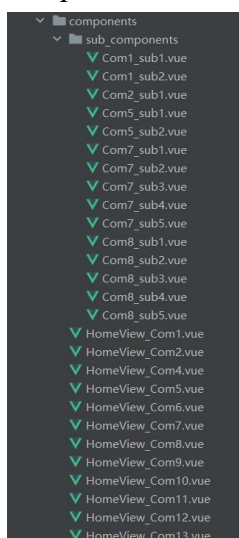


图 2 components 目录

- router 目录： 存放 Vue Router 相关的配置文件，定义路由信息。

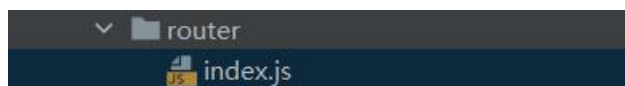


图 3 router 目录

- assets 目录： 通常存放静态资源文件，如图片、字体等。但是由于本项目的  
所有图片均存储在阿里云 OSS，因此 assets 目录为空。
- node\_modules： 存放项目依赖的 Node.js 模块，包括 axios 模块等。

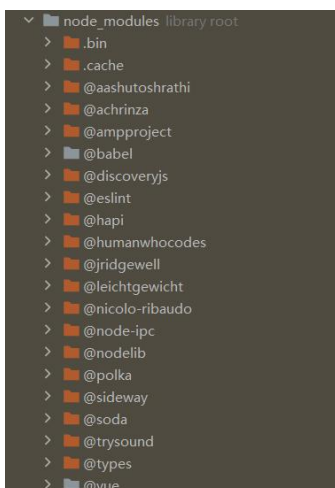


图 4 node\_modules

- App.vue： 项目的根组件，整个应用的入口。
- main.js： 项目的主入口文件，用于初始化 Vue 实例，引入各种插件和配置。



图 5 App.vue 和 main.js

## (2) 后端代码架构：

- 源代码： 包含主应用程序类 MyProjectApplication，Control 控制类，Mapper 类，Service 接口及其实现类，Pojo 实体类。

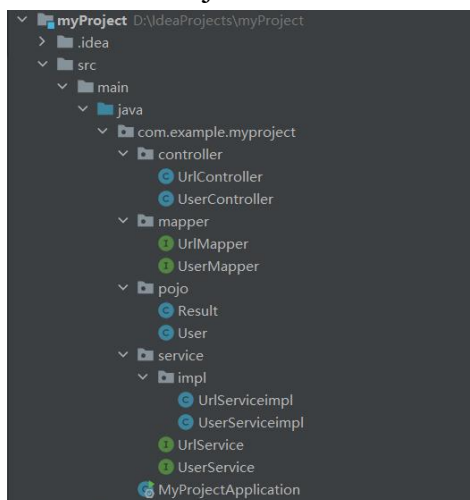


图 6 后端源代码

- 静态资源：包括 MarkDown 格式的 api 接口文档。

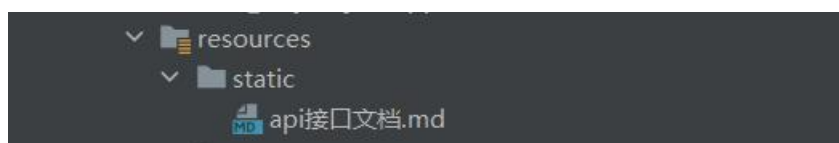


图 7 后端静态资源

- pom.xml 文件：Maven 项目的配置文件，包含了项目的依赖、插件、构建配置等信息。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <!-- 父类项目 -->
6      <parent...>
7      <!-- 版本控制, 项目名称, 项目描述 -->
12     <groupId>com.example</groupId>
13     <artifactId>myProject</artifactId>
14     <version>0.0.1-SNAPSHOT</version>
15     <name>myProject</name>
16     <description>myProject</description>
17     <properties...>
21     <!-- 各种依赖项 -->
22     <dependencies...>
80     <!-- 各种插件 -->
81     <build...>
87 </project>
    
```

图 8 pom.xml 文件

- application.properties：Spring Boot 应用的数据库配置，其中主要包括了数据源（DataSource）和 MyBatis 的配置。

```

1  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
2  spring.datasource.url=jdbc:mysql://localhost:3306/myproject
3  spring.datasource.username=root
4  spring.datasource.password=root
5  mybatis.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl
6  mybatis.configuration.map-underscore-to-camel-case=true
    
```

图 9 application.properties

## 2.2.主要功能的流程

### （1）功能一：用户注册

#### ➤ 前端（Vue.js）：

- 创建注册页面组件：在 Vue 项目中创建一个用于用户注册的页面组件，包括邮箱和密码的输入框、注册按钮等元素。
- 收集用户输入：使用 Vue 的数据绑定机制，收集用户在注册页面输入的邮箱和密码。
- 数据验证：验证用户提供的邮箱是否合法、密码是否符合要求。
- 发送注册请求：当用户点击注册按钮时，使用 Vue 的 HTTP 库（axios）向后端发送包含用户注册信息的请求。

#### ➤ 后端（Spring Boot + MyBatis）：

- 处理注册请求： 在后端，依次创建一个处理注册请求的 Mapper 层，Service 层和 Controller 层，接收前端发送的注册请求，提取用户的邮箱和密码信息。
- 存储用户信息： 将加密后的密码和用户邮箱存储到数据库中。
- 返回注册结果： 后端向前端返回注册成功或失败的消息。

## (2) 功能二：图片云存储

实现“图片云存储”功能，即将图片保存在阿里云 OSS 平台，并通过后端从数据库获取图片 URL 列表传递给前端，涉及前端（Vue.js）、后端（Spring Boot + MyBatis）、阿里云 OSS 等多个环节。以下是基本的实现流程：

### ➤ 图片上传至阿里云 OSS：

- 选择图片： 选择要上传的图片的本地路径
- 处理图片上传请求： 后端创建一个 Java 类，作为工具类。接收图片本地路径、OSS 密钥等数据，同时生成阿里云 OSS 的上传凭证。
- 调用 OSS SDK 上传： 使用阿里云 OSS 的 Java SDK，将接收到的图片文件上传到 OSS。
- 获取上传成功后的图片 URL： 上传成功后，获取阿里云 OSS 返回的图片 URL。
- 保存图片信息到数据库： 将上传成功的图片信息（界面名、序号、OSS 中的 URL 等）保存到 MySQL 数据库中。

### ➤ 图片获取与前端渲染：

#### 1. 前端（Vue.js）：

- 请求图片列表： 当前端渲染某个 Vue 界面时，前端向后端发送请求，请求该界面下的所有图片的 URL。
- 接收后端返回的图片 URL 列表： 后端根据界面名从数据库查询对应的图片 URL 列表，返回给前端。
- 渲染图片： 前端接收到图片 URL 列表后，遍历列表，在页面上渲染 img 元素，使用获取到的 URL 显示图片。

#### 2. 后端（Spring Boot + MyBatis）：

- 处理图片列表请求： 依次创建一个处理注册请求的 Mapper 层，Service 层和 Controller 层，接收前端传来的界面名，从数据库查询该界面下的所有图片 URL。
- 查询数据库： 使用 MyBatis 查询数据库，根据界面名获取该界面下的图片信息。
- 返回图片 URL 列表： 将获取到的图片 URL 列表返回给前端。

通过这个流程，可以实现图片的上传到阿里云 OSS，同时将图片信息保存在 MySQL 数据库中。在渲染 Vue 界面时，后端通过查询数据库返回相应的图片 URL

列表，前端根据 URL 列表将图片渲染到页面上。这样实现了前后端分离的图片云存储功能。

### (3) 功能二：用户登录

#### ➤ 前端 (Vue.js) :

- 创建登录页面组件：在 Vue 项目中创建一个用于用户登录的页面组件，包括邮箱和密码的输入框、登录按钮等元素。
- 收集用户输入：使用 Vue 的数据绑定机制，收集用户在登录页面输入的邮箱和密码。
- 发送登录请求：当用户点击登录按钮时，使用 Vue 的 HTTP 库 (axios) 向后端发送包含用户登录信息的请求。

#### ➤ 后端 (Spring Boot + MyBatis) :

- 处理登录请求：在后端，依次创建一个处理注册请求的 Mapper 层，Service 层和 Controller 层，接收前端发送的注册请求，提取用户的邮箱和密码信息。
- 验证用户信息：根据用户提供的邮箱查询数据库，比对数据库中存储的加密密码。
- 返回登录结果：如果验证成功，后端向前端返回登录成功的消息，并可能包含一些用户信息。如果验证失败，返回登录失败的消息。

## 2.3.主要数据结构的设计

本项目设计如下的主要数据结构，主要包括 Mysql 数据库两个表：Users 表和 Urls 表。这个表的设计分别用于存储用户的注册信息和项目中图片的相关信息。Users 表中存储用户的注册邮箱和密码，Urls 表中存储图片的所在界面名、在该界面的序号和图片的 URL。

### (1) Users 表

- 功能：存储用户的注册信息。
- 结构：

mailbox (唯一，用户注册邮箱)

password (存储的用户密码)

- 建表 SQL 语句：

```
CREATE TABLE users (
    mailbox VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
);
```

### (2) Urls 表：

- 功能：存储项目中图片的信息。
- 结构：

page(图片所在界面名)

name (图片在界面中的序号)

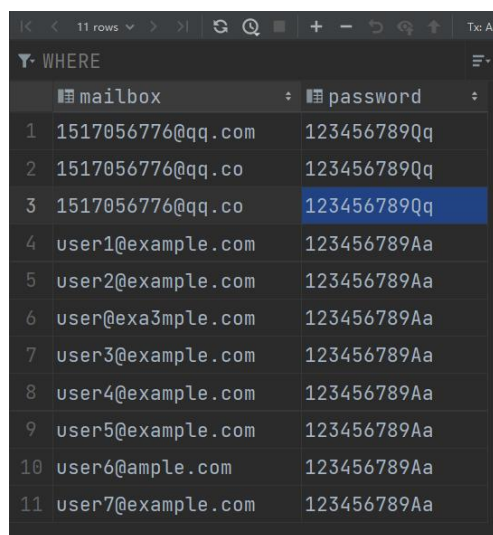
url (阿里云 OSS 的图片 URL)

● 建表 SQL 语句:

```
CREATE TABLE urls (
    page VARCHAR(255) NOT NULL,
    name INT NOT NULL,
    url VARCHAR(255) NOT NULL,
);
```

## 2.4.数据的设计

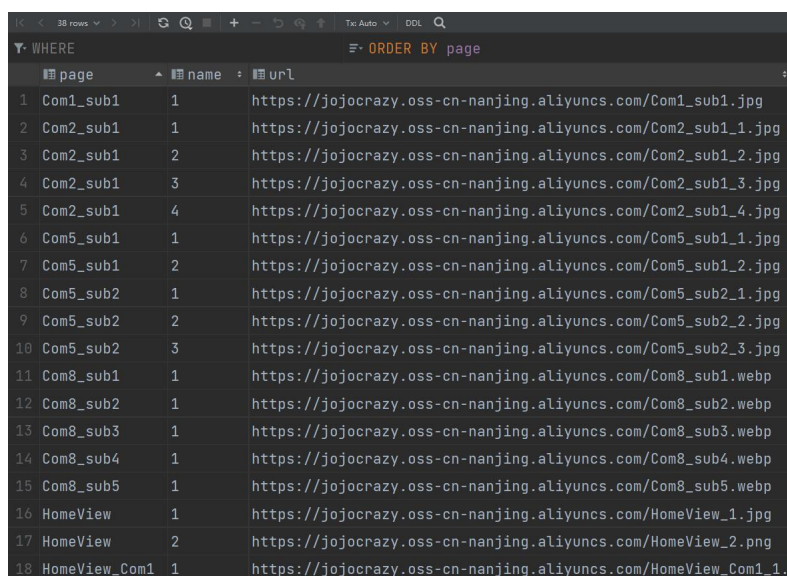
(1) Users 表:



	mailbox	password
1	1517056776@qq.com	123456789Qq
2	1517056776@qq.co	123456789Qq
3	1517056776@qq.co	123456789Qq
4	user1@example.com	123456789Aa
5	user2@example.com	123456789Aa
6	user@exa3mple.com	123456789Aa
7	user3@example.com	123456789Aa
8	user4@example.com	123456789Aa
9	user5@example.com	123456789Aa
10	user6@ample.com	123456789Aa
11	user7@example.com	123456789Aa

图 10 Users 表数据

(2) Urls 表:



	page	name	url
1	Com1_sub1	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com1_sub1.jpg
2	Com2_sub1	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com2_sub1_1.jpg
3	Com2_sub1	2	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com2_sub1_2.jpg
4	Com2_sub1	3	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com2_sub1_3.jpg
5	Com2_sub1	4	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com2_sub1_4.jpg
6	Com5_sub1	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com5_sub1_1.jpg
7	Com5_sub1	2	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com5_sub1_2.jpg
8	Com5_sub2	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com5_sub2_1.jpg
9	Com5_sub2	2	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com5_sub2_2.jpg
10	Com5_sub2	3	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com5_sub2_3.jpg
11	Com8_sub1	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com8_sub1.webp
12	Com8_sub2	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com8_sub2.webp
13	Com8_sub3	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com8_sub3.webp
14	Com8_sub4	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com8_sub4.webp
15	Com8_sub5	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/Com8_sub5.webp
16	HomeView	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/HomeView_1.jpg
17	HomeView	2	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/HomeView_2.png
18	HomeView_Com1	1	https://jojocrazy.oss-cn-nanjing.aliyuncs.com/HomeView_Com1_1.

图 11 Urls 表数据



## 2.5.部分关键的代码

### (1) 前端代码:

主要代码包括主界面 HomeView.vue、路由定义、App.vue、邮箱注册请求、用户密码校验请求、用户注册请求、图片 Url 请求，具体代码如下所示：

#### ➤ HomeView.vue 代码:

```
<template>
  <div class="home">
    <template>
      <comp1 :IsLogIn="IsLogIn"></comp1>
    </template>
    <div style="position: absolute; top: 63%; width: 100%;">
      <div style="z-index: 100">
        <!--筛选条-->
        <comp2></comp2>
      </div>
      <div style="height: 38px;background-color: transparent"></div>
      <div style="z-index: 10">
        <div class="eig" style="height: 8px; background-color: transparent">
        </div>
        <div>
          <!--根据住宿类型浏览-->
          <comp4></comp4>
        </div>
        <!--爆款目的地-->
        <comp5></comp5>
        <div>
          <!--入住我们的热门特色住宿-->
          <comp6></comp6>
        </div>
        <div>
          <!--快速规划-->
          <comp7></comp7>
        </div>
        <div class="eig">
          <!--精彩下一站，灵感这里找-->
```

```

        <comp8></comp8>
    </div>
    <div class="eig" v-if="!IsLogin">
        <!--获取即时优惠-->
        <comp9> </comp9>
    </div>
    <div>
        <!--探索中国-->
        <comp10></comp10>
    </div>
</div>

<div style="display: block; text-align: left;" >
    <div style="font-size: 30px; font-weight: bold; width: 79%; margin: auto">
客人喜爱的民宿短租</div>
        <el-link style="margin-left: 80%;font-size: 23px" type="primary"
href="https://www.baidu.com">
            探索民宿短租</el-link>
    </div>
    <!--客人喜爱的民宿短租-->
    <comp11></comp11>
    <!--热门目的地-->
    <comp12></comp12>
    
    <!--页尾以及 CopyRight-->
    <comp13></comp13>
</div>
</div>
</template>

<script>
import comp1 from '../components/HomeView_Com1.vue'
import comp2 from '../components/HomeView_Com2.vue'
import comp4 from '../components/HomeView_Com4.vue'
import comp5 from '../components/HomeView_Com5.vue'

```

```
import comp6 from '../components/HomeView_Com6.vue'
import comp7 from '../components/HomeView_Com7.vue'
import comp8 from '../components/HomeView_Com8.vue'
import comp9 from '../components/HomeView_Com9.vue'
import comp10 from '../components/HomeView_Com10.vue'
import comp11 from '../components/HomeView_Com11.vue'
import comp12 from '../components/HomeView_Com12.vue'
import comp13 from '../components/HomeView_Com13.vue'
export default {
  data() {
    return {
      urls:[],
      name:"HomeView",
      IsLogIn:false,
    };
  },
  beforeRouteEnter(to, from, next) {
    // 在进入路由前处理参数
    if (to.params.IsLogIn !== undefined) {
      next(vm => {
        vm.IsLogIn = to.params.IsLogIn;
      });
    } else {
      next();
    }
  },
  components: {
    comp1,
    comp2,
    comp4,
    comp5,
    comp6,
    comp7,
    comp8,
    comp9,
```

```

    comp10,
    comp11,
    comp12,
    comp13,
  },
}
</script>

```

```

<style>
.home{
}
.eig{
  width: 80%;
  left: 10%;
  position: relative;
}
</style>

```

### ➤ 路由定义

```

import Vue from 'vue'
import VueRouter from 'vue-router'

```

```

Vue.use(VueRouter)

```

```

const routes = [
  {
    path: '/home',
    name: 'home',
    component: () => import('../views/element/HomeView.vue'),
  },
  {
    path: '/log',
    name: 'log',
    component: () => import('../views/element/ElementView.vue')
  },
  {

```

```

    path: '/log2',
    name: 'log2',
    component: () => import('../views/element/ElementView2.vue'),
    props: (route) => ({
      log1data: route.params.log1data
    })
  },
  {
    path: '/log3',
    name: 'log3',
    component: () => import('../views/element/ElementView3.vue'),
    props: (route) => ({
      log1data: route.params.log1data
    })
  },
  {
    path: '/log4/:RegisteredMailbox',// 添加参数
    name: 'log4',
    component: () => import('../views/element/ElementView4.vue'),
  },
  {
    path: '/log5/:RegisteredMailbox',
    name: 'log5',
    component: () => import('../views/element/ElementView5.vue')
  },
  {
    path: '/test',
    name: 'test',
    component: () => import('../components/Test_Connection.vue')
  },
  {
    path: '/',
    redirect: '/home',
    // redirect: '/test',
  },

```

]

```
const router = new VueRouter({
  routes
})
```

```
export default router
```

➤ App.vue

```
<template>
```

```
<div>
```

```
<router-view></router-view>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
export default {
```

```
  comments: {},
```

```
  data() {
```

```
    return {
```

```
      message: "hello vue!",
```

```
    }
```

```
  },
```

```
}
```

```
</script>
```

➤ 邮箱注册请求

```
submitForm(ruleForm) {
```

```
  this.$refs[ruleForm].validate((valid) => {
```

```
    //跳转至 log3
```

```
    if (valid) {
```

```
      axios.get('api/users', {
```

```
        params: {mailbox: this.ruleForm.mailbox}
```

```
      })
```

```
      .then(response => {
```

```
        if (response.data.data === 1) {
```

```
          // 用户存在，进行路由跳转
```

```
          this.$router.push({
```

```
            name: 'log3',
```

```

        params: { log1data: this.ruleForm },
    });
  } else {
    // 用户不存在的处理
    this.$router.push({
      name: 'log2',
      params: { log1data: this.ruleForm },
    });
  }
})
.catch(error => {
  console.error('Error:', error);
  alert('查询失败');
});
} else {
  // alert('error submit!!');
  return false;
}
});
},

```

➤ 用户密码校验请求

```

submitForm(formName) {
  this.$refs[formName].validate((valid) => {
    this.LogInfo.mailbox = this.RegisteredMailbox;
    this.LogInfo.password = this.ruleForm.pass;
    if (valid) {
      axios.get('api/password', {
        params: { mailbox: this.LogInfo.mailbox,
          password: this.LogInfo.password }
      })
      .then(response => {
        if (response.data.data === true) {
          // 用户存在，进行路由跳转
          this.$router.push({
            name: 'home',

```

```

        params: { IsLogIn: true }
    });
    alert("登录成功! ");
} else {
    // 用户不存在的处理
    alert("密码错误, 请重新输入");
}
})
.catch(error => {
    console.error('Error:', error);
    alert('查询失败');
});
} else {
    // alert('error submit!!');
    return false;
}
});
➤ 用户注册请求
submitForm(formName) {
this.$refs[formName].validate((valid) => {
    this.LogInfo.mailbox = this.mailbox;
    this.LogInfo.password = this.ruleForm.pass;
    if (valid) {
        // 使用 Axios 发送 POST 请求, 设置 Content-Type 为"application/json"
        axios.post('/api/addUser', this.LogInfo, {
            headers: {
                'Content-Type': 'application/json', // 设置 Content-Type 头 为
"application/json"
            },
        })
        .then(response => {
            // 处理响应
            console.log(response.data.code);
            console.log(response.data.msg);
            console.log(response.data.data);

```



```
// 用户存在，进行路由跳转
this.$router.push({
  name: 'home',
  params: { IsLogIn: true }
});
alert("注册成功！");
})
.catch(error => {
  // 处理错误
  console.error(error);
});
} else {
  alert('error submit!!');
  return false;
}
});
➤ 图片 Url 请求
mounted() {
  axios.get('api/urls',{
    params:{page:this.name}
  })
  .then(response => {
    if (response.data.code === 1) {
      // alert("获取成功！");
      // url 获取成功的处理
      this.urls = response.data.data;
    } else {
      // url 获取失败的处理
      console.log(response.data.msg);
      alert("获取失败！");
    }
  })
  .catch(error => {
    console.error('Error:', error);
    alert('查询失败');
```

```
});
},
```

## (2) 后端代码:

主要代码包括 Pojo 实体类、Mapper 层、Service 层、Controller 层以及阿里云 OSS 注入类。具体代码如下所示:

### ➤ Pojo 实体类:

#### ① 请求统一相应类 Result:

```
package com.example.myproject.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Result {
    private Integer code;//响应码, 1 代表成功; 0 代表失败
    private String msg; //响应信息 描述字符串
    private Object data; //返回的数据
    //增删改 成功响应
    public static Result success(){
        return new Result(1,"success",null);
    }
    //查询 成功响应
    public static Result success(Object data){
        return new Result(1,"success",data);
    }
    //失败响应
    public static Result error(String msg){
        return new Result(0,msg,null);
    }
}
```

#### ② 用户类 User:

```
package com.example.myproject.pojo;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private String mailbox;
    private String password;
}
```

➤ Mapper 层:

① UserMapper

```
package com.example.myproject.mapper;
```

```
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;
```

```
@Mapper
public interface UserMapper {
```

```
    @Select("SELECT COUNT(*) FROM myproject.users WHERE mailbox =
#{mailbox}")
    int GetUserAccount(@Param("mailbox") String mailbox);

    @Select("SELECT password from myproject.users WHERE mailbox =
#{mailbox}")
    String GetUserPassword(@Param("mailbox")String mailbox);

    @Insert("INSERT INTO users (mailbox, password) VALUES (#{mailbox},
#{password})")
    int InsertUser(@Param("mailbox") String mailbox, @Param("password")
String password);
}
```

## ② UrlMapper

```
package com.example.myproject.mapper;
```

```
import org.apache.ibatis.annotations.Mapper;
```

```
import org.apache.ibatis.annotations.Select;
```

```
import java.util.List;
```

```
@Mapper
```

```
public interface UrlMapper {
```

```
    @Select("select url from myproject.urls where page = #{page} order by  
name ASC;")
```

```
    public List<String>GetUrlsAscOrderByName(String page);
```

```
}
```

## ➤ Service 层:

### ① UserServiceimpl

```
package com.example.myproject.service.impl;
```

```
import com.example.myproject.mapper.UserMapper;
```

```
import com.example.myproject.service.UserService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
```

```
public class UserServiceimpl implements UserService {
```

```
    @Autowired
```

```
    private UserMapper userMapper;
```

```
    public int getUserAccount(String mailbox){
```

```
        return userMapper.GetUserAccount(mailbox);
```

```
    }
```

```
    public String getUserPassword(String mailbox){
```

```
        return userMapper.GetUserPassword(mailbox);
```

```
    }
```

```
    public int insertUser(String mailbox,String password){
```

```

        int flag = userMapper.GetUserAccount(mailbox);
        if (flag == 0) {
            return userMapper.InsertUser(mailbox, password);
        }
        else {
            return 0;
        }
    }
}

```

## ② UrlServiceimpl

```
package com.example.myproject.service.impl;
```

```

import com.example.myproject.mapper.UrlMapper;
import com.example.myproject.service.UrlService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

```

```
import java.util.List;
```

```
@Service
```

```

public class UrlServiceimpl implements UrlService {
    @Autowired
    private UrlMapper urlMapper;
    public List<String>GetUrlsAscOrderByName(String page){
        return urlMapper.GetUrlsAscOrderByName(page);
    }
}

```

## ➤ Controller 层:

### ① UrlControl:

```
package com.example.myproject.controller;
```

```

import com.example.myproject.pojo.Result;
import com.example.myproject.service.UrlService;
import com.example.myproject.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;

```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
public class UrlController {

    @Autowired
    private UrlService urlService;

    @GetMapping("/urls")
    public Result GetUrlsAscOrderByName(String page){
        List<String> urls = urlService.GetUrlsAscOrderByName(page);
        if(urls.isEmpty()) {
            return Result.error("获取 url 失败！");
        }
        else {
            return Result.success(urls);
        }
    }
}
```

② UserControl:

```
package com.example.myproject.controller;

import com.example.myproject.pojo.Result;
import com.example.myproject.pojo.User;
import com.example.myproject.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin
public class UserController {

    @Autowired
```

```

private UserService userService;

@GetMapping("/users")
public Result GetUserAccount(String mailbox){
    System.out.println("查询指定邮箱的账户");
    int account = userService.getUserAccount(mailbox);
    if(account == 1)
        return Result.success(1);
    else{
        return Result.error("用户不存在");
    }
}

@GetMapping("/password")
public Result GetUserPassword(String mailbox,String password){
    System.out.println("查询指定邮箱对应密码");
    String TruePassword = userService.getUserPassword(mailbox);
    if (TruePassword.equals(password)){
        return Result.success(true);
    } else {
        return Result.success(false);
    }
}

@PostMapping("/addUser")
//JSON 接收参数
public Result InsertUser(@RequestBody User user){
    System.out.println("插入指定用户");
    String mailbox = user.getMailbox();
    String password = user.getPassword();
    int flag = userService.insertUser(mailbox, password);
    if (flag == 1){
        return Result.success(1);
    }
    else {

```

```

        return Result.error("用户已存在。");
    }
}
}

```

➤ 阿里云 OSS 注入类：

```

package com.example.myproject;
import com.aliyun.oss.ClientException;
import com.aliyun.oss.OSS;
import com.aliyun.oss.common.auth.*;
import com.aliyun.oss.OSSClientBuilder;
import com.aliyun.oss.OSSException;
import com.aliyun.oss.model.PutObjectRequest;
import com.aliyun.oss.model.PutObjectResult;
import java.io.File;

public class Demo {

    public static void main(String[] args) throws Exception {
        // Endpoint 以华东 1（杭州）为例，其它 Region 请按实际情况填写。
        String endpoint = "https://oss-cn-nanjing.aliyuncs.com";
        String accessKeyId = "LTAI5tLGziFBPP7AiVxBrd";
        String accessKeySecret = "M28eYsC380uh7ynJNPmwgr5ysnO";
        // 从环境变量中获取访问凭证。运行本代码示例之前，请确保已设置
        // 环境变量 OSS_ACCESS_KEY_ID 和 OSS_ACCESS_KEY_SECRET。
        EnvironmentVariableCredentialsProvider credentialsProvider =
        CredentialsProviderFactory.newEnvironmentVariableCredentialsProvider();
        // 填写 Bucket 名称，例如 examplebucket。
        String bucketName = "jojocrazy";
        // 填写 Object 完整路径，完整路径中不能包含 Bucket 名称，例如
        // exampledir/exampleobject.txt。
        String objectName = "HomeView_Com7_1.png";
        // 填写本地文件的完整路径，例如 D:\\localpath\\examplefile.txt。
        // 如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上
        // 传文件。
        String filePath=

```



```
"C:\\Users\\15170\\Desktop\\vue\\vue-project\\src\\assets\\quickcity.png";

// 创建 OSSClient 实例。
OSS ossClient = new OSSClientBuilder().build(endpoint,
accessKeyId,accessKeySecret);

try {
    // 创建 PutObjectRequest 对象。
    PutObjectRequest putObjectRequest = new
PutObjectRequest(bucketName, objectName, new File(filePath));
    // 如果需要上传时设置存储类型和访问权限, 请参考以下示例代码。
    // ObjectMetadata metadata = new ObjectMetadata();
    // metadata.setHeader(OSSHeaders.OSS_STORAGE_CLASS,
StorageClass.Standard.toString());
    // metadata.setObjectAcl(CannedAccessControlList.Private);
    // putObjectRequest.setMetadata(metadata);

    // 上传文件。
    PutObjectResult result = ossClient.putObject(putObjectRequest);
} catch (OSSException oe) {
    System.out.println("Caught an OSSException, which means your
request made it to OSS, "
        + "but was rejected with an error response for some reason.");
    System.out.println("Error Message:" + oe.getErrorMessage());
    System.out.println("Error Code:" + oe.getErrorCode());
    System.out.println("Request ID:" + oe.getRequestId());
    System.out.println("Host ID:" + oe.getHostId());
} catch (ClientException ce) {
    System.out.println("Caught an ClientException, which means the client
encountered "
        + "a serious internal problem while trying to communicate
with OSS, "
        + "such as not being able to access the network.");
    System.out.println("Error Message:" + ce.getMessage());
} finally {
```

```
        if (ossClient != null) {
            ossClient.shutdown();
        }
    }
}
```

三、系统测试

(1) 主界面：

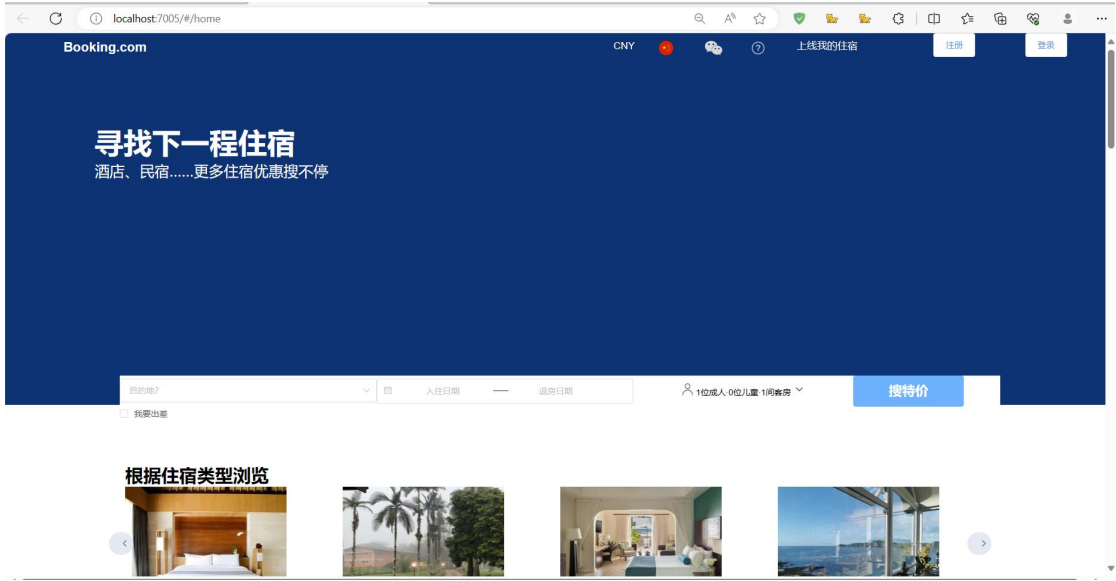


图 12 主界面部分一

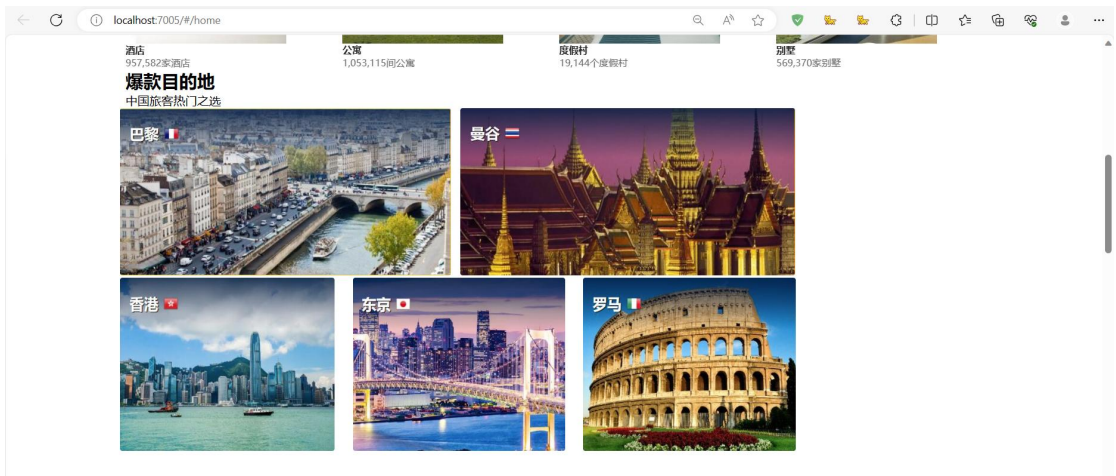


图 13 主界面部分二

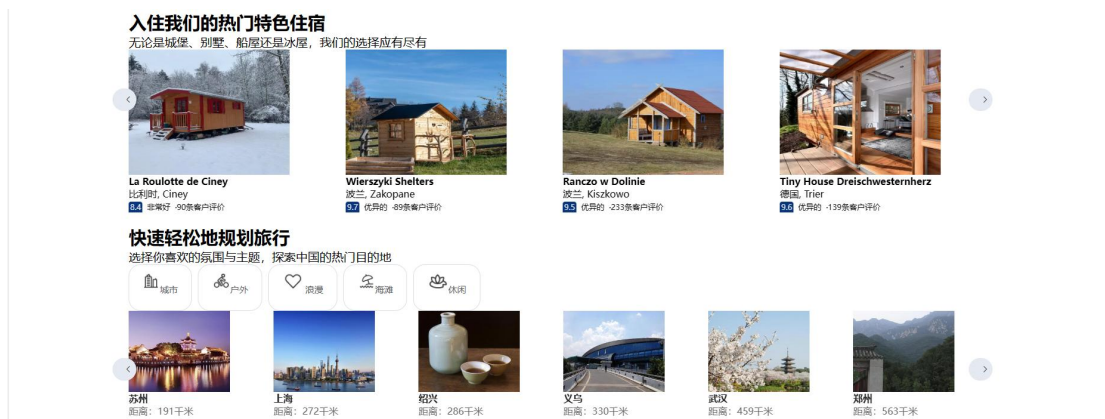


图 14 主界面部分三



图 15 主界面部分四

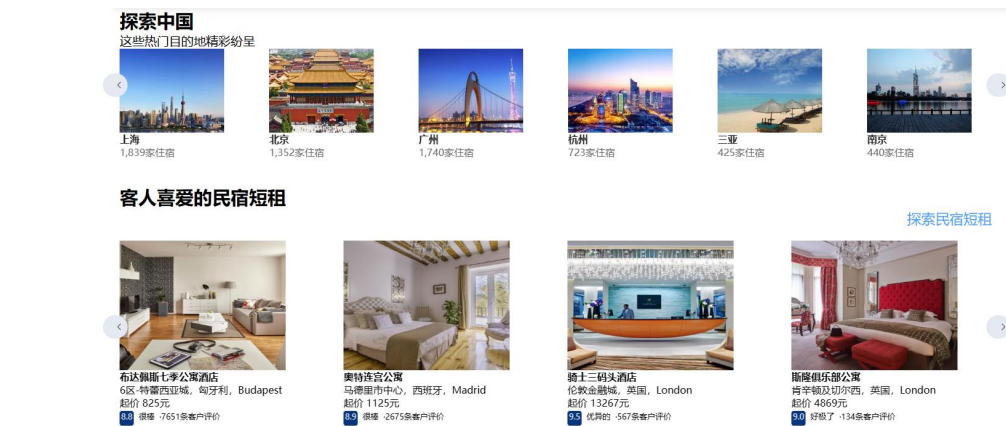


图 16 主界面部分五

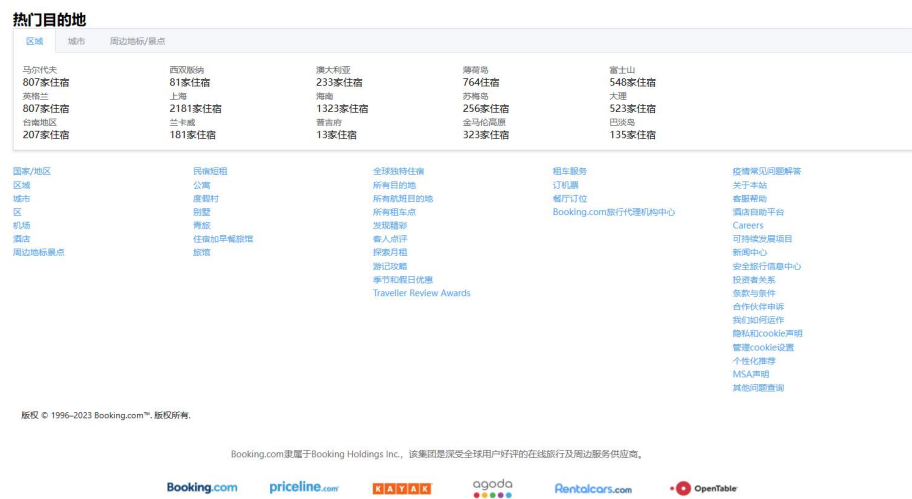


图 17 主界面部分六

(2) 注册 or 登录界面:



图 18 注册 or 登录界面

(3) 首次注册输入密码:

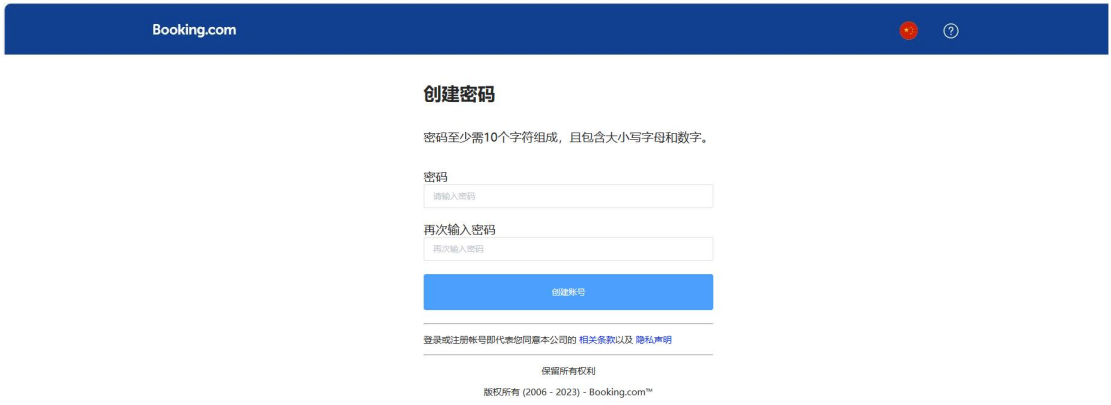


图 19 首次注册输入密码界面

(4) 登录密码校验:

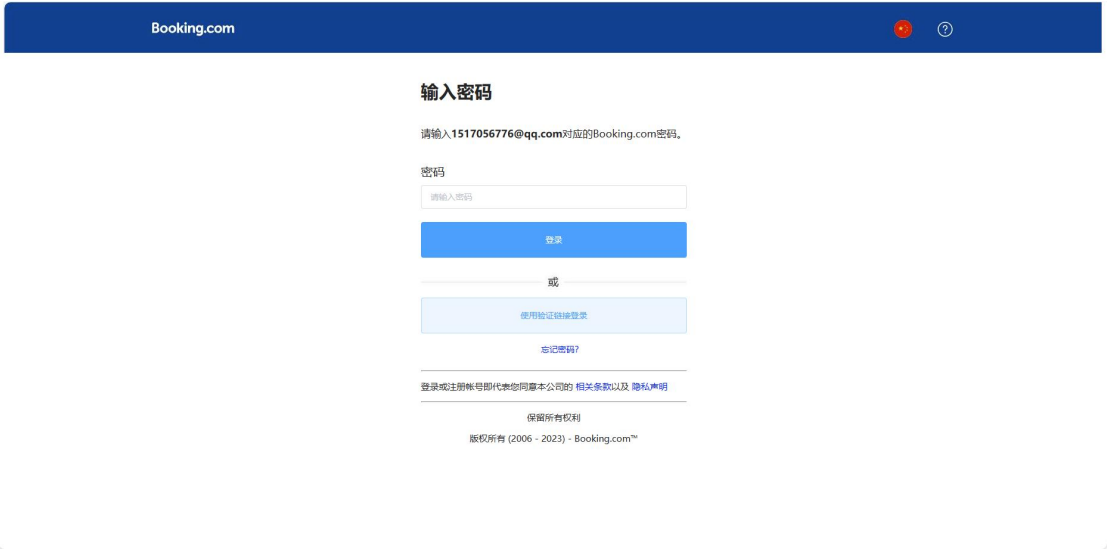


图 20 登录密码校验界面

(5) 忘记密码，发送邮件校验:



图 21 忘记密码界面



图 22 发送邮件校验界面

(6) 登录成功，返回主界面，右上角显示登录图标：

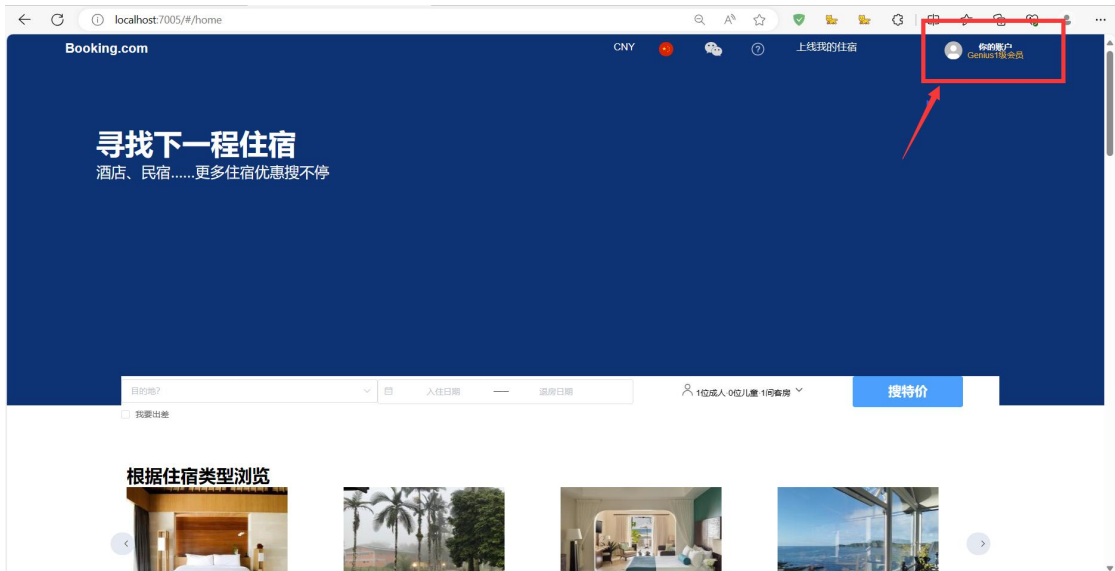


图 23 登录成功，返回主界面

(7) 后端数据库响应请求：

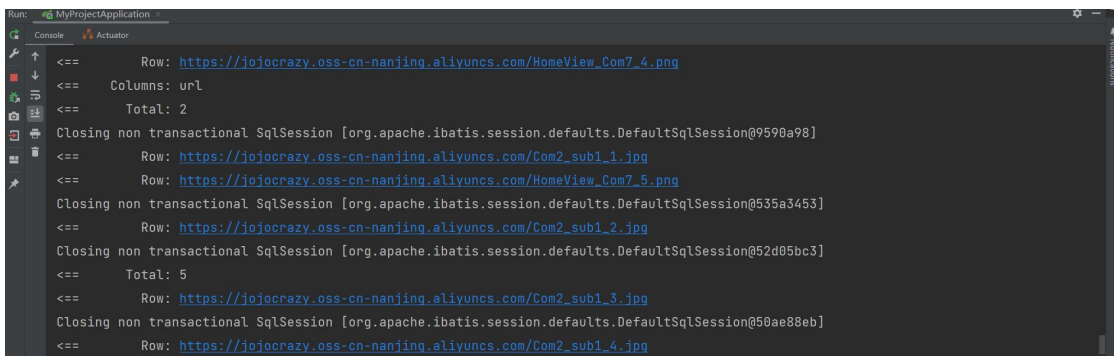


图 24 后端数据库响应请求界面

#### (8) 结果分析:

测试结果良好, 前后端联动工作正常, 所有功能都能够成功实现。

- 前端请求和后端响应: 前端能够成功发起请求, 而后端能够正确接收并发起相应。这表明前后端接口设计和通信机制是有效的。
- 用户注册功能: 用户注册功能正常工作, 用户的邮箱和密码能够被成功保存到数据库中。
- 图片展示功能: 图片展示功能也正常工作, 前端能够通过请求界面名, 后端能够正确从数据库中读取对应界面下的图片 URL 并返回给前端。这显示了前后端对图片数据的传递和处理是有效的。

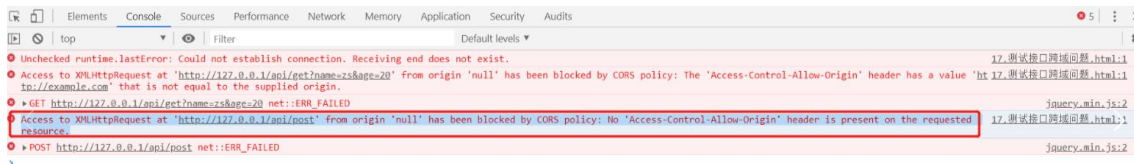
## 四、评价与结论

在本次实验中, 我首次接触到“前后端分离架构”的 Web 应用, 这成为我学习的一个重要契机。通过探索和实践, 我深入了解了一系列先进的技术框架, 包括 Vue.js、Spring Boot、MyBatis 等。

同时, 我也遇到了很多问题。其中关键问题及其解决方案如下:

- 报错信息:

在前端向后端发起请求时, 无法正确获取后端数据, 浏览器控制台发生类似下述报错:



- 错误原因:

这个错误是由于浏览器的同源策略 (Same-Origin Policy) 引起的, 浏览器不允许从一个源 (Origin) 的网页直接请求另一个源的资源, 除非目标服务器允许跨域请求。解决这个问题的常见方法是启用 CORS (跨域资源共享) 在服务器端。我的前端代码运行在 `http://localhost:7005`, 而后端接口在 `http://localhost:8080`。因此, 需要在后端服务器上配置 CORS 以允许跨域请求。

- 解决方案:

➤ 在 `vue.config.js` 中进行如下配置:

```
const { defineConfig } = require('@vue/cli-service');
module.exports = defineConfig({
  assetsDir: 'static',
  devServer: {
    port: 7005,
```

```
proxy: {
  '/api': {
    target: 'http://localhost:8080', // 后端服务器地址
    changeOrigin: true,
    pathRewrite: {
      '^/api': ''
    }
  }
}
});
```

- 在前端代码中使用相对路径来发起请求，而不需要指定完整的 URL。这样，前端请求会通过代理路径/api 到达后端服务器，无需指定完整的 URL。这个配置将使前端能够通过代理解决 CORS 问题，访问后端服务器的数据。修改 submitForm 函数如下：

```
submitForm() {
  // 使用相对路径，不需要指定完整的 URL
  axios.get('/api/depts') // 注意这里使用了代理前缀 /api
    .then(response => {
      if (response.data.code === 1) {
        // 如果响应状态码为 1，表示成功
        this.departments = response.data.data; // 存储实际数据
      } else {
        // 如果响应状态码为 0，表示失败
        console.error('Error fetching Data:', response.data.msg);
      }
    })
    .catch(error => {
      // 处理请求错误
      console.error('Error fetching departments:', error);
    });
},
```

总的来说，这次实验不仅让我了解了前后端分离的架构思想，还让我深入了解了整个 Web 应用的开发生命周期。从前端到后端，再到数据库的设计与管理，我更加全面地掌握了一个完整项目的构建与维护。这是一次收获颇丰的实验。



评分内容	评分细则		优秀	良好	中等	差
平时成绩  (20%)	学习态度：遵守实验室规定；（10分）					
	考勤情况（10分）					
设计开发成果和验收答辩  (60%)	程序演示与项目介绍：课题功能丰富，程序运行结果正确；界面友好，设计合理，创新性强。（10分）					
	程序开发与实现能力；工作量饱满；代码质量和运行性能良好，算法正确；（30分）					
	答辩时内容阐述清晰准确；问题回答正确（20分）					
设计报告质量 (20%)	报告文档书写格式规范、排版美观、文字流畅（10分）					
	内容正确详实、结构合理、反映系统设计流程（10分）					
评分等级						
指导教师 签名		日期	2023-11-12			
备注	评分等级有五种：优秀、良好、中等、及格、不及格					