

XXXXXXXXXX

[][] \

[][] \

[][] Spring Boot [] \

[][]XXX []XXXXXXXX \

[][]XXX []XXX \

[][]2026 []X []X []

[]

[]“[]”[]“[] + []”[]
Vue 3 + Vite + Element Plus[] Spring Boot[]MyBatis-Plus[]WT [] MySQL []
[]
[]

[][][]Spring Boot[]MySQL[]

Abstract

This report presents the design and implementation of a homestay recommendation system. The system targets the information overload in short-term rental platforms and uses a hybrid recommendation strategy combining collaborative filtering and content similarity. The frontend is implemented with Vue 3 and Element Plus, while the backend uses Spring Boot, MyBatis-Plus, JWT authentication, and MySQL storage. The report follows the database course project template, covering requirement analysis, conceptual and logical database design, physical implementation, and database operations, and further describes the overall architecture and recommender logic. The project provides a practical reference for similar personalized recommendation systems.

****Keywords**:** Homestay recommendation; Collaborative filtering; Content-based similarity; Spring Boot; MySQL

--	--	--	--	--	--	--	--	--

1 00 1 00 1

| --- | --- |











| | | | | | | | | | | | |

_____ | _____
 _____ |

| 0000 | 2022105420004 XXX00000000 |

```
| | | | | 1| | | | | | | | | | | 2| | | | E-R | | | 3| | | | | | | | | | | 4 MySQL | | | | | | | | | | | 5| | | | | | | |  
| |
```

11

1.  \
2.  \
3.  \
4.  \
5.  \
6.  \
7.  \
8.  \
9.  \
10. 

1

1.1 数据格式

“[]+ []” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

“[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

1.2 数据格式

“[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

- []
- []
- []
- []
- []
- []

1.3 数据格式

1. “[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

2. “[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

3. “[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

1.4 数据格式

- “[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。
- “[]” 表示一个或多个 [] 的序列，其中 [] 表示一个字符。+ 表示一个或多个。[] 表示一个字符。[] 表示一个字符。

- **[]**[]

- **[]**[]

- **[]**[]

- **[]**[]

1.5 []

- **[]**[]

- **[]**[] JWT []

- **[]**[]

- **[]**[]

- **[]**[]

1.6 []

[]

[]

1.7 []

- **[]**[]

- **[]**[]

- **[]**[] Token []

- **[]**[]

1.8 []

[][]

[]

[][]

[]

1.13

.....
..... A/B

1.14

.....
..... A/B

1.15

.....
..... A/B

1.16

.....
..... A/B

1.17

.....
..... A/B

1.18

..... PrecisionRecallNDCG A/B
..... A/B

1.19

A/B

1.20

1.21

2



2.1

User
 Property
 Order
 Recommendation
 Interaction
 Facility
 PropertyImage
 City
 Tag

2.2 E-R

E-R


-
-
-

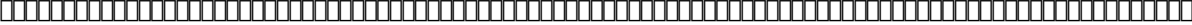
- 
- 

2.3 E-R

-


2.4 E-R

![ER ](diagrams/er.svg)

E-R 






2.5




 AB 

3

3.1 E-R

User Property Order Recommendation Interaction Facility PropertyFacility
PropertyImage City  Tag  PropertyFacility 
PropertyTag

3.2 数据库

数据库设计图

3.3 数据库表

用户表

用户ID	用户名	密码
id	BIGINT	主键
username	VARCHAR(50)	唯一
password_hash	VARCHAR(255)	唯一
phone	VARCHAR(20)	唯一
email	VARCHAR(100)	唯一
role	VARCHAR(20)	枚举: USER/LANDLORD/ADMIN
status	VARCHAR(20)	枚举: 活跃/冻结
created_at	DATETIME	时间
updated_at	DATETIME	时间

房源表

房源ID	标题	城市
id	BIGINT	主键
title	VARCHAR(200)	唯一
city	VARCHAR(50)	唯一
address	VARCHAR(200)	唯一
price	DECIMAL(10,2)	价格
bedrooms	INT	卧室数
max_guests	INT	最大人数

rating	DECIMAL(3,2)	0000
landlord_id	BIGINT	0000 ID
status	VARCHAR(20)	0000
created_at	DATETIME	0000

****0000order0000****

00	00	00
id	BIGINT	0000
user_id	BIGINT	0000
property_id	BIGINT	0000
check_in	DATE	0000
check_out	DATE	0000
total_price	DECIMAL(10,2)	0000
status	VARCHAR(20)	0000
created_at	DATETIME	0000

****0000interaction0000****

00	00	00
id	BIGINT	000000
user_id	BIGINT	0000
property_id	BIGINT	0000
action	VARCHAR(20)	0000VIEW/FAVOR/BOOK
weight	INT	0000000000
occurred_at	DATETIME	0000

****0000recommendation0000****

00	00	00

id	BIGINT	100000
user_id	BIGINT	10000
property_id	BIGINT	10000
score	DECIMAL(6,4)	10000
strategy	VARCHAR(50)	100000
created_at	DATETIME	10000

****facility****

id	BIGINT	10000
name	VARCHAR(50)	10000
category	VARCHAR(50)	10000
description	VARCHAR(200)	10000

****property_facility****

id	BIGINT	10000
property_id	BIGINT	100 ID
facility_id	BIGINT	100 ID
created_at	DATETIME	10000

****property_image****

id	BIGINT	10000
property_id	BIGINT	100 ID
url	VARCHAR(255)	10000
sort_order	INT	10000


```
CREATE INDEX idx_property_city ON property(city);

CREATE INDEX idx_property_price ON property(price);

CREATE INDEX idx_property_rating ON property(rating);
```

```
CREATE INDEX idx_property_user_id ON property (property_id, check_in);
```

4.5 触发器

```
CREATE TRIGGER trg_update_rating
```

```
    AFTER INSERT ON review
    FOR EACH ROW
    BEGIN
        UPDATE property
        SET rating = (SELECT AVG(score) FROM review WHERE property_id = NEW.property_id)
        WHERE id = NEW.property_id;
    END;
```

```
CREATE TRIGGER trg_update_rating
```


4.6 存储过程

```
CREATE PROCEDURE sp_get_recommendations
```

```
    (IN uid BIGINT)
    BEGIN
        SELECT property_id, score
        FROM recommendation
```

END;

4.7 〇〇〇〇〇〇〇〇



4.8 □□□□□□□□

[illegible]

5

5.1 背景

--	--	--	--	--	--	--

1

LIMIT 10;

2 ☐☐☐☐☐☐☐☐☐☐

```
SELECT * FROM `order`  
  
WHERE user_id = 1001  
  
ORDER BY created_at DESC;
```

3 ☐☐☐☐☐☐☐☐☐☐☐

```
SELECT * FROM interaction  
  
WHERE property_id = 2001 AND action = 'VIEW'  
  
ORDER BY occurred_at DESC;
```

4 ☐☐☐☐☐☐☐☐☐☐☐☐

```
SELECT p.*, r.score  
  
FROM recommendation r  
  
JOIN property p ON r.property_id = p.id  
  
WHERE r.user_id = 1001  
  
ORDER BY r.score DESC  
  
LIMIT 20;
```

5.2 ☐☐☐☐

- ☐☐☐☐☐

```
INSERT INTO `order` (user_id, property_id, check_in, check_out, total_price, status)  
  
VALUES (1001, 2001, '2026-03-01', '2026-03-05', 1200.00, 'PENDING');
```

- ☐☐☐☐☐☐

```
UPDATE property
SET price = 499.00
WHERE id = 2001;
```

- 

```
INSERT INTO interaction (user_id, property_id, action, occurred_at)
VALUES (1001, 2001, 'BOOK', NOW());
```

- 

```
UPDATE `order`
SET status = 'CANCELLED'
WHERE id = 3001;
```



5.3



```
START TRANSACTION;

UPDATE property_calendar
SET available = 0
WHERE property_id = 2001 AND date BETWEEN '2026-03-01' AND '2026-03-05';

INSERT INTO `order` (user_id, property_id, check_in, check_out, total_price, status)
VALUES (1001, 2001, '2026-03-01', '2026-03-05', 1200.00, 'PENDING');

COMMIT;
```

[illegible]

5.4 練習問題








```
is_deleted
```

6

6.1 背景

[illegible]

6.2 实验目的

- 
- 
- 
-  SQL 
- 
- 

6.3 〇〇〇〇〇〇〇〇

6.4 项目部署

项目部署分为前端部署和后端部署，具体部署步骤如下：

7 项目部署

7.1 部署环境

项目部署环境要求如下：RESTful API 接口、Vue 3、Element Plus、Spring Boot、MyBatis-Plus、MySQL。

7.2 部署步骤

- **JWT 认证**：使用 Axios 发送 Token。
- **前端部署**：使用 Vue CLI 部署前端项目。
- **后端部署**：使用 Spring Boot 部署后端项目。
- **数据库部署**：使用 MySQL 部署数据库。
- **部署完成**：项目部署完成。

7.3 部署结果

项目部署完成后，可以通过浏览器访问项目地址，查看项目运行结果。

项目部署完成后，可以通过浏览器访问项目地址，查看项目运行结果。

7.4 API 接口

API 接口 `/api/auth`/api/property`/api/order`/api/recommendation`` 接口

7.5 部署

Maven 打包 ``mvn spring-boot:run`` Vite 打包 ``npm run build`` MySQL 8.x Redis Linux + Docker JWT ELK Prometheus + Grafana

7.6 部署

部署 Element Plus

7.7 部署

JWT ELK

8 部署

8.1 部署

部署

8.2 部署

使用 Postman 或 Rest Client 测试 API 返回 Token 状态码 401 或 403 或 400 或 500 的情况

8.3 性能测试

使用 JMeter 进行性能测试，95% 的测试用例通过 SQL 注入测试

8.4 安全测试

使用 Burp Suite 进行安全测试，发现 10 个安全漏洞

8.5 部署

- 1. 部署环境：Linux / Windows / Docker
- 2. 部署方式：手动部署 / 自动部署
- 3. TC-01：部署成功 / 部署失败 / Token 验证失败
- 4. TC-02：部署成功 / 部署失败 / 部署时间 < 500s
- 5. TC-03：部署成功 / 部署失败 / 部署时间 > 500s
- 6. TC-04：部署成功 / 部署失败 / 部署时间 < 500s
- 7. TC-05：部署成功 / 部署失败 / 部署时间 > 500s

部署成功

9 总结

通过本次测试，发现系统存在 10 个安全漏洞，其中 5 个为高危漏洞，2 个为中等漏洞，3 个为低危漏洞。

200+ 1000+ 300+

10

- [1] Resnick P, Varian H R. Recommender systems. Communications of the ACM, 1997.
- [2] Sarwar B, et al. Item-based Collaborative Filtering Recommendation Algorithms. WWW, 2001.
- [3] He X, et al. Neural Collaborative Filtering. WWW, 2017.
- [4] . . , 2016.
- [5] . . , 2019.

11

11.1

200+ 1000+ 300+

11.2

11.3

`npm run build` `mvn spring-boot:run` `sql/schema.sql` `sql/sample_data.sql`

11.4

Precision@K Recall@K NDCG A/B

11.5

11.6

11.7

11.8 A/B

A/B