<eyulingo>

测试报告

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <日/月/年> | <x.x> | <详细信息> | <姓名> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 范围 4

1.3 定义、首字母缩写词和缩略语 4

1.4 参考资料 4

1.5 概述 4

2. 测试概要 4

3. 测试环境 4

4. 测试结果及分析 4

4.1 需求覆盖率及缺陷分布 4

4.2 缺陷严重程度 5

5. 缺陷清单 5

5.1 功能性缺陷 5

5.2 非功能性缺陷 6

6. 测试结论与建议 6

测试报告

# 简介

## 目的

该“测试计划”文档有助于实现以下目标:

（1) 为软件可靠性与安全性的评估提供依据

（2) 提供系统化、规范化、工程化、实用化的测试技术规范，尽早发现故障。

（3) 检查软件对误操作的处理能力

（4) 此计划适用于系统中各组成部分的软件测试工作。

（5) 验证软件是否满足软件设计所规定的技术要求

## 范围

测试工作包括：

（1) 功能测试

## 定义、首字母缩写词和缩略语

## 参考资料

《优邻购软件需求规约》

《优邻购测试用例》

## 概述

本测试文档以测试为核心，从测试概要开始，接着列出测试的环境，然后较详细的列出了测试的结果，之后罗列出测试中发现的缺陷，最后得出了本次测试的结论与建议。

# 测试概要

测试时间为 2019/7/22-2019/7/22，

测试地点为上海交通大学闵行校区，

主要测试人员有于喜千，何思泽，高鹏成，高鸿博。

测试中，涉及用户体验的均为手动访问系统测试。

# 测试环境

移动端：iOS 13.0

Web端：火狐浏览器

后台；Intellij IDEA（IDE）2017.3.4 64 位版本

# 测试结果及分析



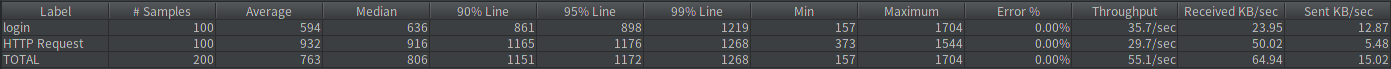
## 4.1.1需求覆盖率及缺陷分布

**表1 需求覆盖率及缺陷分布**

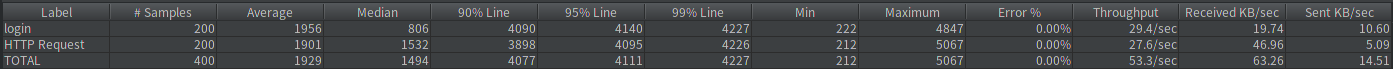
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **测试内容** | | **测试**  **用例数** | **需求**  **覆盖率** | **缺陷数** | **缺陷率** | **备注** |
| **功能项** | **获取验证码** | 4 | 100% | 0 | 0% |  |
| **用户注册** | 9 | 100% | 0 | 0% |  |
| **用户登录** | 3 | 100% | 0 | 0% |  |
| **修改密码** | 5 | 100% | 0 | 0% |  |
| **找回密码** | 6 | 100% | 0 | 0% |  |
| **管理员登录** | 3 | 100% | 0 | 0% |  |
| **管理员获取经销商信息** | 1 | 100% | 0 | 0% |  |
| **管理员修改经销商信息** | 1 | 100% | 0 | 0% |  |
| **管理员获取店铺信息** | 1 | 100% | 0 | 0% |  |
| **管理员修改店铺信息** | 1 | 100% | 0 | 0% |  |
| **经销商登录** | 3 | 100% | 0 | 0% |  |
| **经销商获取信息** | 1 | 100% | 0 | 0% |  |
| **经销商修改信息** | 1 | 100% | 0 | 0% |  |
| **用户修改头像** | 1 | 100% | 0 | 0% |  |
| **用户查看常用收货地址** | 1 | 100% | 0 | 0% |  |
| **用户编辑常用收货地址** | 2 | 100% | 0 | 0% |  |
| **用户增加、删除常用收货地址** | 2 | 100% | 0 | 0% |  |
| **用户搜索商品** | 2 | 100% | 0 | 0% |  |
| **用户查看商品详情** | 1 | 100% | 0 | 0% |  |
| **查看、撰写商品评论** | 2 | 100% | 0 | 0% |  |
| **购买商品** | 6 | 100% | 0 | 0% |  |
| **用户搜索店铺** | 3 | 100% | 0 | 0% |  |
| **查看店铺详情** | 2 | 100% | 0 | 0% |  |
| **店铺评论** | 3 | 100% | 0 | 0% |  |
| **订单项目** | 3 | 100% | 0 | 0% |  |
| **功能项小计** | 70 | 100% | 0 | 0% |  |
| **非功**  **能项** | **性能** |  |  |  |  |  |
| **可靠性** |  |  |  |  |  |
| **兼容性** | 3 | 100% | 0 | 0% |  |
|  | **非功能项小计** |  |  |  |  |  |
|  | **总计** |  |  |  |  |  |

## 4.1.2压力测试

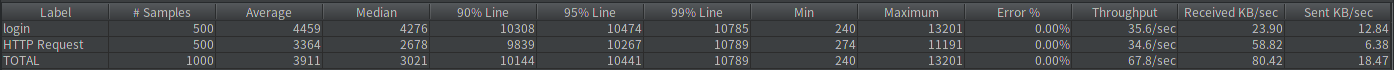
（在线人数1000，并发量100）



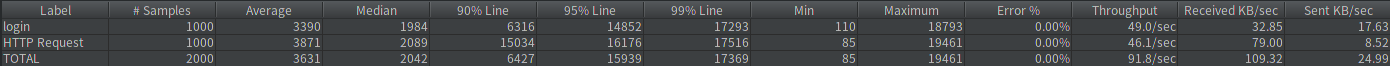
（在线人数1000，并发量200）



（在线人数1000，并发量500）



（在线人数1000，并发量1000）



## 4.2缺陷严重程度

**表2 缺陷严重程度**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **严重程度** | **致命** | **严重** | **一般** | **细微** | **总数** |
| **缺陷个数** | 0 | 0 | 1 | 1 | 0 |
| **占缺陷百分比** | 0% | 0% | 50% | 50% | 100% |

# 缺陷清单

## 功能性缺陷

**表3 功能性缺陷列表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序**  **号** | **缺陷**  **编号** | **严重程度** | **功能模块** | **缺陷标题** | **缺陷描述** | **测试用例编号** |
| 2 | #2 | 细微  minor | iOS 客户端 | 偶发的状态栏遮挡缺陷 | 在反复切换iOS客户端各窗口时，偶尔出现状态栏下移20px，遮挡正常界面元素的漏洞 | UI 测试 #2 |

表中相关项说明：

1. 缺陷编号：为每个缺陷赋予一个唯一的编号，可以通过此编号对缺陷进行跟踪。例如：Bug001。
2. 缺陷严重程度：缺陷可以根据严重程度分为以下几种情况。
3. 致命（fatal）：致命的错误，测试执行直接导致系统死机、蓝屏、挂起、或是程序非法退出；系统的主要功能或需求没有实现。
4. 严重（critical）：严重错误，系统的次要功能点或需求点没有实现；数据丢失或损坏。执行软件主要功能的测试用例导致系统出错，程序无法正常继续执行；程序执行过于缓慢或是占用过大的系统资源。
5. 一般（major）：不太严重的错误，这样的缺陷虽然不影响系统的基本使用，但没有很好地实现功能，没有达到预期的效果。如次要功能丧失，界面错误，打印内容、格式错误，提示信息不太正确，或用户界面太差，简单的输入限制未放在前台进行控制，删除操作未给出提示，操作时间长等。
6. 细微（minor）：一些小问题，对功能几乎没有影响，产品及属性仍可使用。如软件的实际执行过程与需求有较小的差异；程序的提示信息描述容易使用户产生混淆；有个别错别字、 文字排列不整齐；辅助说明描述不清楚，显示格式不规范，长时间操作未给用户进度提示，提示窗口文字未采用行业术语，可输入区域和只读区域没有明显的区分标志，系统处理未优化等。
7. 功能模块：所测试并出现该缺陷的功能模块名称。
8. 缺陷标题：描述缺陷的标题。
9. 缺陷描述：对缺陷的详细描述。描述缺陷的重现步骤和问题所在，描述需清晰明了，使研发人员可以按照描述将缺陷重现出来。必要时可以放上相应的截图，以便于研发人员理解问题所在。

## 非功能性缺陷

**表4 非功能性缺陷列表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序**  **号** | **缺陷**  **编号** | **严重程度** | **测试类型** | **缺陷标题** | **缺陷描述** | **测试用例编号** |
| 1 | #1 | 一般  major | 性能 | 高并发量请求时的拒绝服务缺陷 | 在并发量过大（大约超过500请求／秒）时，偶尔出现请求失败的漏洞 | 压力测试#1 |

# 测试结论与建议

我们组的测试工作事实上开展得比较晚，因为在此之前我们的功能实现还不完善，尤其是基础性的功能出现错误时很难支撑后续的测试。因此我们初步进行的「测试」主要是边进行代码编写边进行的人工测试，结合上CI／CD的黑盒测试，确保每次开发部署版本的「基本可用性」。

其中，UI测试主要是通过自动执行一些流程化的操作，帮助iOS客户端每次的部署中减少许多机械性的重复工作，提升测试效率。

后续在功能完成到一定程度后，我们停止开发，并进行了后续的客户端及服务器的单元测试和iOS 移动端UI界面测试。借此我们完成了错误返回流程的标准化，确保了部分不常用的错误流程分支都被执行到，提高了代码测试覆盖率。

最后，我们使用Apache JMeter和Python脚本构造了大量临时数据对服务器进行了分段的压力测试，确定了服务器的服务性能上限，并根据此调整Docker容器的部分设定，提高了响应时间，并实现了简单的攻击防御机制。