

软件质量保证

实验报告



成员 1

成员 2

姓名

班级

学号

电话

Email

日期

目录

一、题目要求	1
二、被测项目简介	1
三、十种测试技术的应用	3
1、 等价类测试	3
2、 基于规格说明的测试	8
3、 基于风险的测试	9
4、 压力测试	10
5、 回归测试	11
6、 探索性测试	12
7、 用户测试	18
8、 场景测试	21
9、 随机测试	25
10、 功能测试	26
四、JUnit 测试	27
1、 Admin 类	27
1、 Bed 类	30
1、 Building 类	32
1、 Park 类	36
1、 Room 类	39
1、 Student 类	42
1、 User 类	44
五、EclEmma 测试	49
六、缺陷报告	53
报告一	53
报告二	54
报告三	55
报告四	56
七、测试总结报告	57
八、附件说明	58

一、题目要求

任选一个被测软件（建议自己以往开发的软件项目）

1. 采用课堂所讲述的 10 种黑盒测试技术，对软件进行测试（可以选择部分功能（片段））。

（1）应用了某种技术，讲清楚如何应用的，比如说等价类测试。

（2）如果某种技术不适合被测软件，讲清楚为什么不适合（原因）。

2. 针对被测软件的某个/些类，应用 JUnit 进行测试，给出测试类（测试用例）；其他编程语言选择对应测试工具；

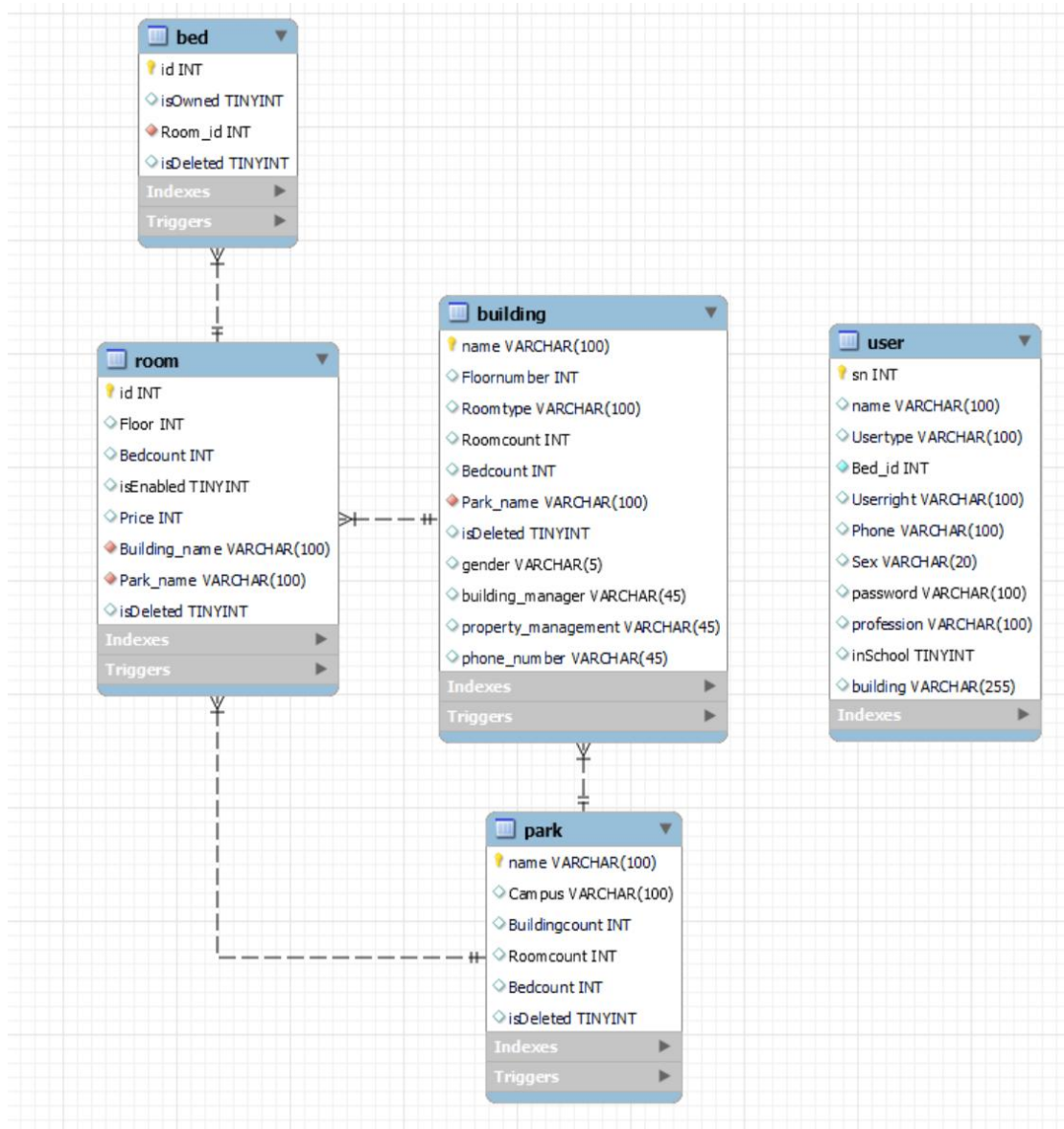
3. 针对被测软件某个/些类，应用 Eclemma 进行测试，给出覆盖分析；其他编程语言选择对应测试工具；

4. 撰写测试报告（1）被测软件简单描述，（2）测试总结报告，（3）每人给出 2-3 个 bug 报告。

二、被测项目简介

被测软件是一个在大二下小学期集体开发的一个房源管理系统。此系统中包含用户注册、用户登录、房源管理、床位管理、用户管理等功能。

房源管理系统的 ER 图如下：



房源管理系统的主要功能模块如下：

用户注册：

用户可以在登陆界面点击注册按钮，输入有关个人信息进行注册。

用户登录：

用户可以在登陆界面输入账号密码，点击登录按钮进行登录。

房源管理：

管理员可对房源（园区、楼栋、房间）进行增、删、改、查操作。

普通用户能看到自己所在房源（园区、楼栋、房间）的房源信息。

床位管理：

管理员可对床位进行增、删、改、查操作。

普通用户可以查看房源信息和床位信息。

用户管理：

管理员可查看、编辑用户信息和更改用户权限。

三、十种测试技术的应用

1、等价类测试

等价类测试是一种常见的黑盒测试技术。由于全面穷举所有可能的测试情况几乎是不现实的，因此需要选择一部分测试用例来对系统进行验证。然而，这种选择性可能导致某些潜在缺陷未被发现。为了在较少的测试用例下实现尽可能全面的测试，等价类测试提供了一种有效的解决方案。

其基本思想是将输入数据划分为若干个等价类，确保每个等价类中的数据在行为上具有相似性。通过选择每个等价类的代表性测试用例，可以有效地覆盖整个等价类。因为一旦代表性用例通过测试，就可以合理地推断该等价类中的其他输入也会得到类似的处理。这样的方法不仅能显著减少所需的测试用例数量，还能保证两个关键方面的实现：

①某种意义上的完备性测试：通过选取代表性用例，确保涵盖了每个等价类的典型情况，在有限的测试用例集中尽可能覆盖各种输入

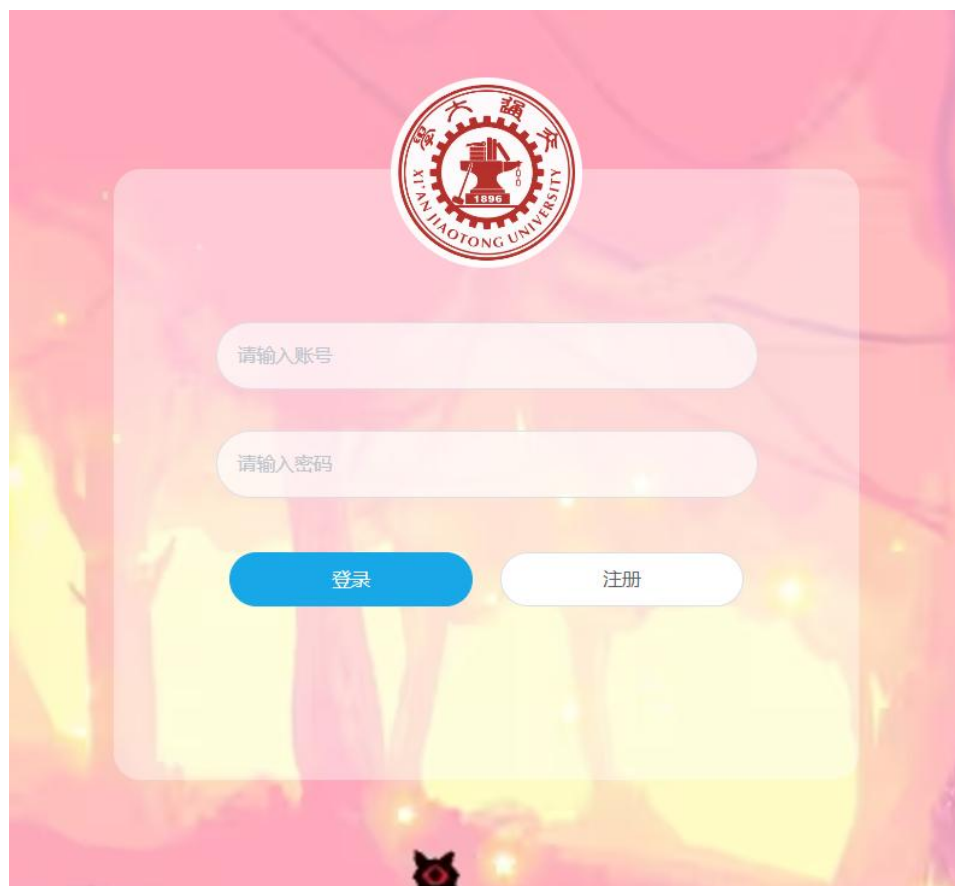
情况，从而提高了测试的全面性和覆盖率。

②避免冗余测试：通过选择具有代表性的测试用例，能够避免对同一等价类中其他测试用例的重复验证，因为对代表性用例的测试结果可推及到该类中的其他案例。

这一方法的优点在于，在确保测试效率的同时，能够通过较少的测试用例实现较为全面的测试覆盖。然而，等价类测试并不能完全覆盖所有可能的情况，它主要是在合理的范围内尽量涵盖各种输入场景。因此，在设计测试用例时，还需结合领域知识和实际应用场景，以保证测试的有效性和全面性。

我们选择对用户登录功能用等价类测试，按照如下步骤进行：

一、确定输入条件以及每个条件的有效等价类和无效等价类



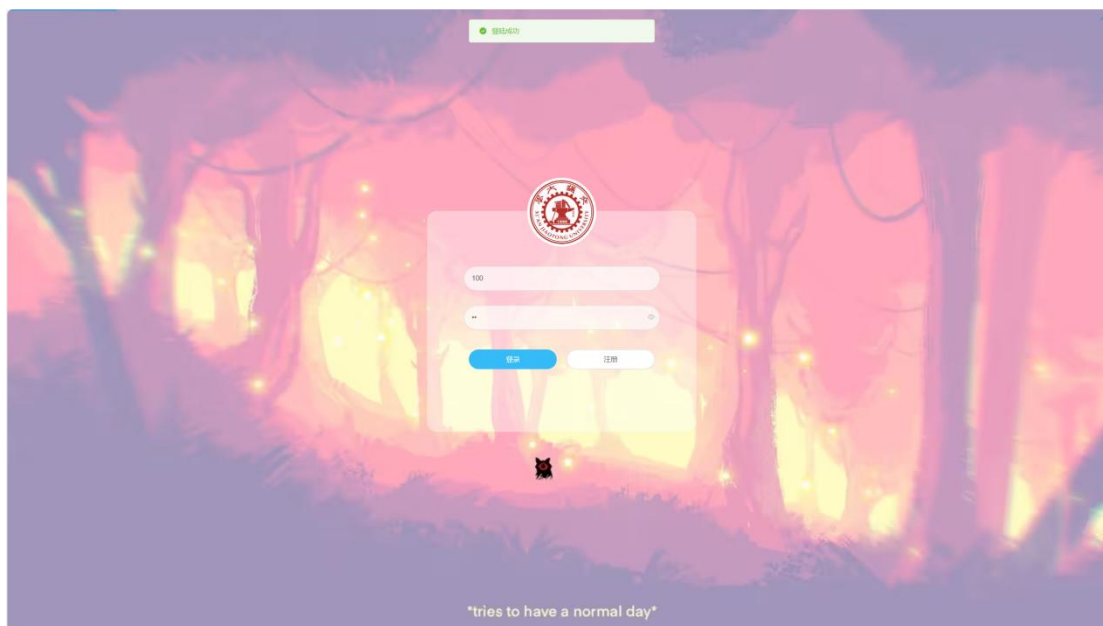
条件	有效等价类	无效等价类
账号格式	①账号格式为：只包含数字且不为空	②账号包含其他字符 ③账号为空
账号是否存在	④是	⑤否
密码是否匹配成功	⑥是	⑦否
密码格式	⑧不为空	⑨为空

二、设计测试用例，采用传统等价类技术设计测试用例

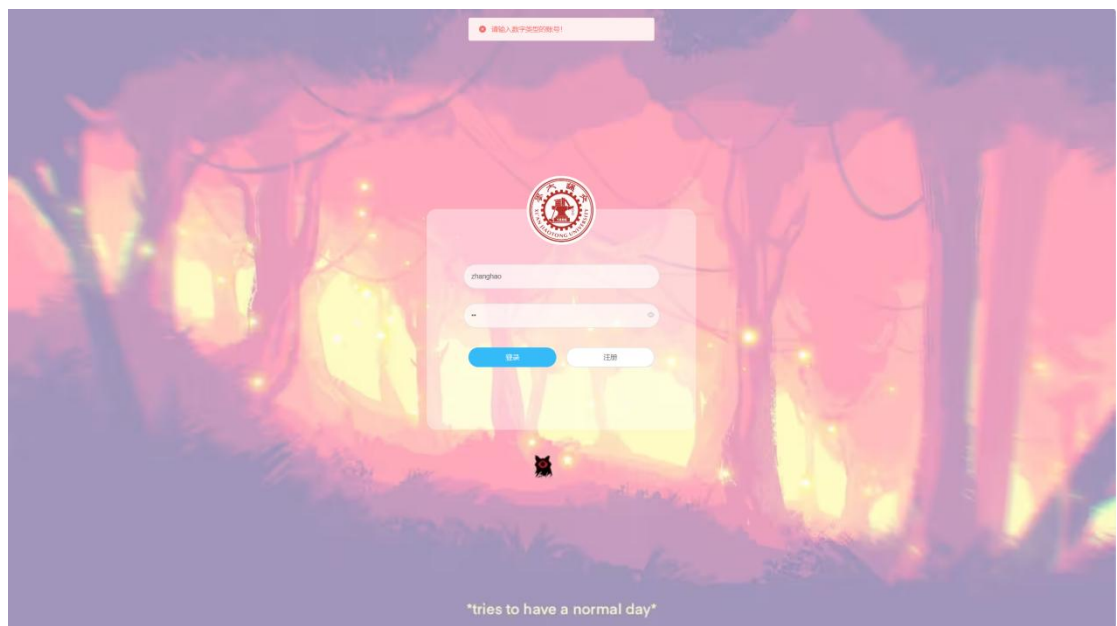
编号	测试输入	预期输出	覆盖等价类
1	账号：100 密码：w1	正常登陆系统	①④⑥⑧
2	账号：zhanghao 密码：w1	弹窗“请输入数字类型的账号”	②
3	账号：（空） 密码：w1	弹窗“账号不能为空”	③
4	账号：99999 密码：w1	弹窗“账号或密码错误”	⑤
5	账号：100 密码：11	弹窗“账号或密码错误”	⑦
6	账号：100 密码：（空）	弹窗“密码不能为空”	⑨

实际运行结果如下图所示：

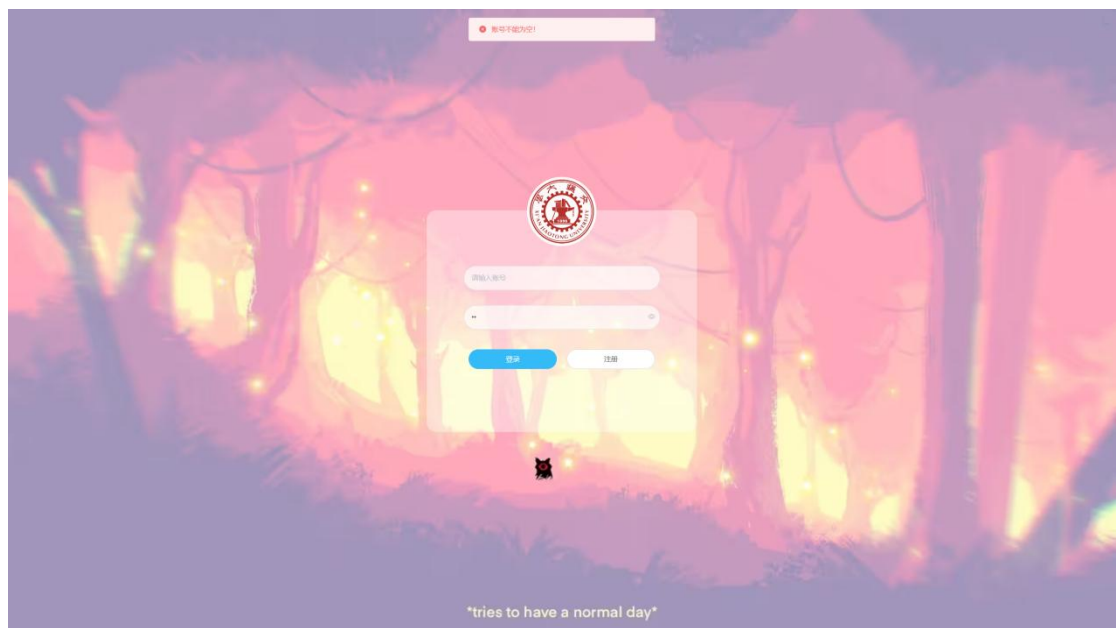
编号 1 运行结果：



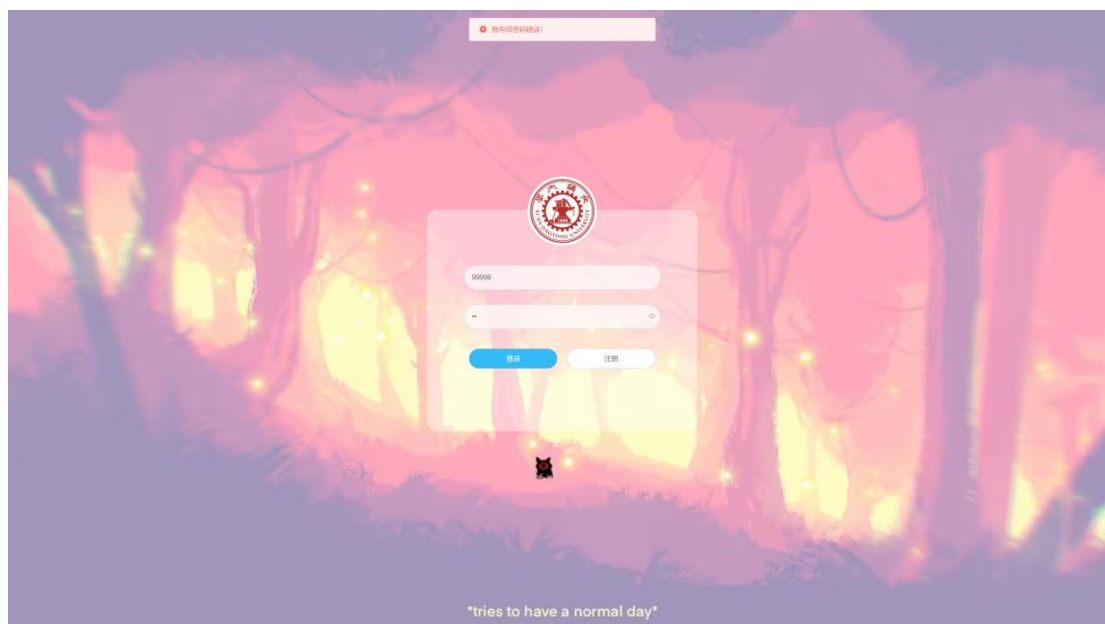
编号 2 运行结果：



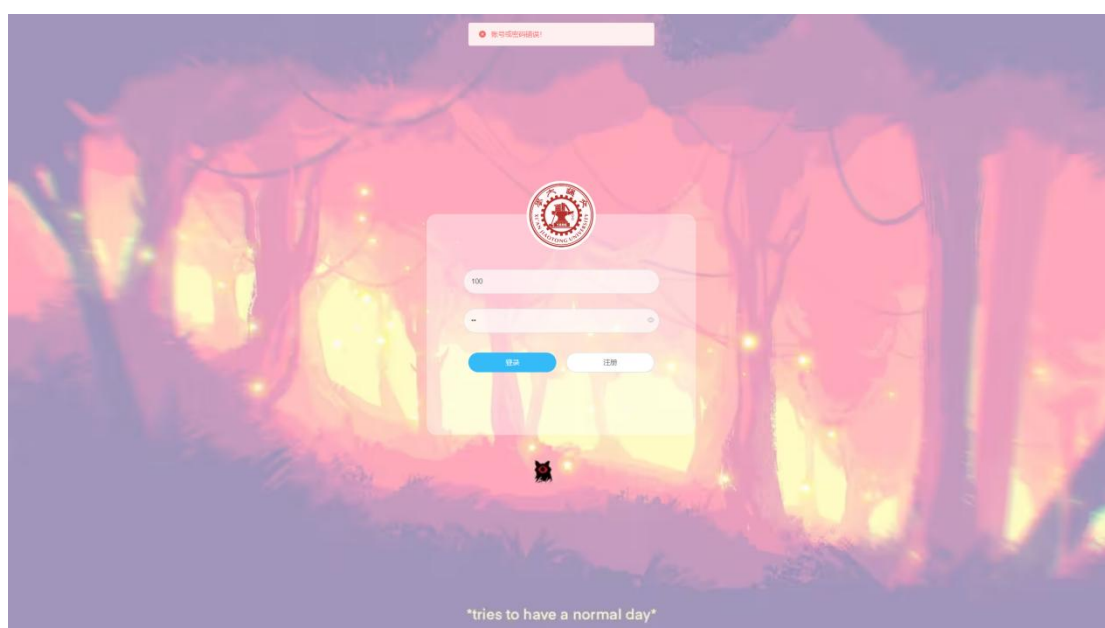
编号 3 运行结果:



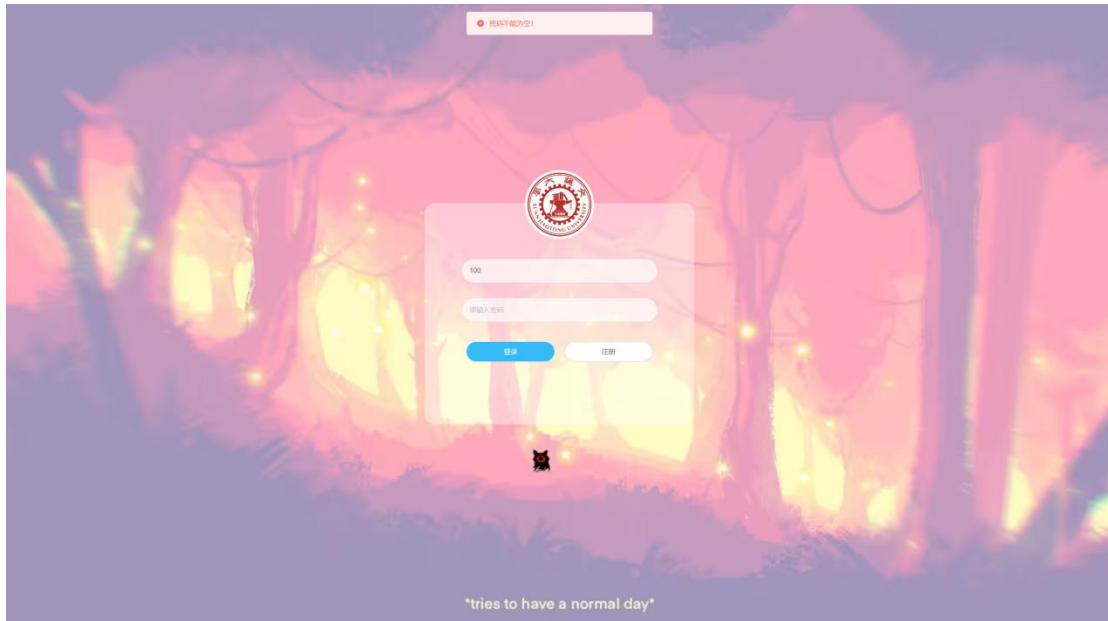
编号 4 运行结果:



编号 5 运行结果:



编号 6 运行结果:



2、基于规格说明的测试

基于规格说明书的测试通常是软件开发过程中的重要步骤。需求规格说明书在设计、编码和测试方面提供了一个框架，但软件的质量并不仅仅取决于是否符合规格说明书。然而，验证软件是否满足规格说明书中的需求仍然是必不可少的。

在某些情况下，尤其是对于功能相对简单的软件项目，可能并未编制详细的需求规格说明书或测试用例。这种情况下，可以参考其他可用资料，如国际标准、产品更新说明等，作为测试依据。即使软件复杂度较低，依然建议进行必要的测试，以确保其在实际应用中能够满足需求。

如果不采用基于需求规格说明书的测试方法，可能需要选择其他灵活的测试策略，例如探索性测试（**Exploratory Testing**）或临时性测试（**Ad-hoc Testing**）。这种测试方式通常更加依赖测试人员的经验和判断，而非严格遵循预先定义的规范。

总体而言，对于简单的软件项目，虽然可能没有完善的需求规格说明书，但测试工作依然是不可忽视的。是否采用基于规格说明书的测试方法应经过充分的权衡，确保所选的替代方案能够满足软件质量保障的需求并验证其功能和合规性。

由于选择的房源管理系统是一个非常简单的软件项目，且该系统是根据实训老师所给的前端模板直接进行前后端开发的，并没有编写需求规格说明书，同时也难以找到如国际标准、软件更新备忘录等替代方案，所以决定不使用基于需求规格说明书的测试技术。

3、基于风险的测试

基于风险的测试是一种以软件潜在风险为核心，制定测试计划和目标的有效方法。通过系统地分析可能存在的风险，该方法能够合理设计和组织测试活动。其核心思想是，全面穷尽所有测试情况在实际中往往难以实现，因为时间和资源的限制使得无法彻底排查所有缺陷。因此，基于风险的测试通过优先关注高风险领域，为这些领域制定针对性的测试策略，从而在有限的资源下最大化地发现关键问题。

基于风险的测试通常包括以下关键步骤：

- （1）识别潜在风险：分析软件可能面临的风险因素。
- （2）设计针对性测试：根据每个风险因素选择最适合的测试方法，并生成相应的测试用例。
- （3）评估测试覆盖率：检查测试是否覆盖了所有高风险领域，并找出测试中可能存在的薄弱环节。

（4）参考历史数据：列出过去的缺陷记录、配置问题以及用户的主要反馈。

（5）评估测试结果：分析测试的实际效果，确定需要进一步解决的风险和问题。

在进行风险评估时，经验丰富的专家往往是不可或缺的。他们能够深入理解软件可能存在的风险点，并据此制定全面的测试计划。然而，对于仅用于演示或不涉及实际应用的软件项目，可能会选择不采用基于风险的测试方法。这种决策通常是基于资源和时间的限制，以及对项目重要性和实际需求的综合考量。

由于风险评估需要专业的评估人员进行判断，而我们组的两位成员并没有相应的经费聘请专家进行风险测试，且我们作为也不具备这种能力，因此不在此进行测试。

4、压力测试

压力测试是性能测试的一种形式，旨在评估系统在特定高负载条件下的稳定性和性能表现。通过模拟极端条件，例如高并发、大量请求或资源占用，压力测试能够分析系统在异常负载下的响应能力、资源消耗情况以及错误处理机制。

压力测试的主要目标包括以下几个方面：

（1）确定系统的最大负载能力：通过测试系统在超出正常工作负载后的表现，明确其所能承受的最大负载水平，找出性能极限。

（2）评估系统稳定性：在负载持续增加或波动的情况下，通过观

察系统的运行状态，发现其在高负载或长时间运行时可能暴露的潜在问题。

(3) 定位性能瓶颈：随着负载的逐步增加，分析系统性能的下降原因，明确导致瓶颈的关键组件或模块。

(4) 验证可扩展性：测试系统在负载增加时的扩展能力，评估其能否高效处理更多的用户或更大的数据量。

在实际实施压力测试时，测试团队通常会借助专业工具来模拟大量用户同时访问系统，或者以其他方式施加高负载。测试结果可以帮助评估系统在不同负载情况下的性能表现，并提供优化和改进的方向。通过这种方式，压力测试确保了系统能够在实际运行环境中保持可靠性和稳定性。

我编写了相应的 Python 脚本，反复上传“探索性测试”中的所有不合法输入，观察系统是否崩溃。由于本人代码能力有限，编写的脚本难以做到短时间内进行大量反复操作，并没有发现系统注册功能的性能瓶颈，测试效果欠佳，建议使用其他测试方法对该功能进行测试。

5、回归测试

回归测试是软件测试中的一种重要测试类型，旨在确保在对软件进行修改、升级或添加新功能后，原有功能仍能正常运行，不会受到新改动的负面影响。其核心目标是验证已有功能的稳定性，防止代码更改引发新的问题。

具体而言，回归测试主要涵盖以下几个方面：

（1）验证功能稳定性：确保核心功能在代码更改后依旧保持稳定，不会因新代码的引入而出现故障。

（2）发现潜在问题：回归测试不仅能防止已有功能被破坏，还能帮助检测由于新功能或代码优化而引入的潜在缺陷。

（3）保证兼容性：特别是在涉及多个组件或系统接口的改动时，回归测试有助于确保新代码的变更不会影响系统的兼容性和正常运行。

（4）验证缺陷修复效果：针对先前版本中发现的缺陷，回归测试可以验证修复是否成功，同时确保修复过程不会引发其他问题。

回归测试在软件开发过程中尤为重要，特别是随着代码的频繁变更。手动执行回归测试可能会耗费大量时间，并容易遗漏问题，因此自动化回归测试成为常见的选择。自动化测试能够快速、高效地验证软件的稳定性和功能完整性，为项目的持续开发提供可靠保障。

针对房源管理系统进行回归测试：

为保证功能修改后，之前的软件仍能正常运行，我们在功能修改后重新运行所有测试用例，即便是不在本次修改范围且已被测试过的功能相关的测试用例也包括在内。

6、探索性测试

探索性测试（Exploratory Testing）是一种灵活且动态的测试方法，特点在于测试人员能够在测试过程中实时设计和执行测试用例。与传统的测试方式不同，探索性测试不依赖于事先制定的详细测试计划或

用例。测试人员会根据他们对系统的理解、经验、直觉以及当前测试目标，动态地创建、执行并调整测试用例。

探索性测试的主要特点包括：

实时设计与执行： 测试人员根据对系统的认知和测试进展，实时生成测试用例，而非依赖事先准备好的详细计划。

依靠专业经验： 测试人员利用其领域知识、经验和判断，灵活应对测试中的变化和未知情况，从而增强测试的适应性。

发现潜在缺陷： 由于测试人员能够即时调整测试方向，探索性测试有助于识别那些事先未能预见的潜在问题和缺陷。

适用于快速变化的环境： 在需求变化快速或项目进展较快的环境中，探索性测试表现出其独特优势，因为它能够迅速适应变化并提供系统稳定性和质量的及时反馈。

高度灵活性和自由度： 测试人员可以根据测试过程中的观察和结果，灵活调整测试策略和方向，从而保证测试过程更具实效性。

尽管探索性测试具有很高的灵活性和动态性，为了确保测试的高效性和有效性，通常仍然需要一定的框架和记录。测试人员通常会记录他们的测试思路、执行的测试步骤及发现的问题，以便更好地组织、总结测试过程并共享测试结果。以下是针对房源管理系统的用户注册功能进行的探索性测试：

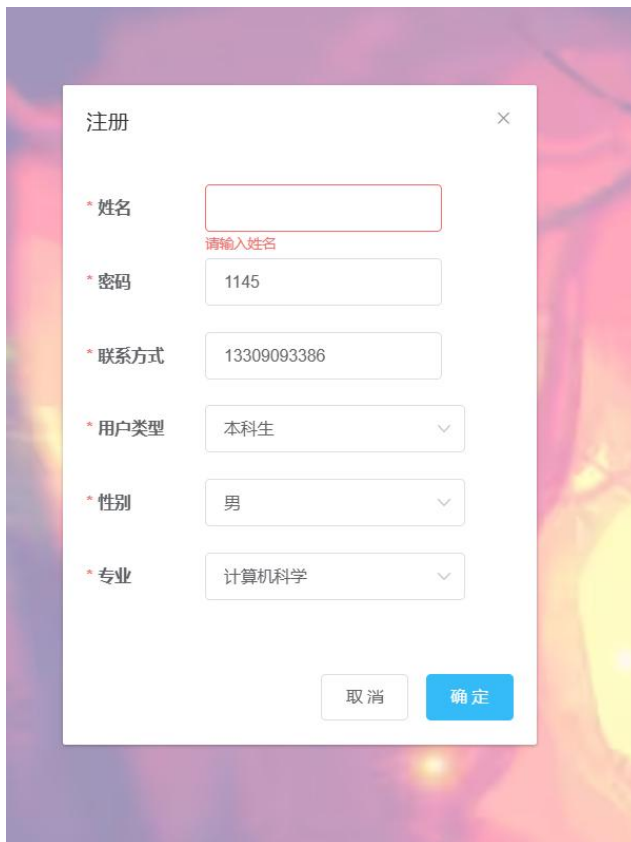
编号	测试输入	预期结果
1	对一个信息均为空的用户信息进行注册	注册失败，提示“请输入密码”、“请输入姓名”、“请输入联系方式”、“请选择角色”、“请选择专业”、“请选择性别”

2	对一个姓名为空、其他信息均保持合法的用户信息进行注册	注册失败，提示“请输入姓名”
3	对一个密码为空、其他信息均保持合法的用户信息进行注册	注册失败，提示“请输入密码”
4	对一个联系方式为空、其他信息均保持合法的用户信息进行注册	注册失败，提示“请输入联系方式”
5	对一个用户类型为空、其他信息均保持合法的用户信息进行注册	注册失败，提示“请选择角色”
6	对一个性别为空、其他信息均保持合法的用户信息进行注册	注册失败，提示“请选择性别”
7	对一个专业为空、其他信息均保持合法的用户信息进行注册	注册失败，提示“请选择专业”
8	对一个所有信息均保持合法的用户信息进行注册	注册成功

实际运行结果如下图所示：

编号 1 运行结果：

编号 2 运行结果：

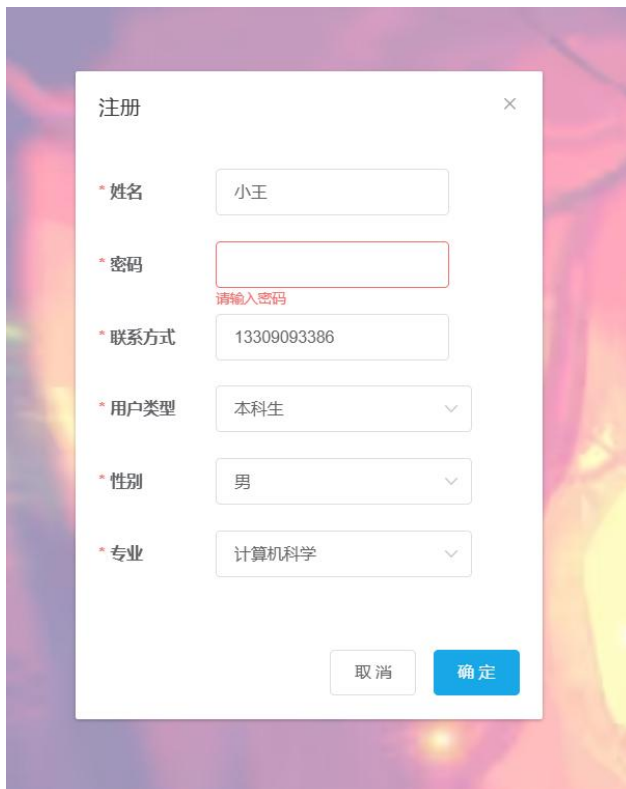


A screenshot of a registration form titled "注册" (Register) with a close button (X) in the top right corner. The form contains the following fields and values:

- * 姓名 (Name): An empty text box with a red border and the error message "请输入姓名" (Please enter name) below it.
- * 密码 (Password): A text box containing the value "1145".
- * 联系方式 (Contact Information): A text box containing the value "13309093386".
- * 用户类型 (User Type): A dropdown menu with "本科生" (Undergraduate) selected.
- * 性别 (Gender): A dropdown menu with "男" (Male) selected.
- * 专业 (Major): A dropdown menu with "计算机科学" (Computer Science) selected.

At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (Confirm).

编号 3 运行结果：

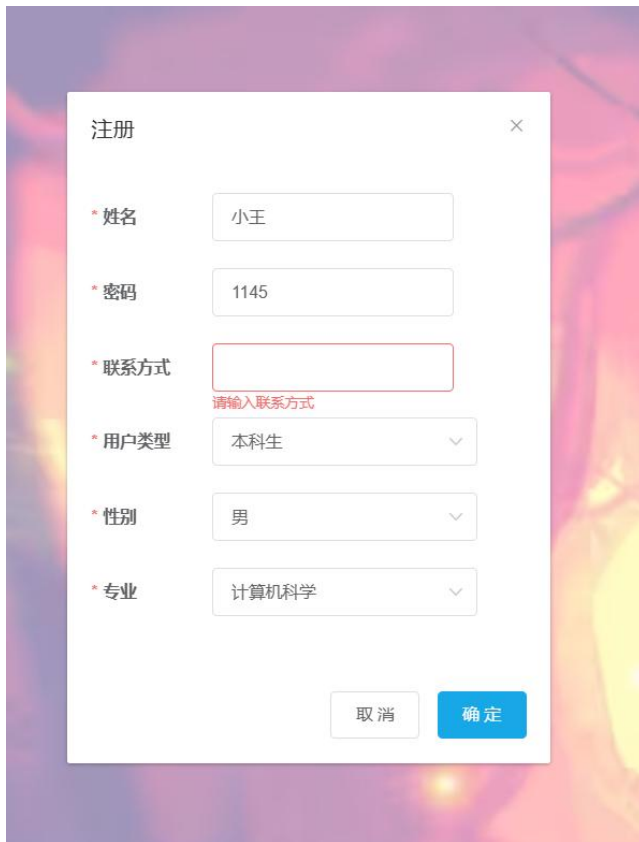


A screenshot of a registration form titled "注册" (Register) with a close button (X) in the top right corner. The form contains the following fields and values:

- * 姓名 (Name): A text box containing the value "小王" (Xiao Wang).
- * 密码 (Password): An empty text box with a red border and the error message "请输入密码" (Please enter password) below it.
- * 联系方式 (Contact Information): A text box containing the value "13309093386".
- * 用户类型 (User Type): A dropdown menu with "本科生" (Undergraduate) selected.
- * 性别 (Gender): A dropdown menu with "男" (Male) selected.
- * 专业 (Major): A dropdown menu with "计算机科学" (Computer Science) selected.

At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (Confirm).

编号 4 运行结果：

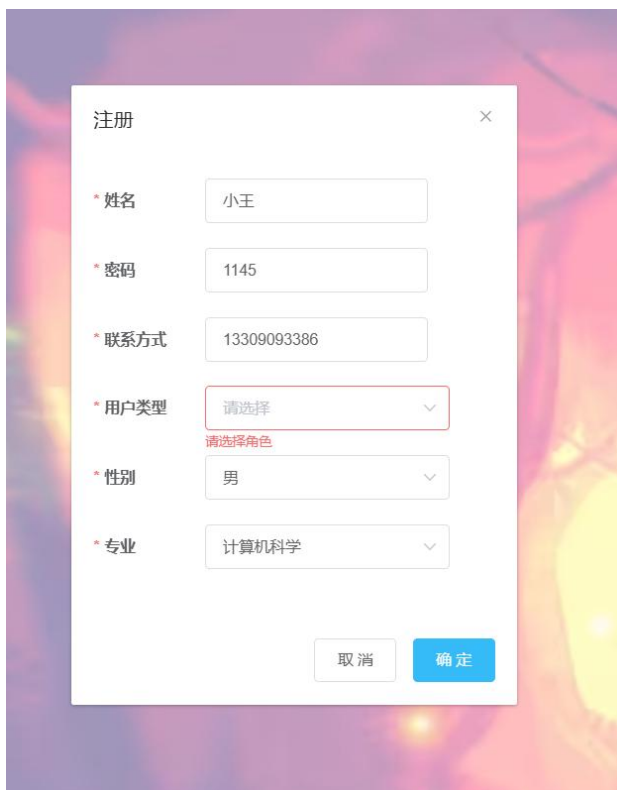


A registration form titled "注册" (Register) with a close button (X) in the top right corner. The form contains the following fields:

- * 姓名 (Name): Text input with value "小王" (Xiao Wang).
- * 密码 (Password): Text input with value "1145".
- * 联系方式 (Contact Information): Text input, currently empty. Below it is a red error message: "请输入联系方式" (Please enter contact information).
- * 用户类型 (User Type): Dropdown menu with value "本科生" (Undergraduate).
- * 性别 (Gender): Dropdown menu with value "男" (Male).
- * 专业 (Major): Dropdown menu with value "计算机科学" (Computer Science).

At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (Confirm).

编号 5 运行结果：

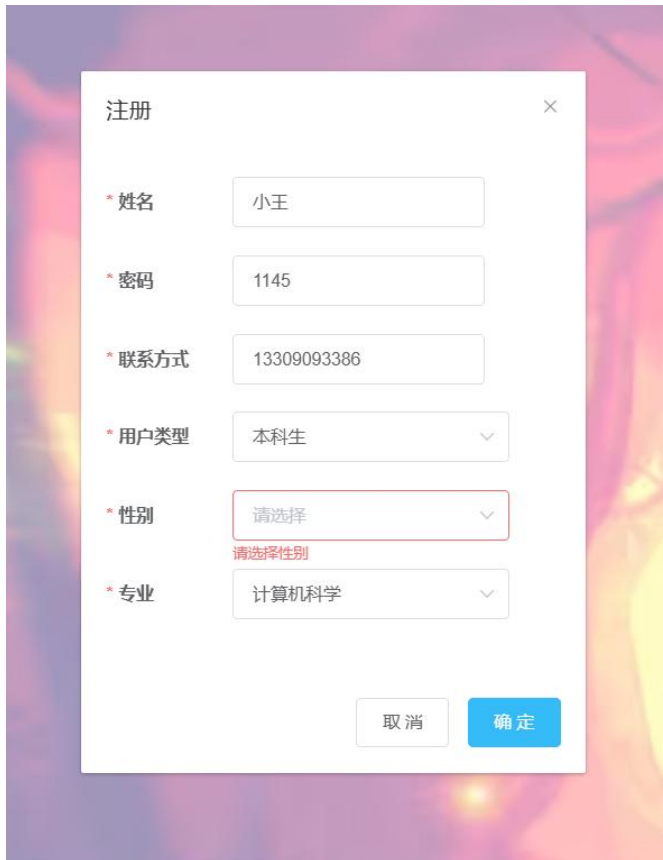


A registration form titled "注册" (Register) with a close button (X) in the top right corner. The form contains the following fields:

- * 姓名 (Name): Text input with value "小王" (Xiao Wang).
- * 密码 (Password): Text input with value "1145".
- * 联系方式 (Contact Information): Text input with value "13309093386".
- * 用户类型 (User Type): Dropdown menu with value "请选择" (Please select). Below it is a red error message: "请选择角色" (Please select a role).
- * 性别 (Gender): Dropdown menu with value "男" (Male).
- * 专业 (Major): Dropdown menu with value "计算机科学" (Computer Science).

At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (Confirm).

编号 6 运行结果：

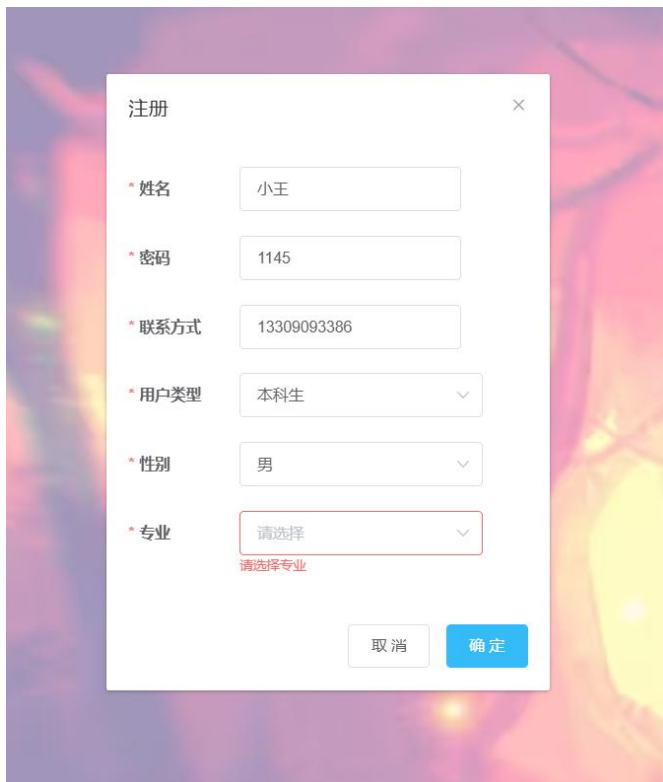


A screenshot of a registration form titled "注册" (Register) with a close button (X) in the top right corner. The form contains the following fields:

- * 姓名 (Name): Text input with value "小王" (Xiao Wang).
- * 密码 (Password): Text input with value "1145".
- * 联系方式 (Contact Information): Text input with value "13309093386".
- * 用户类型 (User Type): Dropdown menu with value "本科生" (Undergraduate).
- * 性别 (Gender): Dropdown menu with value "请选择" (Please select). Below the dropdown is a red error message "请选择性别" (Please select gender).
- * 专业 (Major): Dropdown menu with value "计算机科学" (Computer Science).

At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (Confirm).

编号 7 运行结果：




A screenshot of a registration form titled "注册" (Register) with a close button (X) in the top right corner. The form contains the following fields:

- * 姓名 (Name): Text input with value "小王" (Xiao Wang).
- * 密码 (Password): Text input with value "1145".
- * 联系方式 (Contact Information): Text input with value "13309093386".
- * 用户类型 (User Type): Dropdown menu with value "本科生" (Undergraduate).
- * 性别 (Gender): Dropdown menu with value "男" (Male).
- * 专业 (Major): Dropdown menu with value "请选择" (Please select). Below the dropdown is a red error message "请选择专业" (Please select major).

At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (Confirm).

编号 8 运行结果：



The image shows a registration form titled '注册' (Registration) with a close button '×'. The form contains the following fields:

- * 姓名 (Name): 小王
- * 密码 (Password): 1145
- * 联系方式 (Contact Information): 1309093386
- * 用户类型 (User Type): 本科生 (Undergraduate)
- * 性别 (Gender): 男 (Male)
- * 专业 (Major): 计算机科学 (Computer Science)

At the bottom of the form are two buttons: '取消' (Cancel) and '确定' (Confirm).

Below the form, a green message box displays: '注册成功,你的工号是: 11113' (Registration successful, your employee number is: 11113).

7、用户测试

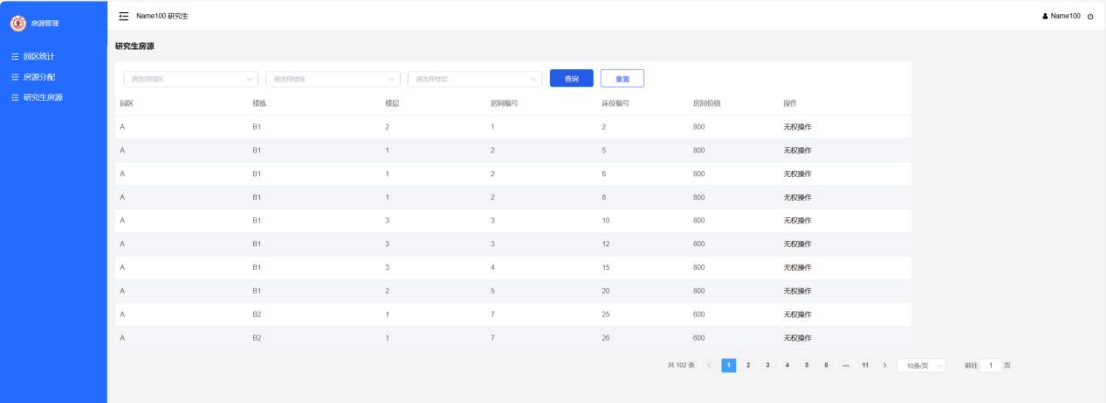
用户测试是一种以实际用户为核心的测试方法，通常在验收测试阶段进行。除了验证软件的功能性需求外，用户测试还涉及多个非功能性方面，例如用户界面的易用性、信息的可理解性、是否充分考虑了用户的背景、经验和技能水平、错误信息的清晰度和可操作性。这种测试方法的核心是通过模拟真实用户的使用情境，收集反馈，确保产品不仅在功能上达标，而且符合用户的期望和操作习惯。

从用户易用性以及稳定性的角度出发可以得到以下测试用例，本用例测试的功能是用户对房源进行查询的功能：

编号	测试输入	预期结果
1	在首页点击“本科生房源”/“研究生房源”（根据登录的用户是本科生还是研究生决定向该用户开放哪一个房源查询页面）	进入“本科生房源”/“研究生房源”页面
2	点击“请选择园区”所在下拉框区域，筛选想查询的园区	“请选择园区”下拉框中显示用户选择的园区
3	点击“请选择楼栋”所在下拉框区域，筛选想查询的楼栋	“请选择楼栋”下拉框中显示用户选择的楼栋
4	点击“请选择楼层”所在下拉框区域，筛选想查询的楼层	“请选择楼层”下拉框中显示用户选择的楼层
5	点击“查询”按钮进行查询	显示筛选后的房源信息
6	点击“重置”按钮重置筛选框	三个筛选框回到初始状态
7	点击分页栏的页码或“<”、“>”按钮切换页面	切换房源信息的页数
8	在“前往 页”中输入想浏览的页数，点击回车键	切换到用户输入的房源信息的页数

实际运行结果如下图所示：

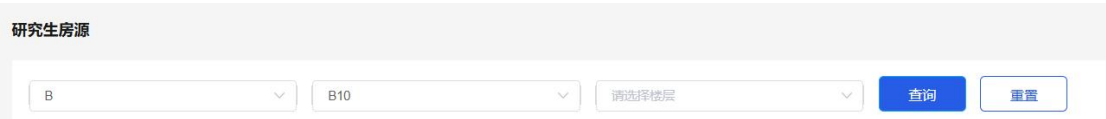
编号 1 运行结果：



编号 2 运行结果：



编号 3 运行结果：



编号 4 运行结果：

研究生房源

B

B10

1F

查询

重置

编号 5 运行结果：

研究生房源

校区	楼栋	楼层	房间编号	床位编号	房间价格	操作
B	B10	1	31	107	500	无权操作
B	B10	1	31	108	500	无权操作

共 2 条 < 1 > 10条/页 前往 1 页

编号 6 运行结果：

研究生房源

校区	楼栋	楼层	房间编号	床位编号	房间价格	操作
A	B1	2	1	2	800	无权操作
A	B1	1	2	5	800	无权操作
A	B1	1	2	6	800	无权操作
A	B1	1	2	8	800	无权操作
A	B1	3	3	10	800	无权操作
A	B1	3	3	12	800	无权操作
A	B1	3	4	15	800	无权操作
A	B1	2	5	20	800	无权操作
A	B2	1	7	25	600	无权操作
A	B2	1	7	26	600	无权操作

共 102 条 < 1 2 3 4 5 6 ... 11 > 10条/页 前往 1 页

编号 7 运行结果：

研究生房源

校区	楼栋	楼层	房间编号	床位编号	房间价格	操作
B	B7	1	22	81	500	无权操作
B	B7	3	24	85	300	无权操作
B	B8	1	25	90	500	无权操作
B	B8	2	26	93	400	无权操作
B	B8	3	27	96	300	无权操作
B	B8	3	27	97	300	无权操作
B	B9	1	28	98	500	无权操作
B	B9	1	28	99	500	无权操作
B	B9	2	29	100	400	无权操作
B	B9	2	29	101	400	无权操作

共 102 条 < 1 ... 3 4 5 6 7 ... 11 > 10条/页 前往 5 页

编号 8 运行结果：

研究生房源						
请选择校区		请选择楼栋		请选择楼层		
				查询		重置
校区	楼栋	楼层	房间编号	床位编号	房间价格	操作
D	B18	2	56	181	400	无权限操作
D	B18	2	56	182	400	无权限操作
D	B18	2	56	183	400	无权限操作
D	B18	3	57	185	300	无权限操作
D	B18	3	57	186	300	无权限操作
D	B18	3	57	187	300	无权限操作
D	B19	1	58	188	500	无权限操作
D	B19	2	59	192	400	无权限操作
D	B19	3	60	193	300	无权限操作
D	B19	3	60	195	300	无权限操作

8、场景测试

系统的控制流程通常由触发事件引发。当事件发生时，所表现出的情境被称为场景，有时也被称为“故事”或使用情境。在同一事件触发的情况下，因触发顺序或处理结果的不同，会形成不同的事件流。

场景测试是软件测试中的一种方法，主要通过使用场景来描述系统的功能或业务流程，以提升测试效果。这种方法从用户的角度分析软件的应用场景，通过这些场景来设计测试用例。场景测试特别关注用户在实际使用过程中可能遇到的各种情况，确保系统能够在不同的使用情境下稳定、正确地运行。场景测试的目标是从实际使用的视角出发，通过构建和测试多个场景，提升系统功能和业务流程的覆盖范围。这种方法有助于发现潜在的缺陷，并确保系统在实际环境中的稳定性和可靠性。

对于一个系统而言，控制流程大多由时间触发事件的发生来驱动。每个事件的触发情境形成了一个场景，而相同事件在不同触发顺序和处理结果下的变化则构成了事件流。

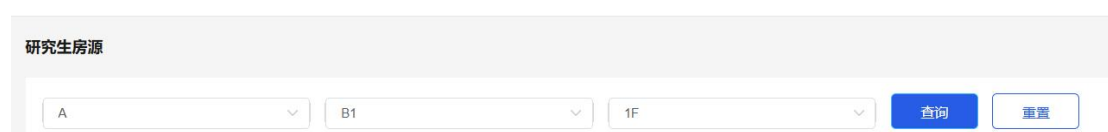
对用户房源进行查询这一功能进行场景测试：

- ①用户已经相中某一园区某一楼栋某一楼层的房间，对园区、楼栋、楼层依次进行筛选，再进行查询操作；
- ②用户已经相中了某个园区的某个楼栋，对园区、楼栋进行筛选，后进行查询操作；
- ③用户已经相中了某一园区且对楼层有要求，对园区、楼层进行筛选，后进行查询操作；
- ④用户仅对园区有要求，对园区进行筛选后直接进行查询操作；
- ⑤用户仅对楼栋有要求，对楼栋进行筛选后直接进行查询操作；
- ⑥用户仅对楼层有要求，对楼层进行筛选后直接进行查询操作；
- ⑦用户还想查看其它房源信息，点击重置按钮重置筛选功能；

实际运行结果如下图所示：

事件流 1：

- (1) 用户对园区、楼栋、楼层依次进行筛选



- (2) 点击查询按钮



园区	楼栋	楼层	房间编号	床位编号	房间价格	操作
A	B1	1	2	5	800	无权操作
A	B1	1	2	6	800	无权操作
A	B1	1	2	8	800	无权操作

事件流 2：

- (1) 用户对园区、楼栋依次进行筛选

研究生房源

A

B1

请选择楼层

查询

重置

(2) 点击查询按钮

研究生房源

园区	楼栋	楼层	房间编号	床位编号	房间价格	操作
A	B1	2	1	2	800	无权操作
A	B1	1	2	5	800	无权操作
A	B1	1	2	6	800	无权操作
A	B1	1	2	8	800	无权操作
A	B1	3	3	10	800	无权操作
A	B1	3	3	12	800	无权操作
A	B1	3	4	15	800	无权操作
A	B1	2	5	20	800	无权操作

共 8 条 < 1 > 10条/页 前往 1 页

事件流 3:

(1) 用户对园区、楼层依次进行筛选

研究生房源

A

请选择楼栋

1F

查询

重置

(2) 点击查询按钮

研究生房源

园区	楼栋	楼层	房间编号	床位编号	房间价格	操作
A	B1	1	2	5	800	无权操作
A	B1	1	2	6	800	无权操作
A	B1	1	2	8	800	无权操作
A	B2	1	7	25	600	无权操作
A	B2	1	7	26	600	无权操作
A	B2	1	7	27	600	无权操作
A	B2	1	7	28	600	无权操作
A	B2	1	8	29	500	无权操作
A	B2	1	8	30	500	无权操作
A	B2	1	8	31	500	无权操作

共 14 条 < 1 2 > 10条/页 前往 1 页

事件流 4:

(1) 用户对园区进行筛选

研究生房源

A

请选择楼栋

请选择楼层

查询

重置

(2) 点击查询按钮

研究生房源

A

请选择楼栋

请选择楼层

查询

重置

园区	楼栋	楼层	房间编号	床位编号	房间价格	操作
A	B1	2	1	2	800	无权操作
A	B1	1	2	5	800	无权操作
A	B1	1	2	6	800	无权操作
A	B1	1	2	8	800	无权操作
A	B1	3	3	10	800	无权操作
A	B1	3	3	12	800	无权操作
A	B1	3	4	15	800	无权操作
A	B1	2	5	20	800	无权操作
A	B2	1	7	25	600	无权操作
A	B2	1	7	26	600	无权操作

共 35 条 < 1 2 3 4 > 10条/页 前往 1 页

事件流 5:

(1) 用户对楼栋进行筛选

研究生房源

请选择园区

B8

请选择楼层

查询

重置

(2) 点击查询按钮

研究生房源

请选择园区

B8

请选择楼层

查询

重置

园区	楼栋	楼层	房间编号	床位编号	房间价格	操作
B	B8	1	25	90	500	无权操作
B	B8	2	26	93	400	无权操作
B	B8	3	27	96	300	无权操作
B	B8	3	27	97	300	无权操作

共 4 条 < 1 > 10条/页 前往 1 页

事件流 6:

(1) 用户对楼层进行筛选

研究生房源

请选择园区

请选择楼栋

3F

查询

重置

(2) 点击查询按钮

研究生房源

请选择园区

请选择楼栋

3F

查询

重置

园区	楼栋	楼层	房间编号	床位编号	房间价格	操作
A	B1	3	3	10	800	无权操作
A	B1	3	3	12	800	无权操作
A	B1	3	4	15	800	无权操作
A	B2	3	9	33	200	无权操作
A	B3	3	11	41	400	无权操作
A	B3	3	11	43	400	无权操作
A	B3	3	11	44	400	无权操作
A	B5	3	18	67	300	无权操作
A	B5	3	18	68	300	无权操作
B	B6	3	21	76	300	无权操作

共 38 条

< 1 2 3 4 >

10条/页

前往 1 页

事件流 7:

(1) 用户随机进行了一次查询操作:

研究生房源

C	B12	3F	查询	重置
---	-----	----	----	----

(2) 用户点击重置按钮, 进行下一次查询操作:

研究生房源

请选择园区	请选择楼栋	请选择楼层	查询	重置
-------	-------	-------	----	----

9、随机测试

随机测试是一种通过随机生成测试输入来评估系统性能和鲁棒性的测试方法。在不同的测试场景中, 随机测试可能采取不同的形式, 取决于具体的测试上下文和目标。

与传统的测试方法不同, 随机测试不依赖于预先设定的测试用例, 而是通过引入随机性来模拟多种可能的输入情况。测试人员通过向系统输入随机生成的数据、事件或参数, 观察系统在不同条件下的反应, 从而揭示潜在的缺陷、漏洞或性能瓶颈。

随机测试的灵活性使其适用于多种应用场景，尤其是在面对复杂系统或无法覆盖所有输入情况时，能够提供广泛的测试覆盖。然而，考虑到其随机性，随机测试往往需要较多的资源和时间来执行，且测试结果的分析 and 解读可能相对复杂。尽管如此，随机测试依然是一个有效的工具，能够帮助发现系统中潜在的、难以预见的问题。

由于随机测试耗费时间、资源多且难以直观显示问题所在，故不在此深入进行测试。

10、功能测试

功能测试是一种黑盒测试方法，主要用于验证系统中各个独立功能单元是否按需求规格说明书的要求正常工作。每个功能单元都会经过详细的测试，旨在全面发现和确认潜在的缺陷。功能测试的重点在于验证单个功能是否能够按预期执行，不关注其内部实现细节。

其主要优点是实施简便，容易执行，且能够快速验证系统中每个功能的正确性。但由于功能测试通常集中在单个功能单元的验证上，它有一定的局限性，无法发现不同功能单元之间可能存在的数据传输问题或接口定义错误。

进行功能测试时，首先需要明确定义待测试的单元，这些单元可以是一个函数、类或模块等。明确单元后，测试人员需要根据系统需求分析出哪些功能需要被验证，并为这些功能设计测试用例。

功能测试在确保系统各个功能单元的正确性和独立性方面起到了重要作用，但为了确保系统在整个层面上的稳定性和一致性，还需要

结合集成测试等其他测试类型，特别是在多个功能单元协作的场景下。综合使用不同层次的测试方法有助于提高测试覆盖率，确保软件系统的质量和可靠性。

由于在上述九种不同的测试方法的叙述中已经将房源管理系统的几个主要功能进行了测试，故不再此处赘述。

四、JUnit 测试

针对房源管理系统后端代码 `model` 层下的 `domain` 包，我们对其中的 `Admin` 类、`Bed` 类、`Building` 类、`Park` 类、`Room` 类、`Student` 类、`User` 类进行了 JUnit 测试。

以下是测试代码（`import` 没有在此申明，完整代码在电子版附件）：

对 `Admin` 类的测试：

```
public class AdminTest {
    private Admin admin;

    @Before
    public void setUp() {
        // 初始化 Admin 实例
        admin = new Admin();
    }

    @Test
    public void testSetAndGetId() {
        admin.setId(1L);
        Assert.assertEquals(Long.valueOf(1L), admin.getId());
    }

    @Test
    public void testSetAndGetLoginName() {
        admin.setLoginName("123");
        Assert.assertEquals("123", admin.getLoginName());
    }
}
```

```

@Test
public void testSetAndGetPassword() {
    admin.setPassword("password");
    Assert.assertEquals("password", admin.getPassword());
}

@Test
public void testSetAndGetLastLoginTime() {
    LocalDateTime dateTime = LocalDateTime.now();
    admin.setLastLoginTime(dateTime);
    Assert.assertEquals(dateTime, admin.getLastLoginTime());

    // 测试为空
    admin.setLastLoginTime(null);
    Assert.assertNull(admin.getLastLoginTime());
}

@Test
public void testSetAndGetRemark() {
    admin.setRemark("Test Remark");
    Assert.assertEquals("Test Remark", admin.getRemark());
}

@Test
public void testSetAndGetDeleted() {
    // 测试设置为 true
    admin.setDeleted(true);
    Assert.assertTrue(admin.getDeleted());

    // 测试设置为 false
    admin.setDeleted(false);
    Assert.assertFalse(admin.getDeleted());
}

@Test
public void testSetAndGetStudents() {
    // 测试设置为空列表
    admin.setStudents(Collections.emptyList());
    Assert.assertTrue(admin.getStudents().isEmpty());

    // 测试设置 null
    admin.setStudents(null);
    Assert.assertNull(admin.getStudents());
}

```

```

}

@Test
public void testPkVal() {
    admin.setId(1L);
    Assert.assertEquals(Long.valueOf(1L), admin.pkVal());
}

@Test
public void testEqualsAndHashCodeWithDifferentIds() {
    Admin admin1 = new Admin().setId(1L).setLoginName("123");
    Admin admin2 = new Admin().setId(2L).setLoginName("123");

    // 断言不同 id 的对象不相等
    Assert.assertFalse(admin1.equals(admin2));
    Assert.assertNotEquals(admin1.hashCode(), admin2.hashCode());
}

@Test
public void testEqualsAndHashCodeWithSameIds() {
    Admin admin1 = new Admin().setId(1L).setLoginName("123");
    Admin admin2 = new Admin().setId(1L).setLoginName("123");

    // 断言相同 id 的对象相等
    Assert.assertTrue(admin1.equals(admin2));
    Assert.assertEquals(admin1.hashCode(), admin2.hashCode());
}

@Test
public void testChainSetters() {
    admin.setId(1L)
        .setLoginName("123")
        .setPassword("password")
        .setRemark("Test remark")
        .setDeleted(true);

    Assert.assertEquals(Long.valueOf(1L), admin.getId());
    Assert.assertEquals("123", admin.getLoginName());
    Assert.assertEquals("password", admin.getPassword());
    Assert.assertEquals("Test remark", admin.getRemark());
    Assert.assertTrue(admin.getDeleted());
}

@Test

```

```

public void testSetAndGetDeletedWithLogicDelete() {
    // 测试逻辑删除字段
    admin.setDeleted(true);
    Assert.assertTrue(admin.getDeleted());

    admin.setDeleted(false);
    Assert.assertFalse(admin.getDeleted());
}

@Test
public void testSetAndGetNullLoginName() {
    admin.setLoginName(null);
    Assert.assertNull(admin.getLoginName());
}

@Test
public void testSetInvalidLoginName() {
    // 模拟一个非法的 loginName
    admin.setLoginName("invalid123");
    Assert.assertEquals("invalid123", admin.getLoginName());
}

@Test
public void testSetInvalidPassword() {
    admin.setPassword("");
    Assert.assertEquals("", admin.getPassword());
}
}

```

对 Bed 类的测试:

```

public class BedTest {
    private Bed bed;

    @Before
    public void setUp() {
        // 初始化 Bed 实例
        bed = new Bed();
    }

    @Test
    public void testSetAndGetId() {
        bed.setId(1);
        Assert.assertEquals(Integer.valueOf(1), bed.getId());
    }
}

```



```

@Test
public void testSetAndGetIsOwned() {
    bed.setIsOwned(1);
    Assert.assertEquals(Integer.valueOf(1), bed.getIsOwned());

    bed.setIsOwned(0);
    Assert.assertEquals(Integer.valueOf(0), bed.getIsOwned());
}

@Test
public void testSetAndGetRoomId() {
    bed.setRoomId(101);
    Assert.assertEquals(Integer.valueOf(101), bed.getRoomId());
}

@Test
public void testSetAndGetIsDeleted() {
    bed.setIsDeleted(1);
    Assert.assertEquals(Integer.valueOf(1), bed.getIsDeleted());

    bed.setIsDeleted(0);
    Assert.assertEquals(Integer.valueOf(0), bed.getIsDeleted());
}

@Test
public void testPkVal() {
    bed.setId(1);
    Assert.assertEquals(Integer.valueOf(1), bed.pkVal());
}

@Test
public void testEqualsAndHashCodeWithDifferentIds() {
    Bed bed1 = new Bed().setId(1).setRoomId(101);
    Bed bed2 = new Bed().setId(2).setRoomId(101);

    // 断言不同 id 的对象不相等
    Assert.assertFalse(bed1.equals(bed2));
    Assert.assertNotEquals(bed1.hashCode(), bed2.hashCode());
}

@Test
public void testSetNullRoom() {
    bed.setRoom(null);
}

```

```

        Assert.assertNull(bed.getRoom());
    }

    @Test
    public void testSetNullUser() {
        bed.setUser(null);
        Assert.assertNull(bed.getUser());
    }

    @Test
    public void testSetAndGetIsOwnedWithInvalidValue() {
        // 测试 isOwned 字段的有效性
        bed.setIsOwned(2);
        Assert.assertEquals(Integer.valueOf(2), bed.getIsOwned());

        bed.setIsOwned(-1);
        Assert.assertEquals(Integer.valueOf(-1), bed.getIsOwned());
    }

    @Test
    public void testSetAndGetIsDeletedWithInvalidValue() {
        // 测试 isDeleted 字段的有效性
        bed.setIsDeleted(2);
        Assert.assertEquals(Integer.valueOf(2), bed.getIsDeleted());

        bed.setIsDeleted(-1);
        Assert.assertEquals(Integer.valueOf(-1), bed.getIsDeleted());
    }
}

```

对 Building 类的测试:

```

public class BuildingTest {
    private Building building;

    @Before
    public void setUp() {
        // 初始化 Building 实例
        building = new Building();
    }

    @Test
    public void testSetAndGetName() {
        building.setName("Building 1");
        Assert.assertEquals("Building 1", building.getName());
    }
}

```

```

}

@Test
public void testSetAndGetFloornumber() {
    building.setFloornumber(10);
    Assert.assertEquals(Integer.valueOf(10), building.getFloornumber());
}

@Test
public void testSetAndGetRoomtype() {
    building.setRoomtype("Single");
    Assert.assertEquals("Single", building.getRoomtype());
}

@Test
public void testSetAndGetRoomcount() {
    building.setRoomcount(100);
    Assert.assertEquals(Integer.valueOf(100), building.getRoomcount());
}

@Test
public void testSetAndGetBedcount() {
    building.setBedcount(200);
    Assert.assertEquals(Integer.valueOf(200), building.getBedcount());
}

@Test
public void testSetAndGetParkName() {
    building.setParkName("Park A");
    Assert.assertEquals("Park A", building.getParkName());
}

@Test
public void testSetAndGetIsDeleted() {
    building.setIsDeleted(true);
    Assert.assertTrue(building.getIsDeleted());

    building.setIsDeleted(false);
    Assert.assertFalse(building.getIsDeleted());
}

@Test
public void testSetAndGetGender() {
    building.setGender("Male");

```

```

        Assert.assertEquals("Male", building.getGender());
    }

    @Test
    public void testSetAndGetPhoneNumber() {
        building.setPhoneNumber("1234567890");
        Assert.assertEquals("1234567890", building.getPhoneNumber());
    }

    @Test
    public void testSetAndGetPropertyManagement() {
        building.setPropertyManagement("XYZ Property");
        Assert.assertEquals("XYZ Property", building.getPropertyManagement());
    }

    @Test
    public void testSetAndGetBuildingManager() {
        building.setBuildingManager("John Doe");
        Assert.assertEquals("John Doe", building.getBuildingManager());
    }

    @Test
    public void testPkVal() {
        building.setName("Building 1");
        Assert.assertEquals("Building 1", building.pkVal());
    }

    @Test
    public void testEqualsAndHashCodeWithDifferentNames() {
        Building building1 = new Building().setName("Building 1");
        Building building2 = new Building().setName("Building 2");

        // 断言不同名称的对象不相等
        Assert.assertFalse(building1.equals(building2));
        Assert.assertNotEquals(building1.hashCode(), building2.hashCode());
    }

    @Test
    public void testEqualsAndHashCodeWithSameNames() {
        Building building1 = new Building().setName("Building 1");
        Building building2 = new Building().setName("Building 1");

        // 断言相同名称的对象相等
        Assert.assertTrue(building1.equals(building2));
    }

```

```

        Assert.assertEquals(building1.hashCode(), building2.hashCode());
    }

    @Test
    public void testChainSetters() {
        building.setName("Building 1")
            .setFloornumber(10)
            .setRoomtype("Single")
            .setRoomcount(100)
            .setBedcount(200)
            .setParkName("Park A")
            .setIsDeleted(true)
            .setGender("Male")
            .setPhoneNumber("1234567890")
            .setPropertyManagement("XYZ Property")
            .setBuildingManager("John Doe");

        Assert.assertEquals("Building 1", building.getName());
        Assert.assertEquals(Integer.valueOf(10), building.getFloornumber());
        Assert.assertEquals("Single", building.getRoomtype());
        Assert.assertEquals(Integer.valueOf(100), building.getRoomcount());
        Assert.assertEquals(Integer.valueOf(200), building.getBedcount());
        Assert.assertEquals("Park A", building.getParkName());
        Assert.assertTrue(building.getIsDeleted());
        Assert.assertEquals("Male", building.getGender());
        Assert.assertEquals("1234567890", building.getPhoneNumber());
        Assert.assertEquals("XYZ Property", building.getPropertyManagement());
        Assert.assertEquals("John Doe", building.getBuildingManager());
    }

    @Test
    public void testSetNullValues() {
        building.setGender(null);
        Assert.assertNull(building.getGender());

        building.setPhoneNumber(null);
        Assert.assertNull(building.getPhoneNumber());

        building.setPropertyManagement(null);
        Assert.assertNull(building.getPropertyManagement());

        building.setBuildingManager(null);
        Assert.assertNull(building.getBuildingManager());
    }

```

```

@Test
public void testSetInvalidFloornumber() {
    building.setFloornumber(-1);
    Assert.assertEquals(Integer.valueOf(-1), building.getFloornumber());
}

@Test
public void testSetInvalidRoomcount() {
    building.setRoomcount(-10);
    Assert.assertEquals(Integer.valueOf(-10), building.getRoomcount());
}

@Test
public void testSetInvalidBedcount() {
    building.setBedcount(-20);
    Assert.assertEquals(Integer.valueOf(-20), building.getBedcount());
}

@Test
public void testSetInvalidIsDeleted() {
    building.setIsDeleted(null);
    Assert.assertNull(building.getIsDeleted());
}
}

```

对 **Park** 类的测试:

```

public class ParkTest {
    private Park park;

    @Before
    public void setUp() {
        // 初始化 Park 实例
        park = new Park();
    }

    @Test
    public void testSetAndGetName() {
        park.setName("Park A");
        Assert.assertEquals("Park A", park.getName());
    }

    @Test
    public void testSetAndGetCampus() {

```

```

        park.setCampus("Campus 1");
        Assert.assertEquals("Campus 1", park.getCampus());
    }

    @Test
    public void testSetAndGetBuildingcount() {
        park.setBuildingcount(10);
        Assert.assertEquals(Integer.valueOf(10), park.getBuildingcount());
    }

    @Test
    public void testSetAndGetRoomcount() {
        park.setRoomcount(100);
        Assert.assertEquals(Integer.valueOf(100), park.getRoomcount());
    }

    @Test
    public void testSetAndGetBedcount() {
        park.setBedcount(200);
        Assert.assertEquals(Integer.valueOf(200), park.getBedcount());
    }

    @Test
    public void testSetAndGetIsDeleted() {
        park.setIsDeleted(true);
        Assert.assertTrue(park.getIsDeleted());

        park.setIsDeleted(false);
        Assert.assertFalse(park.getIsDeleted());
    }

    @Test
    public void testPkVal() {
        park.setName("Park A");
        Assert.assertEquals("Park A", park.pkVal());
    }

    @Test
    public void testEqualsAndHashCodeWithDifferentNames() {
        Park park1 = new Park().setName("Park A");
        Park park2 = new Park().setName("Park B");

        // 断言不同名称的对象不相等
        Assert.assertFalse(park1.equals(park2));
    }

```

```

        Assert.assertNotEquals(park1.hashCode(), park2.hashCode());
    }

    @Test
    public void testEqualsAndHashCodeWithSameNames() {
        Park park1 = new Park().setName("Park A");
        Park park2 = new Park().setName("Park A");

        // 断言相同名称的对象相等
        Assert.assertTrue(park1.equals(park2));
        Assert.assertEquals(park1.hashCode(), park2.hashCode());
    }

    @Test
    public void testChainSetters() {
        park.setName("Park A")
            .setCampus("Campus 1")
            .setBuildingcount(10)
            .setRoomcount(100)
            .setBedcount(200)
            .setIsDeleted(true);

        Assert.assertEquals("Park A", park.getName());
        Assert.assertEquals("Campus 1", park.getCampus());
        Assert.assertEquals(Integer.valueOf(10), park.getBuildingcount());
        Assert.assertEquals(Integer.valueOf(100), park.getRoomcount());
        Assert.assertEquals(Integer.valueOf(200), park.getBedcount());
        Assert.assertTrue(park.getIsDeleted());
    }

    @Test
    public void testSetNullValues() {
        park.setCampus(null);
        Assert.assertNull(park.getCampus());

        park.setIsDeleted(null);
        Assert.assertNull(park.getIsDeleted());
    }

    @Test
    public void testSetInvalidBuildingcount() {
        park.setBuildingcount(-1);
        Assert.assertEquals(Integer.valueOf(-1), park.getBuildingcount());
    }
}

```



```

@Test
public void testSetInvalidRoomcount() {
    park.setRoomcount(-10);
    Assert.assertEquals(Integer.valueOf(-10), park.getRoomcount());
}

@Test
public void testSetInvalidBedcount() {
    park.setBedcount(-20);
    Assert.assertEquals(Integer.valueOf(-20), park.getBedcount());
}

@Test
public void testSetInvalidIsDeleted() {
    park.setIsDeleted(null);
    Assert.assertNull(park.getIsDeleted());
}
}

```

对 Room 类的测试:

```

public class RoomTest {
    private Room room;

    @Before
    public void setUp() {
        // 初始化 Room 实例
        room = new Room();
    }

    @Test
    public void testSetAndGetId() {
        room.setId(1);
        Assert.assertEquals(Integer.valueOf(1), room.getId());
    }

    @Test
    public void testSetAndGetFloor() {
        room.setFloor(3);
        Assert.assertEquals(Integer.valueOf(3), room.getFloor());
    }

    @Test
    public void testSetAndGetBedcount() {

```

```
        room.setBedcount(2);
        Assert.assertEquals(Integer.valueOf(2), room.getBedcount());
    }
}
```

```
@Test
public void testSetAndGetIsEnabled() {
    room.setIsEnabled(true);
    Assert.assertTrue(room.getIsEnabled());

    room.setIsEnabled(false);
    Assert.assertFalse(room.getIsEnabled());
}
}
```

```
@Test
public void testSetAndGetPrice() {
    room.setPrice(5000);
    Assert.assertEquals(Integer.valueOf(5000), room.getPrice());
}
}
```

```
@Test
public void testSetAndGetBuildingName() {
    room.setBuildingName("Building A");
    Assert.assertEquals("Building A", room.getBuildingName());
}
}
```

```
@Test
public void testSetAndGetParkName() {
    room.setParkName("Park 1");
    Assert.assertEquals("Park 1", room.getParkName());
}
}
```

```
@Test
public void testSetAndGetIsDeleted() {
    room.setIsDeleted(true);
    Assert.assertTrue(room.getIsDeleted());

    room.setIsDeleted(false);
    Assert.assertFalse(room.getIsDeleted());
}
}
```

```
@Test
public void testPkVal() {
    room.setId(1);
    Assert.assertEquals(Integer.valueOf(1), room.pkVal());
}
```

```

}

@Test
public void testEqualsAndHashCodeWithDifferentIds() {
    Room room1 = new Room().setId(1);
    Room room2 = new Room().setId(2);

    // 断言不同 ID 的对象不相等
    Assert.assertFalse(room1.equals(room2));
    Assert.assertNotEquals(room1.hashCode(), room2.hashCode());
}

@Test
public void testEqualsAndHashCodeWithSameIds() {
    Room room1 = new Room().setId(1);
    Room room2 = new Room().setId(1);

    // 断言相同 ID 的对象相等
    Assert.assertTrue(room1.equals(room2));
    Assert.assertEquals(room1.hashCode(), room2.hashCode());
}

@Test
public void testChainSetters() {
    room.setId(1)
        .setFloor(3)
        .setBedcount(2)
        .setIsEnabled(true)
        .setPrice(5000)
        .setBuildingName("Building A")
        .setParkName("Park 1")
        .setIsDeleted(false);

    Assert.assertEquals(Integer.valueOf(1), room.getId());
    Assert.assertEquals(Integer.valueOf(3), room.getFloor());
    Assert.assertEquals(Integer.valueOf(2), room.getBedcount());
    Assert.assertTrue(room.getIsEnabled());
    Assert.assertEquals(Integer.valueOf(5000), room.getPrice());
    Assert.assertEquals("Building A", room.getBuildingName());
    Assert.assertEquals("Park 1", room.getParkName());
    Assert.assertFalse(room.getIsDeleted());
}

@Test

```

```

public void testSetNullValues() {
    room.setBuildingName(null);
    Assert.assertNull(room.getBuildingName());

    room.setParkName(null);
    Assert.assertNull(room.getParkName());

    room.setIsDeleted(null);
    Assert.assertNull(room.getIsDeleted());
}

@Test
public void testSetInvalidValues() {
    room.setPrice(-1000);
    Assert.assertEquals(Integer.valueOf(-1000), room.getPrice());

    room.setBedcount(-1);
    Assert.assertEquals(Integer.valueOf(-1), room.getBedcount());
}
}

```

对 Student 类的测试:

```

public class StudentTest {
    private Student student;

    @Before
    public void setUp() {
        // 初始化 Student 实例
        student = new Student();
    }

    @Test
    public void testSetAndGetId() {
        student.setId(1L);
        Assert.assertEquals(Long.valueOf(1L), student.getId());
    }

    @Test
    public void testSetAndGetName() {
        student.setName("John Doe");
        Assert.assertEquals("John Doe", student.getName());
    }

    @Test

```

```

public void testSetAndGetSn() {
    student.setSn("S12345");
    Assert.assertEquals("S12345", student.getSn());
}

@Test
public void testSetAndGetTeacherId() {
    student.setTeacherId(10L);
    Assert.assertEquals(Long.valueOf(10L), student.getTeacherId());
}

@Test
public void testSetAndGetAdmin() {
    Admin admin = new Admin().setId(1L).setLoginName("admin01").
setPassword("password");
    student.setAdmin(admin);
    Assert.assertNotNull(student.getAdmin());
    Assert.assertEquals("admin01", student.getAdmin().getLoginName());
}

@Test
public void testPkVal() {
    student.setId(1L);
    Assert.assertEquals(Long.valueOf(1L), student.pkVal());
}

@Test
public void testEqualsAndHashCodeWithDifferentIds() {
    Student student1 = new Student().setId(1L);
    Student student2 = new Student().setId(2L);

    // 断言不同 ID 的对象不相等
    Assert.assertFalse(student1.equals(student2));
    Assert.assertNotEquals(student1.hashCode(), student2.hashCode());
}

@Test
public void testEqualsAndHashCodeWithSameIds() {
    Student student1 = new Student().setId(1L);
    Student student2 = new Student().setId(1L);

    // 断言相同 ID 的对象相等
    Assert.assertTrue(student1.equals(student2));
    Assert.assertEquals(student1.hashCode(), student2.hashCode());
}

```

```

    }

    @Test
    public void testChainSetters() {
        student.setId(1L)
            .setName("John Doe")
            .setSn("S12345")
            .setTeacherId(10L);

        Assert.assertEquals(Long.valueOf(1L), student.getId());
        Assert.assertEquals("John Doe", student.getName());
        Assert.assertEquals("S12345", student.getSn());
        Assert.assertEquals(Long.valueOf(10L), student.getTeacherId());
    }

    @Test
    public void testSetNullValues() {
        student.setName(null);
        Assert.assertNull(student.getName());

        student.setSn(null);
        Assert.assertNull(student.getSn());

        student.setTeacherId(null);
        Assert.assertNull(student.getTeacherId());

        student.setAdmin(null);
        Assert.assertNull(student.getAdmin());
    }

    @Test
    public void testSetInvalidValues() {
        student.setTeacherId(-1L);
        Assert.assertEquals(Long.valueOf(-1L), student.getTeacherId());
    }
}

```

对 User 类的测试:

```

public class UserTest {
    private User user;

    @Before
    public void setUp() {
        // 初始化 User 实例
    }
}

```

```

        user = new User();
    }

    @Test
    public void testSetAndGetSn() {
        user.setSn(1001);
        Assert.assertEquals(Integer.valueOf(1001), user.getSn());
    }

    @Test
    public void testSetAndGetName() {
        user.setName("John Doe");
        Assert.assertEquals("John Doe", user.getName());
    }

    @Test
    public void testSetAndGetType() {
        user.setUserType("Admin");
        Assert.assertEquals("Admin", user.getUserType());
    }

    @Test
    public void testSetAndGetBedId() {
        user.setBedId(10);
        Assert.assertEquals(Integer.valueOf(10), user.getBedId());
    }

    @Test
    public void testSetAndGetUserRight() {
        user.setUserRight("Full Access");
        Assert.assertEquals("Full Access", user.getUserRight());
    }

    @Test
    public void testSetAndGetPhone() {
        user.setPhone("1234567890");
        Assert.assertEquals("1234567890", user.getPhone());
    }

    @Test
    public void testSetAndGetSex() {
        user.setSex("Male");
        Assert.assertEquals("Male", user.getSex());
    }

```

```

@Test
public void testSetAndGetInSchool() {
    user.setInSchool(1);
    Assert.assertEquals(Integer.valueOf(1), user.getInSchool());
}

@Test
public void testSetAndGetPassword() {
    user.setPassword("password123");
    Assert.assertEquals("password123", user.getPassword());
}

@Test
public void testSetAndGetProfession() {
    user.setProfession("Software Engineer");
    Assert.assertEquals("Software Engineer", user.getProfession());
}

@Test
public void testSetAndGetBuilding() {
    user.setBuilding("Building A");
    Assert.assertEquals("Building A", user.getBuilding());
}

@Test
public void testPkVal() {
    user.setSn(1001);
    Assert.assertEquals(Integer.valueOf(1001), user.pkVal());
}

@Test
public void testEqualsAndHashCodeWithDifferentSn() {
    User user1 = new User().setSn(1001);
    User user2 = new User().setSn(1002);

    // 断言不同 sn 的对象不相等
    Assert.assertFalse(user1.equals(user2));
    Assert.assertNotEquals(user1.hashCode(), user2.hashCode());
}

@Test
public void testEqualsAndHashCodeWithSameSn() {
    User user1 = new User().setSn(1001);

```



```

        User user2 = new User().setSn(1001);

        // 断言相同 sn 的对象相等
        Assert.assertTrue(user1.equals(user2));
        Assert.assertEquals(user1.hashCode(), user2.hashCode());
    }

    @Test
    public void testChainSetters() {
        user.setSn(1001)
            .setName("John Doe")
            .setUsertype("Admin")
            .setBedId(10)
            .setUserright("Full Access")
            .setPhone("1234567890")
            .setSex("Male")
            .setInSchool(1)
            .setPassword("password123")
            .setProfession("Software Engineer")
            .setBuilding("Building A");

        Assert.assertEquals(Integer.valueOf(1001), user.getSn());
        Assert.assertEquals("John Doe", user.getName());
        Assert.assertEquals("Admin", user.getUsertype());
        Assert.assertEquals(Integer.valueOf(10), user.getBedId());
        Assert.assertEquals("Full Access", user.getUserright());
        Assert.assertEquals("1234567890", user.getPhone());
        Assert.assertEquals("Male", user.getSex());
        Assert.assertEquals(Integer.valueOf(1), user.getInSchool());
        Assert.assertEquals("password123", user.getPassword());
        Assert.assertEquals("Software Engineer", user.getProfession());
        Assert.assertEquals("Building A", user.getBuilding());
    }

    @Test
    public void testSetNullValues() {
        user.setName(null);
        Assert.assertNull(user.getName());

        user.setUsertype(null);
        Assert.assertNull(user.getUsertype());

        user.setBedId(null);
        Assert.assertNull(user.getBedId());
    }

```

```

        user.setUserright(null);
        Assert.assertNull(user.getUserright());

        user.setPhone(null);
        Assert.assertNull(user.getPhone());

        user.setSex(null);
        Assert.assertNull(user.getSex());

        user.setInSchool(null);
        Assert.assertNull(user.getInSchool());

        user.setPassword(null);
        Assert.assertNull(user.getPassword());

        user.setProfession(null);
        Assert.assertNull(user.getProfession());

        user.setBuilding(null);
        Assert.assertNull(user.getBuilding());
    }

    @Test
    public void testSetInvalidValues() {
        user.setInSchool(-1); // -1 是无效值
        Assert.assertEquals(Integer.valueOf(-1), user.getInSchool());

        user.setPhone("invalid phone"); // 电话号码格式错误，但不会抛异常
        Assert.assertEquals("invalid phone", user.getPhone());
    }
}

```

覆盖率分析:



五、EclEmma 测试

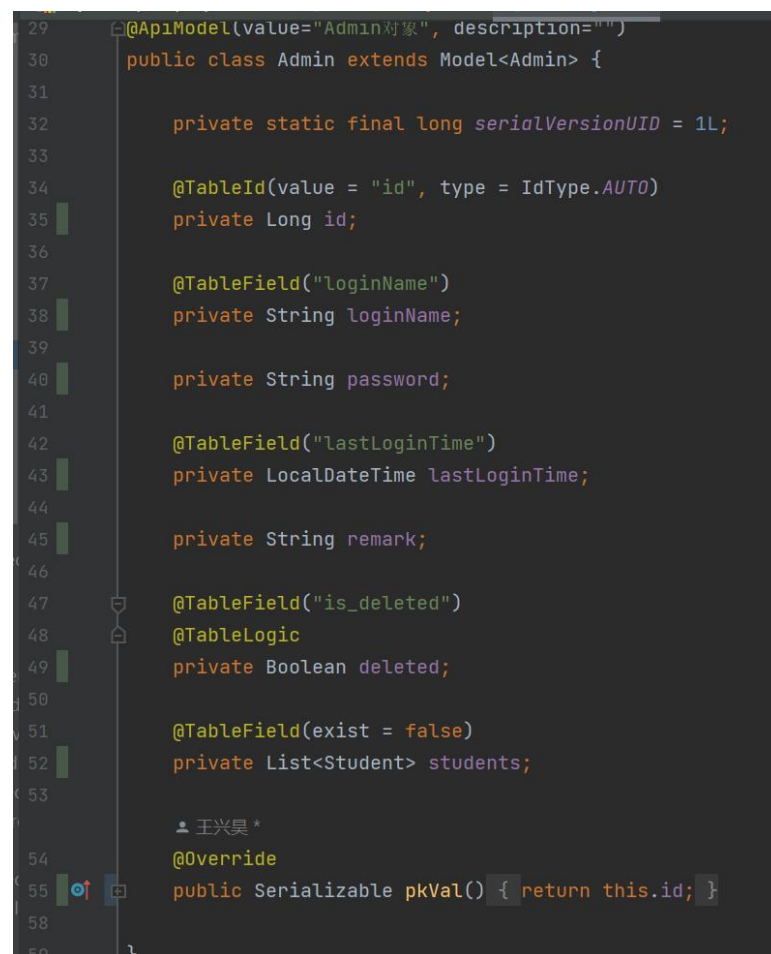
根据上述 JUnit 测试，应用 EclEmma 进行测试。

以下是测试代码的覆盖率分析：



以下是测试代码的具体覆盖情况(左侧方框为绿色即为成功覆盖)：

Admin 类：



Bed 类:

```
21  @Data
22  @EqualsAndHashCode(callSuper = false)
23  @Accessors(chain = true)
24  @ApiModel(value="Bed对象", description="")
25  public class Bed extends Model<Bed> {
26      private static final long serialVersionUID = 1L;
27
28      @TableId("id")
29      private Integer id;
30
31      @TableField("isOwned")
32      private Integer isOwned;
33
34      @TableField("Room_id")
35      private Integer roomId;
36
37      @TableField("isDeleted")
38      private Integer isDeleted;
39
40      @TableField(exist = false)
41      private Room room;
42
43      @TableField(exist = false)
44      private User user;
45
46      * 王兴昊 *
47      @Override
48      public Serializable pkVal() { return this.id; }
```

Building 类:

```
22  @Data
23  @EqualsAndHashCode(callSuper = false)
24  @Accessors(chain = true)
25  @ApiModel(value="Building对象", description="")
26  public class Building extends Model<Building> {
27      private static final long serialVersionUID = 1L;
28
29      @TableId("name")
30      private String name;
31
32      @TableField("Floornumber")
33      private Integer Floornumber;
34
35      @TableField("Roomtype")
36      private String Roomtype;
37
38      @TableField("Roomcount")
39      private Integer Roomcount;
40
41      @TableField("Bedcount")
42      private Integer Bedcount;
43
44      @TableField("Park_name")
45      private String parkName;
46
47      @TableField("isDeleted")
48      @TableLogic
49      private Boolean isDeleted;
50      @TableField("gender")
51      private String gender;
52      @TableField("phone_number")
53      private String phoneNumber;
54      @TableField("property_management")
55      private String propertyManagement;
56      @TableField("building_manager")
57      private String buildingManager;
58      * wuhoulinyin *
59      @Override
60      public Serializable pkVal() { return this.name; }
```

Park 类:

```
22  *
23  * @Data
24  * @EqualsAndHashCode(callSuper = false)
25  * @Accessors(chain = true)
26  * @ApiModel(value="Park对象", description="")
27  *
28  * public class Park extends Model<Park> {
29  *
30  *     private static final long serialVersionUID = 1L;
31  *
32  *     @TableId("name")
33  *     private String name;
34  *
35  *     @TableField("Campus")
36  *     private String Campus;
37  *
38  *     @TableField("Buildingcount")
39  *     private Integer Buildingcount;
40  *
41  *     @TableField("Roomcount")
42  *     private Integer Roomcount;
43  *
44  *     @TableField("Bedcount")
45  *     private Integer Bedcount;
46  *
47  *     @TableField("isDeleted")
48  *     @TableLogic
49  *     private Boolean isDeleted;
50  *
51  *     * wuhoulinyin *
52  *     @Override
53  *     public Serializable pkVal() { return this.name; }
54  *
55  * }
```

Room 类:

```
22  *
23  * @Data
24  * @EqualsAndHashCode(callSuper = false)
25  * @Accessors(chain = true)
26  * @ApiModel(value="Room对象", description="")
27  *
28  * public class Room extends Model<Room> {
29  *
30  *     private static final long serialVersionUID = 1L;
31  *
32  *     @TableId("id")
33  *     private Integer id;
34  *
35  *     @TableField("Floor")
36  *     private Integer Floor;
37  *
38  *     @TableField("Bedcount")
39  *     private Integer Bedcount;
40  *
41  *     @TableField("isEnabled")
42  *     private Boolean isEnabled;
43  *
44  *     @TableField("Price")
45  *     private Integer Price;
46  *
47  *     @TableField("Building_name")
48  *     private String buildingName;
49  *
50  *     @TableField("Park_name")
51  *     private String parkName;
52  *
53  *     @TableField("isDeleted")
54  *     @TableLogic
55  *     private Boolean isDeleted;
56  *
57  *     * wuhoulinyin *
58  *     @Override
59  *     public Serializable pkVal() { return this.id; }
60  *
61  * }
```

Student 类:

```
22  @Data
23  @EqualsAndHashCode(callSuper = false)
24  @Accessors(chain = true)
25  @ApiModel(value="Student对象", description="")
26  public class Student extends Model<Student> {
27
28      private static final long serialVersionUID = 1L;
29
30      @TableId(value = "id", type = IdType.AUTO)
31      private Long id;
32
33      private String name;
34
35      private String sn;
36
37      private Long teacherId;
38
39      @TableField(exist = false)
40      private Admin admin;
41
42      @Override
43      public Serializable pkVal() { return this.id; }
44
45  }
```

User 类:

```
21  @Data
22  @EqualsAndHashCode(callSuper = false)
23  @Accessors(chain = true)
24  @ApiModel(value="User对象", description="")
25  public class User extends Model<User> {
26
27      @TableId("sn")
28      private Integer sn;
29
30      private String name;
31
32      @TableField("UserType")
33      private String UserType;
34
35      @TableField("Bed_id")
36      private Integer bedId;
37
38      @TableField("UserRight")
39      private String UserRight;
40
41      @TableField("Phone")
42      private String Phone;
43
44      @TableField("Sex")
45      private String Sex;
46
47      @TableField("inSchool")
48      private Integer inSchool;
49
50      @TableField("Password")
51      private String Password;
52
53      @TableField("Profession")
54      private String Profession;
55      @TableField("building")
56      private String building;
57
58      @Override
59      public Serializable pkVal() { return this.sn; }
60
61  }
```

六、BUG 报告

报告一

标题	系统登录功能失效		
编号	A1		
产品名称	房源管理系统	模块名称	用户登录
报告者		报告日期	
发布版本号		内部版本号	
配置	Win11 , Edge 浏览器		
可复现性	是	严重级别	高
优先级	高	报告类型	编码错误
对用户影响	高		
问题概述	只要输入了合法的账号和密码，无论两者是否匹配，均能成功登录并进入系统主界面。		
关键词	用户登录		
Bug 描述	1. 用户进入登陆界面 2. 用户输入不匹配的账号与密码 3. 成功登录，进入系统主界面		
预期结果	登录失败		
实际结果	登陆成功		
建议解决方式	检查登录功能对应代码是否被正常使用，检查是否与数据库进行连接。		
指派给		状态	关闭
Bug 解决描述	已解决。检查并修改了对应代码，使之能对输入的账号和密码进行匹配。		
解决版本	20241126a	解决人员	
解决日期		解决后测试人员	
变更历史			

Bug 关闭描述	问题成功解决
----------	--------

报告二

标题	更改用户权限功能失效		
编号	A2		
产品名称	房源管理系统	模块名称	用户管理
报告者		报告日期	
发布版本号		内部版本号	
配置	Win11 , Edge 浏览器		
可复现性	是	严重级别	高
优先级	高	报告类型	编码错误
对用户影响	高		
问题概述	管理员更改用户权限，输入对应信息后并更新系统后，对应用户权限没有发生变化。		
关键词	用户管理，权限更改		
Bug 描述	1. 管理员点击“添加权限” 2. 输入要更改权限的用户 id 3. 点击要更改的权限内容并点击确认 4. 系统更新，对应用户权限没有改变		
预期结果	用户权限发生改变		
实际结果	用户权限未发生改变		
建议解决方式	查询权限更改对应代码的逻辑是否正确，检查该部分代码是否正常运行		
指派给		状态	关闭
Bug 解决描述	已解决，检查并修改了对应代码，使之能保证用户权限得以更改		
解决版本		解决人员	
解决日期		解决后测试人员	
变更历史			

Bug 关闭描述	问题成功解决
----------	--------

报告三

标题	床位分配功能失效		
编号	B1		
产品名称	房源管理系统	模块名称	床位管理
报告者		报告日期	
发布版本号		内部版本号	
配置	Win11，Edge 浏览器		
可复现性	是	严重级别	高
优先级	高	报告类型	编码错误
对用户影响	高		
问题概述	对用户进行分配床位操作时，用户并没有得到该床位		
关键词	床位管理，分配		
Bug 描述	管理员给用户分配床位，结束后数据库显示床位的状态改为被占有，但用户关联床位的属性依旧为空		
预期结果	床位分配成功，用户关联床位的属性为床位 id		
实际结果	床位分配成功，用户关联床位的属性为空		
建议解决方式	检查相应代码，查看代码逻辑是否出错，并检查是否漏写更改用户关联床位的属性这一功能。		
指派给		状态	关闭
Bug 解决描述	已解决。检查并修改了对应代码，使分配床位后用户表关联床位表的属性能正常更改		
解决版本		解决人员	
解决日期		解决后测试人员	
变更历史			
Bug 关	问题成功解决		

闭描述	
-----	--

报告四

标题	查询房源功能失效		
编号	B2		
产品名称	房源管理系统	模块名称	房源管理
报告者		报告日期	
发布版本号		内部版本号	
配置	Win11 , Edge 浏览器		
可复现性	是	严重级别	高
优先级	高	报告类型	编码错误
对用户影响	高		
问题概述	筛选了园区、楼栋、楼层后查询房源，查询结果仍然为查询前显示的结果		
关键词	房源管理，查询		
Bug 描述	用户对房源进行针对性查询，筛选了园区、楼栋、楼层后点击查询按钮进行查询，系统无响应		
预期结果	显示筛选后的房源信息		
实际结果	系统无响应		
建议解决方式	查看筛选功能是否生效		
指派给		状态	关闭
Bug 解决描述	已解决。检查并修改了对应代码，使之能正常进行筛选工作。		
解决版本		解决人员	
解决日期		解决后测试人员	
变更历史			
Bug 关闭描述	问题成功解决		

七、总结报告

系统测试报告																													
测试轮次	第一轮	测试时间	20241125-20241129																										
测试地点	宿舍	测试环境	Windows11, Maven																										
负责人		测试人员																											
风险																													
由于前期进度拖沓，导致后期的测试时间较少，可能有很多缺陷没有被发现。																													
职责																													
测试经理：																													
测试分析员：																													
测试设计师：																													
测试员：																													
测试情况																													
名称	测试类型	预算时间	实际花费时间	本轮发现缺陷数	目前总缺陷数																								
用户登录	人工	1d/人	1d/人	1	1																								
权限管理	人工	1d/人	1d/人	1	1																								
床位管理	人工	1d/人	1d/人	1	1																								
房源管理	人工	1d/人	1d/人	1	1																								
项目缺陷度量																													
<div>Defect Statistics</div> <table><thead><tr><th>Date</th><th>Submitted Defects</th><th>Repaired Defects</th></tr></thead><tbody><tr><td>11.24</td><td>0.0</td><td>0.0</td></tr><tr><td>11.25</td><td>1.0</td><td>0.0</td></tr><tr><td>11.26</td><td>1.0</td><td>1.0</td></tr><tr><td>11.27</td><td>1.0</td><td>1.0</td></tr><tr><td>11.28</td><td>1.0</td><td>1.0</td></tr><tr><td>11.29</td><td>0.0</td><td>1.0</td></tr><tr><td>11.30</td><td>0.0</td><td>0.0</td></tr></tbody></table>						Date	Submitted Defects	Repaired Defects	11.24	0.0	0.0	11.25	1.0	0.0	11.26	1.0	1.0	11.27	1.0	1.0	11.28	1.0	1.0	11.29	0.0	1.0	11.30	0.0	0.0
Date	Submitted Defects	Repaired Defects																											
11.24	0.0	0.0																											
11.25	1.0	0.0																											
11.26	1.0	1.0																											
11.27	1.0	1.0																											
11.28	1.0	1.0																											
11.29	0.0	1.0																											
11.30	0.0	0.0																											
确认延期或不修复的缺陷																													
本轮测试中发现的所有缺陷都将按期修复，没有延期或者不修复的缺陷																													

八、附件说明

由于被测试的房源管理系统所涉及的代码量比较大，故不在 word 文档内进行展示。我们会把源代码、测试报告编写过程中所使用的测试文件以电子版的形式提交。