1. 绪论

本文先是讨论了测试的必要性以及自动化测试在软件开发中的重要性，再而介绍了BDD（行为驱动开发）的原理与支持BDD的自动化测试Cucumber框架，并着重介绍了基于Cucumber测试用例的全平台自动化测试web系统的设计。

软件测试是对项目给定需求的功能验证，是软件质量管理Quality Assurance（品质保证）的重要手段。1979年，Glenford Myers在《The Art of Software Testing》对软件测试作出了定义：“软件测试是为了发现错误而执行程序或系统的过程”。而基于TDD（测试驱动开发）的思想，本文对软件测试有了新的定义：软件测试是通过执行程序，发现系统漏洞与bug来进行对产品质量的验证，软件测试能修正软件的错误，或在测试中发现产品的非错误性不足以驱动开发改进。

软件测试具有发现产品质量问题并改善问题的功能，而在实际开发中，软件测试是不可或缺的，软件测试在当前对软件应用的高安全性、可用性、高质量性的需求当中，扮演着愈发重要的角色。

1.1研究背景与意义

随着应用软件的发展、测试难度的需求复杂化，自动化测试的理念逐渐被大多数企业所接受。并且在软件过程管理中，随着迭代式开发与敏捷开发等框架取代瀑布模型，测试的周期缩短，工作增加，需要对软件的多个迭代版本进行测试，并增加了许多重复性的测试劳动，自动化测试的优点是显而易见的。

软件测试主要由四部分组成：被测应用、环境、方式、测试人员。

完善的自动化流程可以让环境部署的时间大大减少，以自动执行一些重复但是必要的任务的方式减轻测试人员的负担，并且解决许多手工测试完成不了的工作（如并发访问等），所以标准的自动化测试软件服务对项目持续交付和持续测试意义深远，同时对于项目的质量检测、监测都是不可或缺的。

然而，自动化测试并不是万能的，自动化测试在相当一部分的环境下，具有极大的局限性。首先，自动化测试难以管理，在对系统进行全面的自动化测试当中，需要将许多的测试脚本综合管理。并且，自动化测试的门槛相较于手工测试较高，它要求自动化测试开发人员具有对测试环境进行搭建、并编写自动化测试脚本的编程能力。

所以，对自动化测试的研究以及如何应用在企业项目开发中的软件过程管理中的探讨势在必行，而使用SaaS软件服务提供BDD风格的自动化测试管理平台，将为自动化测试提供一个标准、严格、合乎软件过程管理规范的管理与执行工具。

1.2国内外研究现状

据Oppenheimer Funds 的调查，在2001年前后三年中，全球范围内由于采用了测试自动化手段得到的ROI（投资回报率）高达1500%。随着对测试的要求越来越高以及软件测试技术的发展，实现软件测试自动化的趋势已经不可逆转了。

国外对敏捷过程管理中测试驱动开发以及能够理解用普通语言描述的测试用例的支持BDD的Cucumber的理论研究、实践，侧面证明了自动化测试对于软件开发的质量管理的重要性。而在国内自动化测试属于较为新鲜的技术，在研究方面稍为落后，并且缺少丰富的应用经验。所以自动化测试在国内许多公司处于一个尴尬的境地：大部分公司依旧重度依赖手工测试，或想开展自动化测试却发现并没有收获预想中的效果。

在企业实施自动化过程中，绝大部分中国企业用户选择在企业内部实现测试自动化，选择在企业中开展自己的自动化测试服务，并且成立自己的自动化测试部门来组建自己的自动化测试系统工作。而在国外，由于国外SaaS软件的盛行，国外的企业（尤其是中小型企业）往往是通过选择由专门企业提供的测试自动化平台托管服务，并对此服务付费，或是通过外包、派遣、离岸等多种方式相结合来实现对自动化测试的需求。国外的企业关注测试带来的结果、发现的问题，而不是由如TestingWhiz等专业的提供测试服务的公司提供的测试的技术、过程。这相当一部分原因是因为国内外的软件技术发展趋势的不同与SaaS软件并没有在国内得到很好的发展。

1.3 研究内容和论文结构

（1）本文致力于对

第二章 软件测试概论

2.1 软件测试的概念与目的

软件测试是在有限的时间中保证软件高质量的重要手段，是软件开发过程中相当重要的一部分。测试人员通过特定的手段，针对软件的功能或模块，按照一定的流程来对软件进行执行并找出其中的错误，这是软件测试的基本方式。

IEEE中给出了软件测试的定义：软件测试是使用人工或自动手段来测试某个软件的过程，它的目的在于检查软件是否符合规定并满足需求，或者是对比并找出被测软件的预期行为与实际行为之间的差别。

可以看出，软件测试的目的主要是衡量系统是否符合预期并满足用户的需求。换言之，如果软件系统与用户需求不符，则可判定其中存在缺陷，因为用户需求并不能以系统意志所转移，所以系统将需要得到修正。但是，软件测试的目的并非是追求完美切100%无错的系统，在测试人力、资源上的投入需要满足供需关系，当系统已经趋近无错时，发现漏洞与问题已经愈发困难；而投入过多精力在测试上，则有可能导致不必要的成本增长。

软件测试的通常方式是设计测试用例，并比较实际行为或输出结果与预期行为或预期输出结果之间的差异，且此差异是否在用户可以接受的范围之内。在测试执行的阶段，需要注重测试过程中的缺陷，对用例的执行情况进行评估，以求为发现系统的边界值等极端错误进行合理的测试，来设计优秀的测试用例。

2.2自动化测试

自动化测试是软件测试中的一个有别于手工测试存在的概念，通过相应的测试工具或测试脚本来运行大量测试用例以完成测试计划。它能够完成许多手工测试难以做到的测试方法，并且能够自动地生成测试报告与结果，提示错误信息，从而节省了大量的人力资源与时间。顾名思义，自动化测试是通过机器取代人工，以拟定的方式运行程序，并对程序的结果进行比对以达到测试的目的，从而提高软件质量与可靠性的测试方法。

自动化测试相对于手工测试有许多的优点，使得自动化测试在现代软件开发中拥有至关重要的地位：

1. 自动化测试可以减少重复性的测试工作，节约测试成本。在项目的版本迭代中，或是在项目新功能的添加中，由于代码存在着一定的耦合性，代码之间有可能在一定程度上存在着互相的影响。例如在开发中，由于数据库设计时没有全面的考虑，导致在新功能增填时需要修改数据库的字段，从而导致旧代码可能存在着一些问题。所以，在版本的迭代中，测试人员常常需要对系统进行多次的回归测试（修改了旧代码后，对系统进行重新测试以确认此修改没有引入问题或带来其他的影响），而在大型系统中，完整的软件生命周期中回归测试更是不可少的一部分，在软件开发的各个阶段，都需要多次的回归测试。在整个软件测试的过程中回归测试占了极大的比重，而回归测试在一定程度上可以说是大量的重复性的劳动，它给手工测试带来了许多困难，极大地增加了手工测试的成本与时间的浪费，尤其是在渐进开发与开速迭代开发中，回归测试更加频繁，在极限编程中，一天甚至有可能要进行好几次回归测试。这使得手工测试的缺陷被放大，而自动化测试则能很好地适用于此场景。由于自动化测试脚本的可维护性、可重用性，测试开发人员不需要重新开发测试用例对应的测试脚本，只需要让测试脚本再次运行即可。这样节约了测试成本与时间，缩短项目开发周期。
2. 在相当一部分情况下，自动化测试能够完成许多手工测试无法完成的测试。例如在对网站、App的负载均衡测试中，人工无法模拟大量的（例如百万级别）访问、使用数据，而在自动化测试中可以通过计算机的执行来模拟出大量的访问，可以对软件进行性能测试或者说压力测试。许多测试工具如LoadRunner即是在程序内部封装了访问的代码，并通过配置决定需要的访问量，来做对软件的性能测试与性能分析。而在多线程并发中，bug往往难以复现，而手工地测试更是基本不可能找出错误，只有自动化测试可以模拟微秒级别的并发问题。所以，自动化测试是对高性能需求项目开发中不可缺少的一部分。这也是自动化测试开发的主要意义所在。
3. 自动化测试能够提高软件质量的可信度。手工测试在人力完成的情况下不足以对软件质量进行十分严谨的保证，而自动化测试在单元测试中，可以提供高覆盖率、高精度的保证。在自动化测试用例下，由于脚本由机器执行，所以减少了执行过程中的错误与疏漏问题，而将问题关注到测试用例的设计质量中。特别当在复杂的计算中，自动化测试具有手工测试无法比拟的优势。
4. 自动化测试具有很高的复用性、一致性、可重复性。自动化测试用例往往采用脚本技术，而在较多的不同环境下，所需要编写的测试脚本都是类似的，这使得测试脚本具有很好的可复用性，从而使得编写自动化测试脚本所需时间成本降低。而脚本的执行是由计算机来完成，脚本的输入一致的情况下，输出在没有随机的情况下也是一致的，这很好地保证了测试用例的一致性。而脚本代码往往可以保存下来再次使用，可以提供高频率地重复运行来验证软件的可靠性，同时节省了重复执行同一个测试用例的时间，将测试人员的时间集中在新的特征功能中。

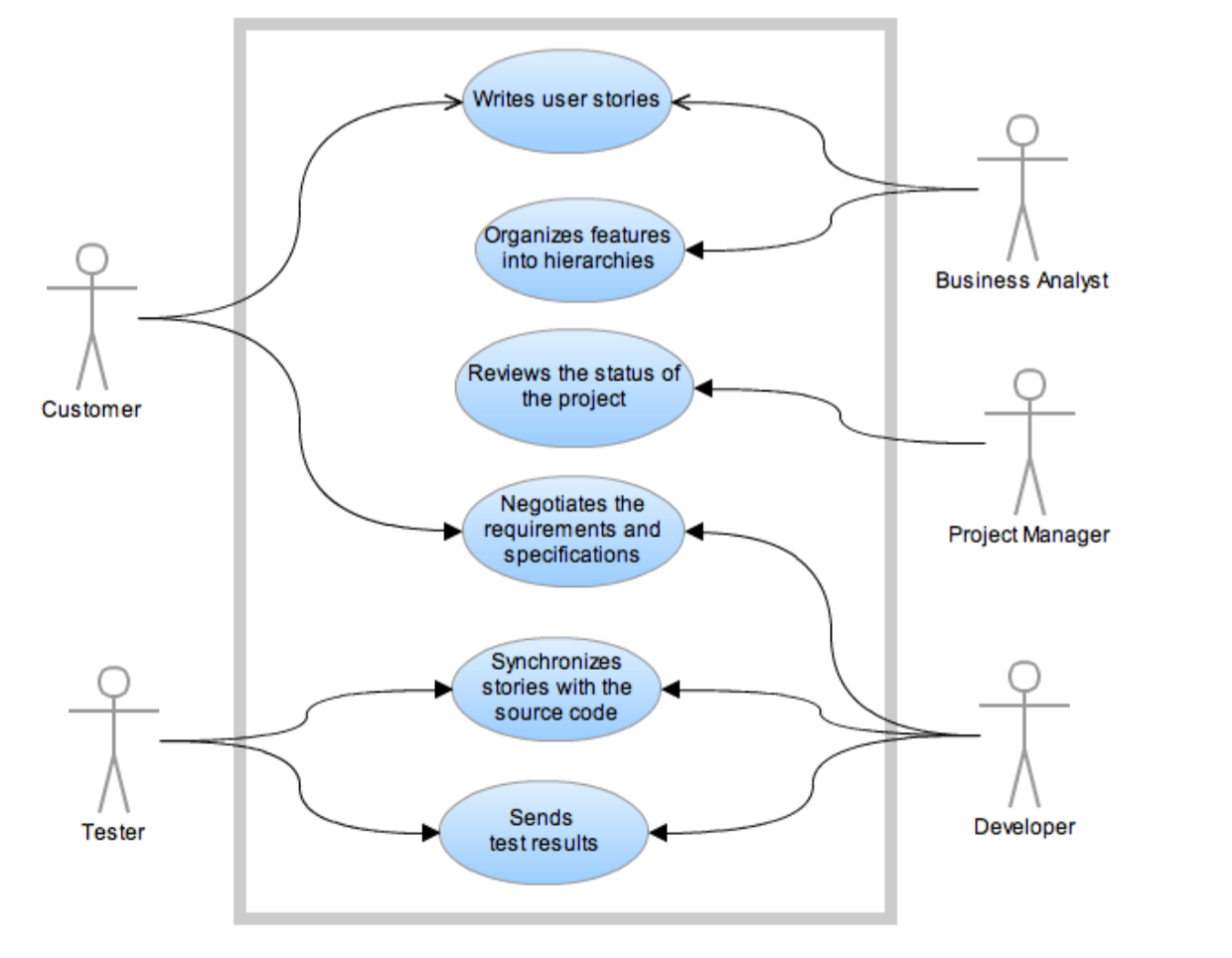
然而自动化测试仍然不能完全取代手工测试。在许多主观性的场景，手工测试还是必不可少的。例如判断web页面中Css样式是否正常、合理，是自动化测试难以完成的。而且手工测试人员在测试过程中可以发现许多测试用例之外的错误或问题，以编写更加完善、有意义的测试用例，这对测试用例为核心的测试过程是十分有意义的。而且，编写测试用例需要测试开发人员具有较好的编程水平，而在某些复杂场景中编写自动化测试脚本往往会花费许多时间，而是否使用自动化测试需要妥善的衡量利弊。

综上所述，自动化测试在提高项目质量与可靠性中具有许多优点，但是自动化测试只是手工测试的补充，它不能取代手工测试，在项目中如何使用自动化测试和手工测试，需要依据具体情况来定义。

2.3BDD与软件测试

BDD（行为驱动开发）是在TDD（测试驱动开发）理念基础上发展的一种敏捷开发方式，它鼓励开发工程师、测试工程师与客户（需求提供者）之间的协作，包括验收测试与客户测试驱动等极限编程的实践，作为对测试驱动开发的回应。2009年，Dan North对行为驱动开发给出了定义：行为驱动开发是由外而内的、需要多方协同合作的、可拓展的、高自动化的第二代敏捷方法。2010年，Chelimsky将BDD具体描述为：行为驱动开发是通过从利益相关者（如客户）的角度描述其行为来实现应用程序的开发方法。

BDD的开发一定程度上借鉴了TDD的思想。TDD是指在编写功能性代码之前，首先选择先编写测试用例，根据测试的需求推动开发的需求。它很好地诠释了“代码即文档”的理念。但是在TDD开发中，由于测试代码往往是程序代码的两倍，并且在开发人员在进行单元测试时，往往是对代码的结构单元，如类或方法进行测试，然而在编写测试时，开发人员往往对代码结构只有一个模糊的想法。此外，测试以来代码结构，使得代码难以重构。

2005年，Astels批评了TDD的不足，并提出了使用行为与规范代替测试，关注软件做什么而不是验证软件是否正确-----即行为驱动开发。行为驱动开发方法提供了一种描述预期行为的方式------使用通用的自然语言描述系统行为，使得项目开发人员与项目的测试人员、需求提供者与产品经理对项目有一致的了解，并允许QA团队从利益相关者的角度关注系统。BDD强烈地关注了客户与客户需求，力求开发出的产品在最大程度上与需求提供者描述的预期相符，从而提高项目质量。 在理想情况下，测试是能够自我描述的。一个好的测试用例应当明确地描述自己要做什么、怎么做，所以，基于BDD开发自动化测试用例，可以让代码本身作为一个可读的自然语言，它定义了用于描述、构造的规范，与测试的理想情况下契合。基于BDD的自动化测试开发不仅在理论上得到了研究，在生产环境中也得到了应用。如2010年开发的RSpec框架，以BDD的风格用Ruby语言编写代码规范，利用了打开现有类并将断言直接并入指定代码（根据Astels给出的建议）的可能性；再如本文中描述的Cucumber框架，在生产中得到了较多的应用。

与其他自动化测试方法相比，基于BDD的自动化测试具有许多优点：

1.基于BDD开发的自动化测试用例有助于生成软件文档。这种方法还可以更好地让开发人员了解测试的目的与目标应该是什么，只需要简单地规划生成文档的规范。它在块（方法或类等）的结构上优化了代码的结构，体现了“代码即文档”的思想。

2.在行为驱动开发风格的自动化测试中，每个期望的行为被定义为具有有关与它的详细信息的描述块。用文本而不是方法名描述此块的选项，为开发人员提供了在将要指定内容的更多自由，使得开发人员能够更好地理解块中的目的，在接口开发中，对输入输出有明确的定义。

3.基于BDD的自动化测试开发中，行为常常被描述为需求并在代码中体现。BDD为每个自动化测试用例提供了相应的场景描述，这不仅提高了代码的可读性，为代码的可维护性、可重构性提供很更好的理解性支持。

4.在BDD风格中，自动化测试数据将便于维护。由于在BDD开发风格中，自动化测试用例与代码将有更紧密的联系，所以在数据的给出种，往往将定义的数据描述在BDD自然语言描述的行为之中，这样分离了数据与代码的耦合，将利于自动化测试数据的维护。

5. BDD方法由于加强团队中各人员的联系，所以在链接自动化测试与手工测试人员的合作中具有极大的帮助。BDD方法在编写行为中，要求行为描述能够被项目中所有人理解，所以自动化测试脚本在提供定义的输入中，可以由手工测试人员负责自动化测试用例的执行。这样不仅可以省却许多测试成本，让测试开发人员专注于测试脚本的开发，更充分地提高了手工测试人员的效率。

第三章 项目技术概述

3.1前端技术概述

项目前端使用的是单页应用（single page web application），即在前端中只存在一个html页面，在实现其他与后台交互的过程中完全使用AJAX以实现对前端。前端使用的技术主要有以下几种：

1. React

React的官方网站中声明：React是一个用于构建用户界面的声明性，高效和灵活的JavaScript库。

React由FaceBook开发，起初是因为该公司对市场中的javascript MVC库不甚满意，所以开发了React框架用于架设Instagram网站，并在2013年开源。React让创建交互式UI更加方便，对代码的复用率高，并且性能出众，是一个新兴但是得到了大量应用的流行框架。React的生态圈相当丰富，在npm（前端开源包管理工具/社区）有许多基于React的开源组件，它们在项目中也得到了相当多地应用，在节省编写大量原生dom的时间的同时提升了页面的美观度。

React框架设计的核心前提是认为UI只是将数据通过映射关系转换成其他形式的数据，它用虚拟DOM的思想对数据进行传递，并使用了“状态”（state）来针对具体的渲染目标。用虚拟DOM代替DOM进行DOM操作，大大地改善了在复杂DOM操作时的短板------性能问题。React采用树形式存储虚拟DOM，每当数据变化则重新建立新的虚拟DOM树，并与当前DOM树与之前DOM树进行对比，并只对之间的区别进行真实DOM在浏览器中渲染。React推荐使用组件的方式定义UI的结构，认为UI应该由组件构成。组件树的结构，使得React的组件拥有可组合性、可重用性与可维护性。React还拥有许多开发人员友好的特性，如JSX的语法，能够在组件的模板中将HTML与JavaScript混写，大大地增加了代码的直观性。

React高性能的原因一大部分是解决了DIFF算法低效性的问题。传统计算树形结构转换的最少步骤，复杂度达到O（n^3）。而React采用了对树分层比较、不考虑跨层级移动（在Web UI中DOM节点跨层级操作非常少，且在React中不被支持）、使用key标识相同位置对应关系的方法，与对相同位置节点若虚拟DOM一样则递归向下比较、不同则替换的方法，提高了React渲染的性能，并使以上巧妙的方法结合实际情形，使复杂度降至O（n）。

项目中采用React作为前端JavaScript作为前端JavaScrpit框架，改善了项目前端性能，并使得项目中重复模块得到了很好的代码复用。

1. Redux

Redux是JavaScript的框架，提供状态容器，并提供可预测的状态管理。

Redux的思想是将UI中所有的状态（state）以对象树的形式存储在store中，并使store中的所有数据只能由action改变，通过带有描述性文字的action，匹配reducer对store中的数据进行改变。Redux的优点在于对state进行全局地管理，它避免了数据不同步的问题，并对数据的改变做了规范的管理，使得state能够存放全局数据，减少了与后端的交互。

在项目中，Redux用于做React 中组件变量的管理。在Redux的store中存放了项目中绝大部分数据，用于对渲染结构的规范管理。项目中将对数据的改变映射为发送action的方法，在Reducer中实现对数据的处理，并分发到各个组件中。在React的组件中，由于分发的属性与数据是通过React的Props传过来的，所以React的组件将会得到更新，从而对UI的真实DOM进行改变。

Redux对数据进行了存储、分发与管理，与React精炼的配合对前端逻辑进行了高效、清楚地实现。

1. Swagger-client

Swagger-client是npm社区中的一个开源工具，它生成一个给定指定请求处理程序和Swagger API规范模式的客户端，用于对后端api的访问。

项目中采用了Swagger-client 作为前端中用于访问后端api的工具，它由后端提供api的访问方式，封装了ajax对后台的访问，实现了前后端数据的传输与交互。

Ajax是Asynchronous JavaScript And XML的缩写，它是建立单页应用中不可缺少的技术。单页技术只从后台请求一次HTML页面，而对与后端交互、与后端数据传输，需要用到Ajax作为联通后端的桥梁。Ajax可以访问指定url，并可以携带数据，并得到返回数据，并对返回数据进行处理或者再次渲染前端页面。它避免了每次都从后台中获取整个页面的麻烦，真正实现了异步数据传输请求，可以动态地改变页面的一小部分以减少数据传输的压力。Ajax的核心思想是通过XMLHTTPRequest来实现对后端的访问与数据交换，避免了页面的重载。它在本项目中对比web socket具有很大的优势-----在许多由用户控制是否访问服务器的情况下，对服务器的访问是偶然地、不频繁地，它不需要维持和后端的通信，节省了服务器资源。

3.2后端技术概述。

后端主要使用了Django实现对请求的处理，并使用了git对个人用户的文件的保存与同步管理。

1.Django

Django是一个基于Python开发的MVC模式web开发开源框架。Django被誉为中小型项目开发效率最高的开源web框架，它具有良好的拓展性，并为开发人员提供了许多安全性的支持。

Python语言是一种开发简单、高效的弱类型语言。它支持面向对象编程，具有极为简单的语法与可读性高的关键字，基于缩进的代码结构十分简洁优美，为开发人员减少了编程的工作量，并降低了代码的臃肿性。同时，由于近年来python生态圈的丰富，python被装备了十分庞大的标准库。大量的Python第三方库使得python几乎是全能型语言：无论是桌面游戏的开发，还是web系统的开发，以及科学计算等方面，都得到了完善的支持。

Django项目采用MTV架构。其中，Model层处理数据库的操作，Model类封装了许多查询数据库的方法（如filter方法），基于这些方法，开发人员可以在一定程度上不关注于SQL语句的实现与查询的方法，在纯粹的类方法操作中对数据库实现增删改查功能。而且由于python中\*\*参数操作符的语法，Model方法对数据库的操作更加简洁、可读性更强。由于Django对数据库的强大处理功能，可以在项目中对Model修改，再使用Django自带的makemigrations、migrate或remigrate等工具对数据库参照model进行直接地修改。而Template用于存放页面处理的模板，与JavaEE中的JSP类似，可以接受并渲染后端数据。在本项目中由于是单页应用，所以Template并未得到较多应用。View层对应MVC模式中的Controller层，用于对请求的分发处理，在Controller中根据请求实现相应的业务逻辑，并对请求作出响应，并可以返回Template中存放页面的模板。Django拥有功能强大的URL分发器：基于正则表达式的URL模板，使得Django的URL分发更灵活、拥有更强的可拓展性。

2.Java Play

作为对测试用例中基于Cucumber for Java的基于Java语言测试用例脚本的解析，要求项目中导入Cucumber的第三方jar包对java语言编写的feature文件进行扫描、解析。而Java 的框架，如spring MVC、structs等框架都显得过于庞大臃肿。于是，Guillaume Bort创建了轻量级框架Java play作为对轻巧好用的Ruby on Rail与Django框架的回应 。

Play框架也是基于MVC架构，其中routes文件以极为简单的方式提供了url与controller的映射。在本项目中，Play框架下的Java服务器提供了一个接口，在项目需要扫描Cucumber框架下的自动化测试用例模板时，通过前端发送请求到Django服务器，再由Djiango服务器请求Play服务器，Play服务器扫描得到大量的数据，发送到Djiango服务器中，再由Django服务器发送到前端作为相应。此处充分利用了Java的性能高于Python的特点，对项目进行了分布式的开发。

1. Git

Git是由Linus Torvalds开发的一款功能强大、开源免费、设计精巧的分布式版本控制系统。它起初是因为Linus在开发linux内核时由于不满SVN的收费条件，而决定开发的基于命令行的（后陆续推出了图形界面方式）在SVN基础上拥有其他多种功能的git。它起初由于晦涩难懂被人抵制，后随着git的开发深入，对其友好性进行了许多提升，逐渐被开源社区接受。它基于commit本地仓库的思想，可以让开发者提交项目代码更新到本地中。在项目中不仅仅使用git作为项目托管工具，更使用了git对用户执行文件进行了控制。

由于Python集成了git的第三方库，项目中使用python的git模块对用户文件进行操作，这避免了shell脚本难以进行有效控制、维护的问题，并能对个人用户下的文件 git版本状态进行检测，在发生错误情况下进行有效的错误处理。

在项目中，使用了git作为用户文件托管工具。在个人用户的账号下，拥有许多的测试执行参数、数据、样本与计划，这些文件是以.feature为后缀的文件，作为测试执行用例参数的编辑结果。在Cucumber框架下，根据已经写好的自动化测试脚本，在此基础上添加.feature文件作为参数配置，可以让测试用例根据用户自定义的数据、顺序被执行。

而个人用户下，每个用户都拥有自己的编辑的测试数据、步骤顺序、自己的参数配置等自定义信息，这些不仅保存在数据库中，也需要被保存在.feature文件中被自动化执行。所以，git不仅可以让每个用户拥有自己的分支，通过 git clone的功能将个人初始配置被克隆到项目独立的仓库中，作为自己的个人“文件夹”被编辑修改，还可以让个人账户下的文件被编辑修改时被保存并通过一系列的git命令提交到远端，让项目个人文件存储在云端，有效地防止了服务器可能出现的问题对用户文件带来影响。

1. REST

REST（Representational State Transfer，表述性状态转移）是一种web服务设计模式。2000年，REST由Roy Fielding在“Architectural Styles and the Design of Network-based Software Architectures”中提出，定义了一组以URI（Uniform Resource Identifie，统一资源标识符）为核心，基于HTTP通信接口的规则，利用这些规则为网络资源提供了良好的设计模式。而满足这种设计模式的设计风格被成为RESTful架构风格。

论述REST之前，首先要明白URI的含义。URI由访问资源的命名机制、资源存放服务器名、资源名组成，它是URL更高级别的抽象，URI定义了资源，而URL定义了访问资源的方式。REST设计是将互联网资源被标识为URI，作为纯粹的资源地址或者资源名称。在这里，资源作为名词，而URI中不含有动词，作为对资源名词的绝对代表。RESTful架构是无状态的，它具有简单、低耦合的特性与优点。

那么，在项目中该如何对资源进行访问、增删改查？其实，在HTTP协议中早已定义好对资源特定的请求方法。在传统非REST风格项目中，项目大多只使用了GET POST请求方式，并且对GET POST的请求方式作出了非HTTP愿意的区分：GET用于直接访问URL，并在URL中携带特定的参数，而POST可以在提高安全性的同时访问URL，在URL中隐藏提交的数据。RESTful架构中，使用了GET、POST、PUT、PATCH、DELETE等方法对项目中资源进行操作。GET代表获取，对URI的访问只负责获取指定资源功能；而PUT表示添加，对指定资源只进行添加一条记录的操作；而PATCH表示修改，对指定URI指定的资源进行携带数据的修改；DELETE则请求删除URI指定的资源。这样的设计模式简单，并且遵循了HTTP请求方法的原语义。服务器中无需对客户端的状态进行保存，只需要对给定URI请求作出响应，降低了项目代码的耦合性。而这些HTTP方法是可拓展的，如上文中的Patch方法就不是HTTP标准方法，这种特性提高了REST架构的灵活性。

RESTful架构还要求项目中具有统一的接口。在客户端与服务器之间的交互中，客户端访问服务器的接口必须是统一的，这意味着，在访问同一资源时，服务器的各个子系统必须提供标准的、一致的、统一的接口，以供客户端调用。这样的设计模式优化了系统接口的风格，在一定程度上减少了各个系统之间通信的压力：由于接口是统一的，在设计、开发中只需要对资源指定URI拥有共识，则在开发中对资源的操作可以变得透明、可猜想、简单。

项目中使用python 的restframework模块。这是python的第三方类库，可以为项目自动生成标准的对model定义的资源的访问以及各种增删改查操作，这节省了开发人员的时间。并且，这些api是可以被重载的，对api的访问依旧可以如同其他web项目中一样实现相应的自定义业务逻辑。

项目中，还使用了Swagger作为前后端通信的契约。通过swagger，项目可以自动生成REST api接口文档，并可以提供访问页面来查看这个接口文档。Swagger还提供了接口的测试功能，在api文档页面，可以通过页面自定制操作来对api进行测试，同时提供详细的文档说明与示例操作。

3.3 自动化测试部分技术

自动化测试部分是使用了Cucumber框架编写的自动化测试用例，基于Selenium与Appium框架执行。

1.Cucumber

Cucumber是一个高级别的，基于BDD的自动化测试框架。它通过自然语言描述了独立的、对用户有价值的、可预测的、可测试的用户故事，使用简单、易懂、符合结构化语言的feature文件作为测试文件，而不是让非开发人员难以理解的可执行代码作为测试文件。这样，非开发人员可以读取测试文件，并理解测试文件、甚至在简单的训练过后编写测试文件。这样的feature文件所用的语言称为Gherkin语言。

Gherkin提供可被Cucumber框架解释道的语言。它可以让你在不关心底层如何实现的情况下描述软件的行为，与Python一样使用缩进来定义结构。在以.feature为后缀的文件中，描述了feature的上下文、行动者、动作与目的。基于.feature文件的Gherkin文件特征如下：

login-with-email.feature

1 Feature: GL-2078

2 用Email登录

3

4 @NotAutoAcceptAlert @P1

5 Scenario: 当登录密码错误时弹出警告

6 Given [Welcome Page] Create Account with invalid account Email:"123 @gmail.com" and Password:"12345678"

7 Then [Alert Dialog] Check If Alert Title "" Content "Password must be a t least 8 characters and include at least 1 letter, 1 number and 1 special c haracter or upper case letter." Button Name "OK" pop up

其中第一、二行描述了Feature的名字与描述，而scenario定义步骤顺序、场景描述。Given与Then是Gherkin的关键字，映入了定义给定场景的上下文的步骤。Scenario由Step构成，每个Step由Given或When、Then等关键字开始，Step（6，7行）用自然语言指定了Scenario中执行的操作。

而为了让Cucumber在执行Scenario时知道在运行到Step时该做什么，Step必须有自己的定义文件，以提供Step代码的底层实现，让Cucumber运行自动化测试用例脚本。所以，Step的定义文件------即Step Definition，需要能够匹配step，并给step装配执行的脚本代码。所以，step可以是字符串或正则表达式，当运行Feature文件时，Cucumber将搜索存在的step definition并为Step找到配对的定义，并执行其对应的方法。在Java测试脚本中，它的形式如下：

@When("^\\[Edit Task Page\\] Update Task Title as \"([^\"]\*)\"$")

public void edit\_task\_Page\_update\_task\_title(String arg){

//执行的自动化测试脚本代码

}

Scenario可以用多个标签标记，以@符号开头。这些标记可以为Scenario执行时提供特征标签，可以给出运行的建议、环境、优先级等配置属性。步骤执行是根据Scenario的上下文执行的，它将Scenario作为一个执行模块，步骤们可以共享实例变量与环境变量、系统状态，在Scenario运行结束时重置。

Cucumber为BDD自动化测试提供了简单的规则语法，并对行为进行了自然语言步骤的解释。它使得开发人员与非开发人员使用相同的语言来描述行为，让手工测试人员可以对自动化脚本进行利用。

2.sele