文件编号：**3107-SWC2018-20180045**

受控状态：■受控　□非受控

保密级别：□公司级 □部门级 ■项目级 □普通级

采纳标准：**CMMI DEV V1.2**

图美集

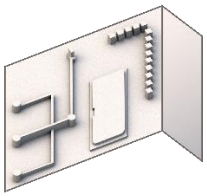
Temage

项目测试文档

Version 1.0.2

2018.11.20

Written by 3107



目录

记录更改历史

序号	更改原因	版本	作者	更改日期	备　注
1	创建	1.0.0	队员 C	2018/11/17	初稿
2	更新	1.0.1	队员 C	2018/11/18	整体修改
3	更新	1.0.2	队员 C	2018/11/19	初赛终稿

1 引言

1.1 编写目的

编写本测试文档的核心意义在于明确本项目的测试目标，为项目的测试建立系统的计划以及分析项目的结构进而设计测试过程以及实现方案。

Temage 项目的测试可分前端部分和后端部分。每部分的测试又可分为单元测试和集成测试，确保前后端分别能够独立正常运行。我们还会进行集成前后端的系统测试，确保系统最终集成之后的正常运行。

1.2 项目背景

Temage 智能图文排版项目是一个运用深度学习技术实现自动图文排版的 web 应用。用户给定文本和图片，即可根据文章的主题，在线智能排版文本与图片，选择合适的风格，生成一篇优美悦目的文章。能够极大地节省文章从排版到发布的时间，提高用户的效率和生产力。

项目针对当下大众资讯获取、媒体信息传播的新变化，具有很好的应用前景和广大的目标用户群体。

Temage 注重用户的体验，旨在为用户提供优美友好的界面和优质流畅的服务。

1.3 术语和缩略语

[1] Vue.js：Vue.js 是一个构建数据驱动的 web 界面的渐进式框架。Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。它不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与单文件组件和 Vue.js 生态系统支持的库结合使用时，Vue.js 也完全能够为复杂的单页应用程序提供驱动。

[2] Karma: A simple tool that allows you to execute JavaScript code in multiple real browsers. 一款前端测试 Javascript 框架。

[3] CI: CI 的全称是 Continuous Integration，表示持续集成。在 CI 环境中，开发人员将会频繁地向主干提交代码。这些新提交的代码在最终合并到主干前，需要经过编译和自动化测试流进行验证。持续集成过程中很重视自动化测试验证结果，以保障所有的提交在合并主线之后没有问题，或对可能出现的一些问题进行预警。

[4] CD: CD 的全称是 Continuous Deployment，表示持续部署。在 CD 环境中，通过自动化的构建、测试和部署循环来快速交付产品。该产品可以进行灰度测试或是直接上线运营。

[5] Mock.js: Mock.js 是一款模拟数据生成器，旨在帮助前端独立于后端进行开发，帮助编写单元测试。提供了根据数据模板生成模拟数据、模拟 Ajax 请求，生成并返回模拟数

据和基于 HTML 模板生成模拟数据。

1.4 参考资料

[1] Karma-npm Available online on https://www.npmjs.com/package/karma

[2] Mockjs online on http://mockjs.com/0.1/

2 测试计划

2.1.1 测试策略与目标

（产品前端框架为 Vue.js，后端框架为 Django 和 Tornado）

测试过程可大致分成两大部分：前端测试以及后端测试。前后端的测试又可细分为单元测试、集成测试。最终我们还会进行集成前后端的系统测试，确保系统最终集成之后的正常运行。

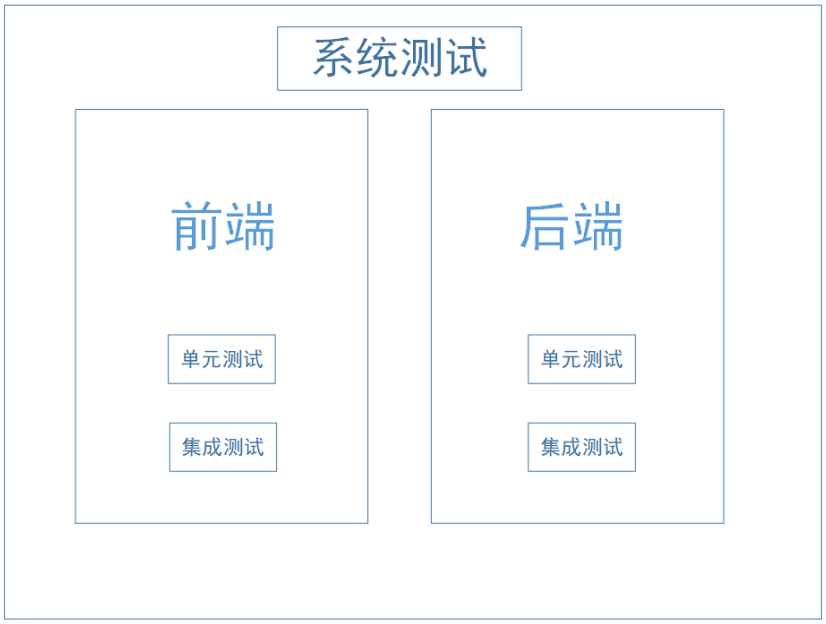


图 2.1 Temage 项目测试架构

前端单元测试主要为编写基本测试用例来检查各项函数的返回值是否正确，此部分测试可以使用测试框架（Karma）进行。鉴于项目的前后端是分离开发的，在前端单元测试中我们也会使用 Mock.js 作为随机数据的生成器，模拟后端的数据进行测试。

前端的集成测试要求将通过单元测试的前端各模块聚合起来，配合 Mock.js 进行前端的集成测试,确保各模块之间能够正常协同运作。我们计划经常性地进行集成测试以确保任何时刻都有一份稳定且版本较新的代码。

后端单元测试的将会使用 Django 和 Tornado 自带的单元测试工具。测试策略与前端的代码相似。

后端集成测试将会使用 Django 的 LiveServerTestCase 进行，确保前后端能够正确协同工

作。

系统测试是检验项目完成度的一大指标，我们决定使用在 Django 的 LiveServerTestCase 中使用 selenium 模拟操作前端，进行前后端集成的系统测试。

项目将会在 CI/CD 架构下进行开发，开发人员在本地的单元测试通过之后，向主代码库提交代码。代码库通过 Webhook 向 Jenkins 服务器推送开发人员的代码变化，Jenkins 服务器获得源码之后自动进行集成测试和系统测试。若集成测试和系统测试通过则会向 Docker Hub 推送新的镜像，同时在服务器上执行远程脚本拉取新的镜像并部署。最后通知软件测试人员部署结果。若集成测试出现问题则同样会提示软件测试人员。

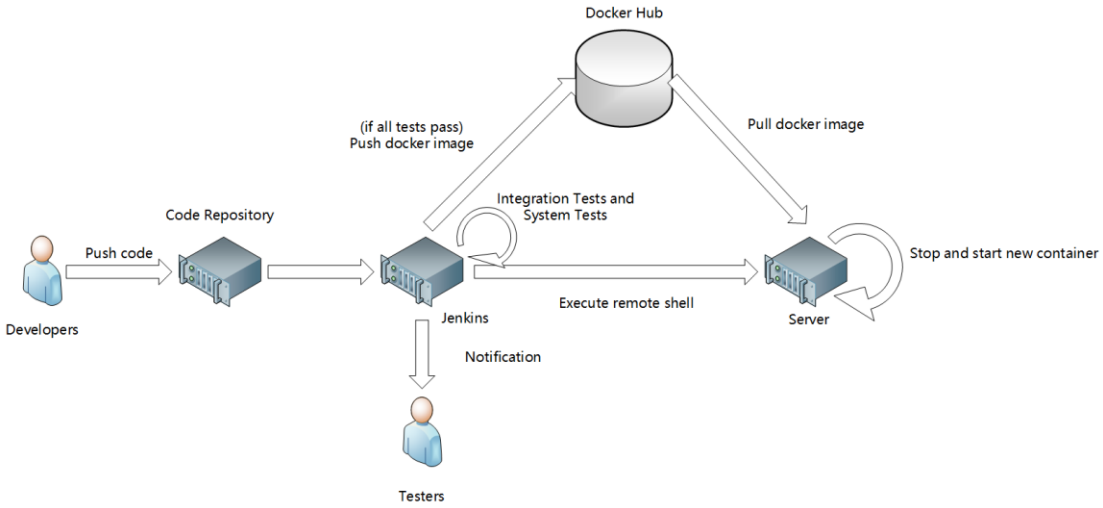


图 2.2 CI/CD 架构图

在该架构下，频繁的系统测试和集成测试能加快项目的迭代、更新，在出现问题时也能在早期解决，总体上减少了 debug 的成本。

项目的测试重点包括：前端页面应能正确解析并显示后端的数据，同时后端应确保业务逻辑的正确性，主要包括用户登录、编辑、推荐、分享等功能。

项目的测试难点包括：我们选择前后端分离进行开发，则相对应的测试也是较为独立的。我们建立的 mock server 帮助前端独立于后端进行开发，帮助编写单元测试。对此，我们选择采用 Mock.js 模拟后端的数据进行测试。在系统各接口均确定之后再进行集成前后端的系统测试；后端的测试将会使用 Django 的 LiveServerTestCase 进行单元测试和集成测试。
我们的测试将分为如下几个层次：

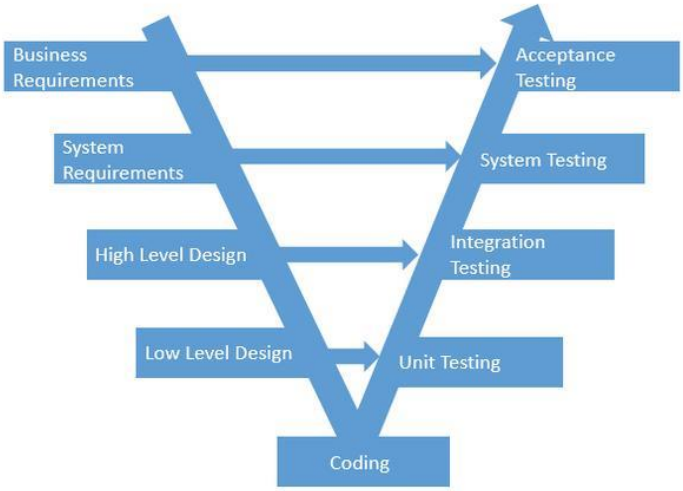


图 2.3 测试分层图（图片来自 http://www.anexinet.com/wp-content/uploads/Acceptance.png）

总体上的测试由下至上为单元测试、集成测试、系统测试和用户的接受测试(上线之后)。其中前后端的测试又相对独立。在进行系统测试时则会进行前后端的集成。

项目的测试目标有如下几点：

1. 前端测试的测试目标首先是各个单元的正确性，其次则是代码的覆盖率。在保证代码的正确性的前提下，通过编写尽可能多的单元测试来覆盖单元的所有功能。
2. 集成测试的测试目标的测试目标主要是验证前端或后端的各个模块业务逻辑的正确性。
3. 系统测试的测试目标主要是验证前后端的系统能够正确满足所有需求，其次是系统的各项非功能性需求（如产品的响应时间、产品的兼容性、产品的抗压测试等等）。

2.1.2 测试范围

我们对测试广度的理解为：测试应从尽可能多的角度对产品进行测试。我们将对前后端的代码进行功能性测试，确保使用逻辑的完整与正确；性能测试，保障用户的使用体验；安全测试，防御常见攻击（csrf 等）；易用性测试，确保用户能快速上手；稳定性测试，保障服务抗压能力；兼容性测试，保障 web 前端在不同主流浏览器上都能正常运行。

我们对测试深度的理解为：测试应对产品的每一个特性或功能进行尽可能多、尽可能深的测试。通过编写大量样例，确保测试将会覆盖前后端的所有代码。每次测试都将会进行代码覆盖率的分析，确保项目的所有特性都能在测试中被覆盖到。

2.1.3 测试环境

所有测试将会运行在有英特尔芯片的机器上。其中前端测试运行在 Node.js 环境中，测试工具为 Karma，Mock。后端测试运行在 python 环境中。测试工具为 Django 和 Tornado 自带的单元测试工具。CI/CD 平台将使用 Travis CI、GitHub 和 Docker Hub 来搭建。