分布式架构实验报告

学　　院：信息工程学院

指导教师：吴倩

班　　级：19 计算机科学与技术 1 班

学生姓名：何淇（19011429）

武伊雪（19011387）

赵琰晴（19011385）

日期：2021 年 11 月 15 日

摘要

在学生成绩评价系统中，前端使用 echarts、layui、jquery、thymeleaf 等技术，后端使用 maven、spring boot 、Mybatis、mysql 等技术，使老师和学生可以通过 Web 网页，发出 Request 请求，然后从 mysql 数据库中查询到学生提交 commits 总数、lines of code changed per week per user per project，并且将所有的数据绘制成 echarts 图表，更为形象立体地展示整体数据。

学生成绩评价系统产品遵循“指标配置灵活”、“评价场景多元”、“评价过程轻松”、“评价数据要有用”四大设计原则，坚持“多元主体参与”、“多数据输入”，老师、学生共同参与全过程。通过数据精准化，功能轻量化，帮助学校和教师高效率完成成绩评估工作。

Abstract

In Student-Grade-Accessment-System, the front-end uses technologies such as echarts, layui, jquery, thymeleaf, and the back-end uses technologies such as maven, spring boot, Mybatis, mysql, etc., so that teachers and students can send Request requests through Web pages, and then from the mysql database In the query, the total number of commits submitted by the students, lines of code changed per week per user per project, and all the data are drawn into an echarts chart to display the overall data more vividly and three-dimensionally.

The product of Student-Grade-Accessment-System follows the four design principles of "flexible indicator configuration", "multiple evaluation scenarios", "easy evaluation process", and "useful evaluation data". It adheres to "multiple subject participation" and "multiple data input". Students participate in the whole process together. Through accurate data and lightweight functions, it helps schools and teachers complete performance evaluation work efficiently.

目录

学生成绩评价系统

一、项目介绍

学生成绩评价系统产品遵循“指标配置灵活”、“评价场景多元”、“评价过程轻松”、“评价数据要有用”四大设计原则，坚持“多元主体参与”、“多数据输入”，老师、学生共同参与全过程。通过数据精准化，功能轻量化，帮助学校和教师高效率完成成绩评估工作。

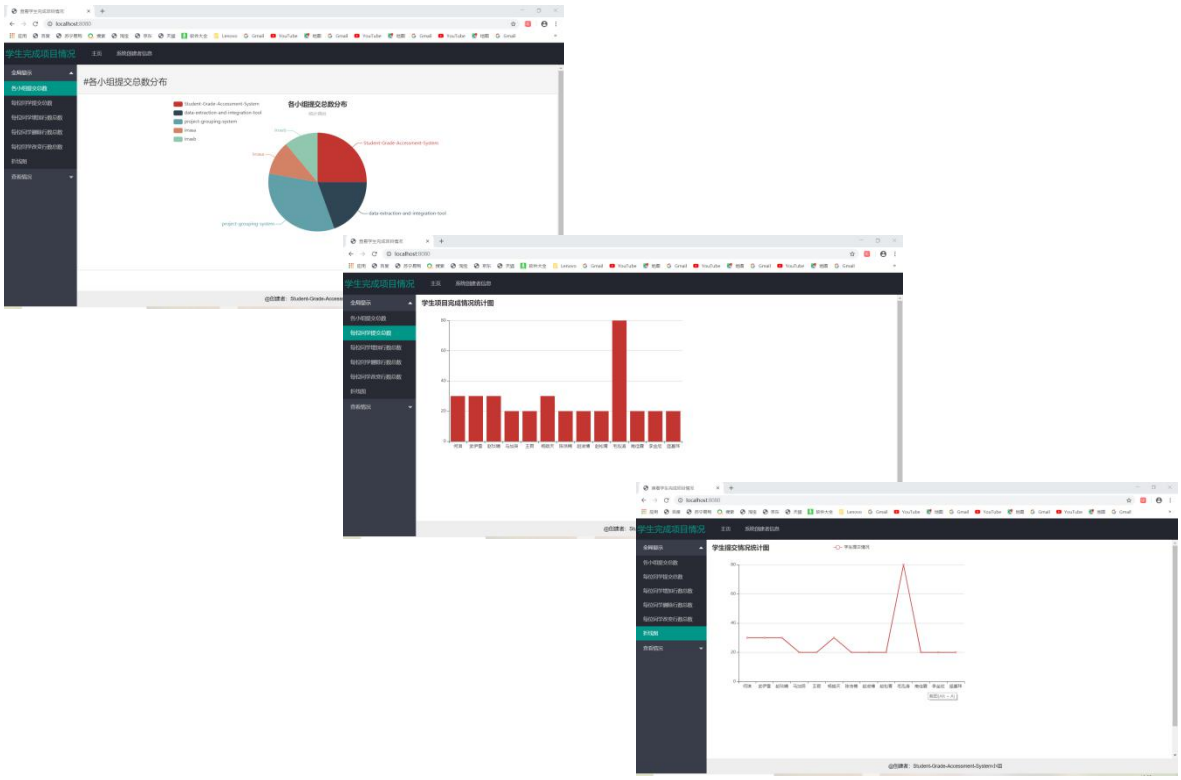


图 1　Web 大屏 + 多端评价结果输出

（一）项目功能体系结构图

根据对学生成绩评价系统进行用例分析画划分，本组将学生成绩管理系统划分为“学生提交作业次数管理”、“学生修改代码行数管理”、“学生基本细腻管理”、“学生作业整体评价管理”、“学生成绩信息备份管理”5 个用例，以下是功能体系结构图：

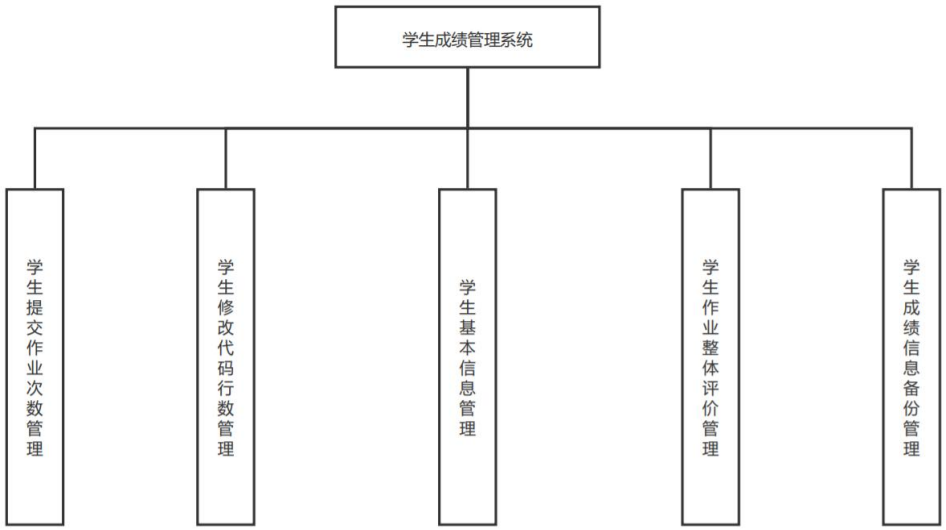


图 功能体系结构图

（二）项目用例解析

在学生成绩评价系统中，系统的角色包括“Student”、“Teacher”、“Database”、“Github”共计 4 个角色，通过业务流程将系统角色与用例进行关联，最终可得初始用例图如下：

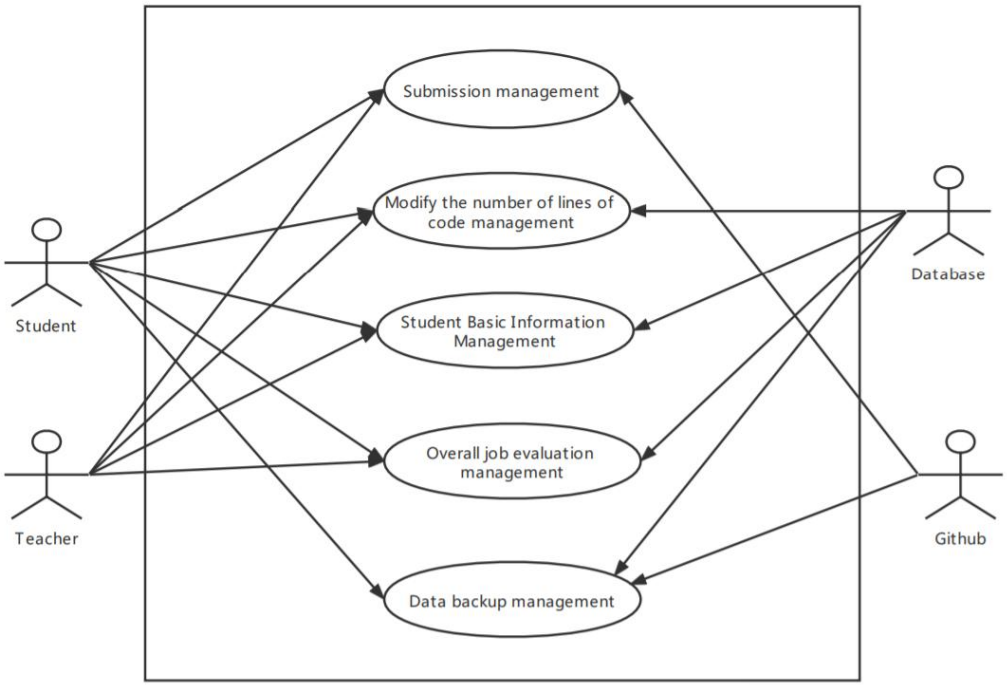


图 2　基础用例图

（三）系统工作流程图

系统的总体功能区可以划分为两个部分：

根据 url 和 token 从 Github API 返回 commit 数据，并将数据统一存入 MySQL 数据库

利用 spring boot 框架，解析前端发送来的请求，从数据库中获取相应信息，最后动态展示到前端网页上

因此，系统工作流程图如下所示：

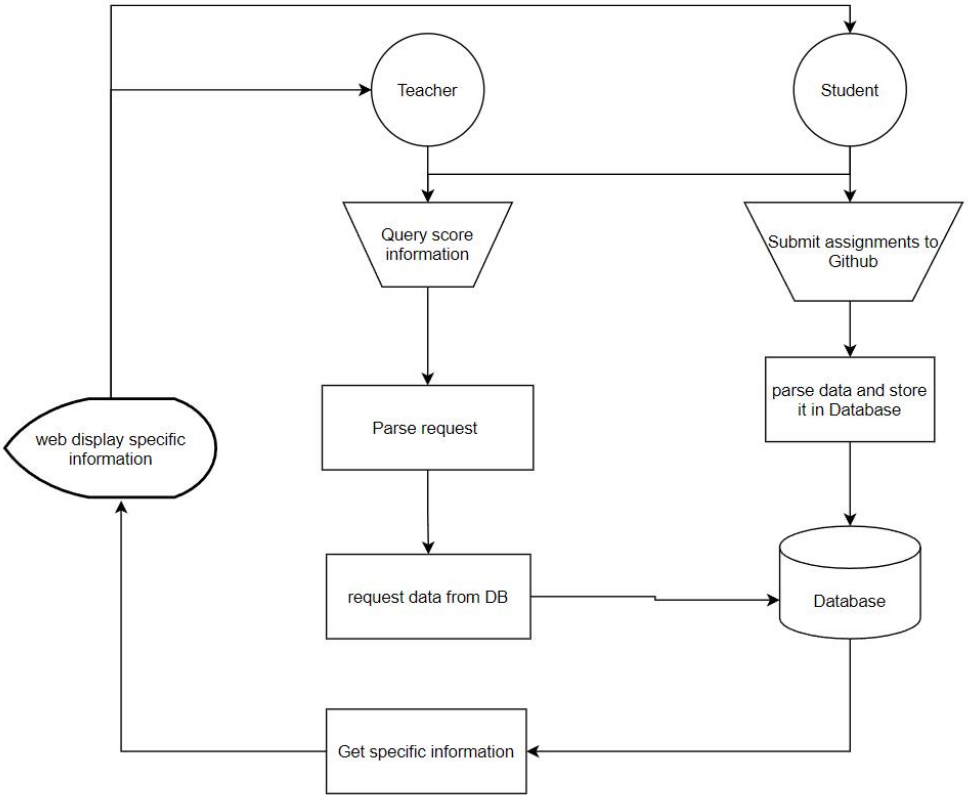

Teacher
Student
Query score information
Submit assignments to Github
parse data and store it in Database
Parse request
web display specific information
request data from DB
Database
Get specific information

图 3　系统工作流程图

二、构建 C4-model

（一）System Context diagram

1. System Context diagram

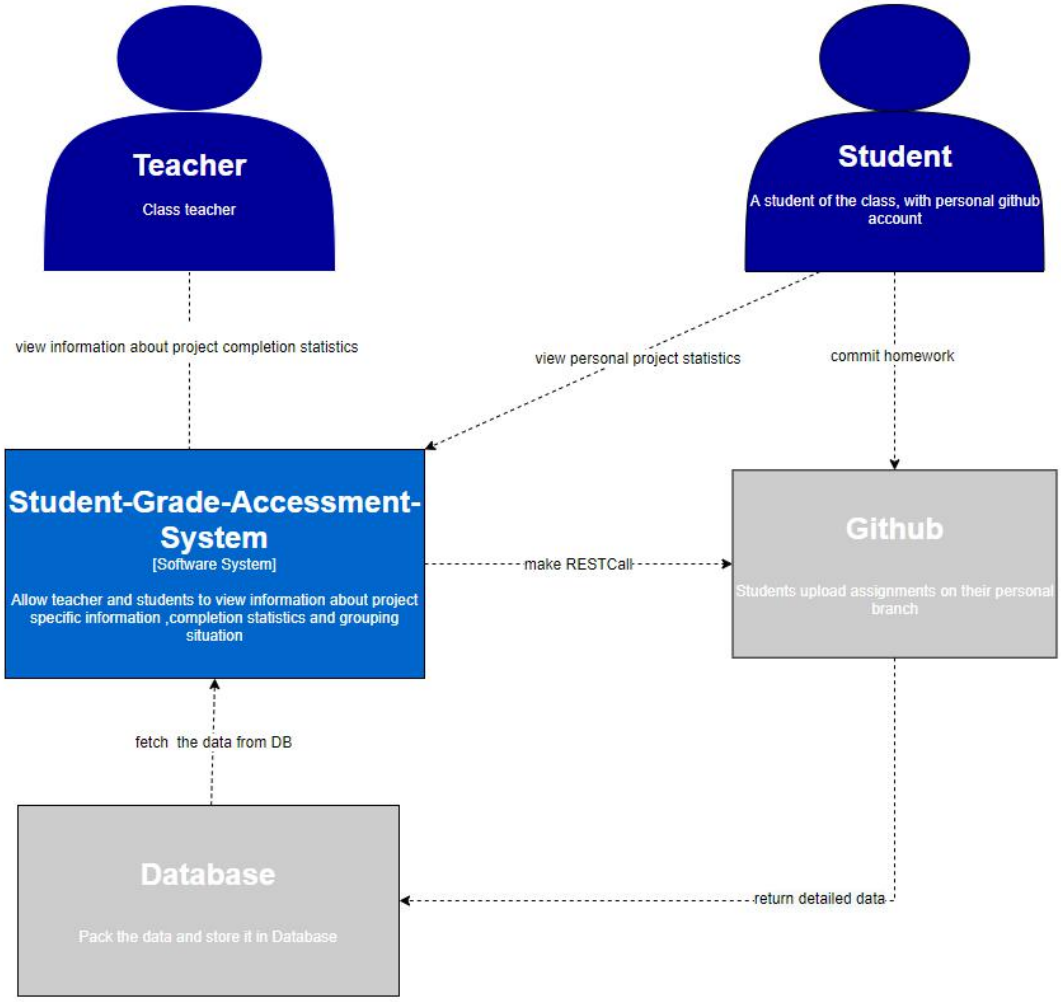


图 4 System Context diagram

2. System Context diagram 详细介绍

学生成绩评价系统实现从 Githee repositories 中收集数据，展示每个项目进度及个人贡献，最后以图表展示形式作出报告。例如展示每个学生的 commits 总数、lines of code changed per week per user per project。主要功能通过访问 Gitee API(java)实现数据获取，并生成一个 Library，提供给同组的成员调用，然后在 web 页面展示结果。

（1）学生在 github 提交数据，系统

在学生成绩评价系统中，每个学生通过 Git Bash Here 向 Github 的 repositories 的个人 branch 传输提交代码。然后访问 Gitee API(java)实现数据获取，对 Json 解析，获取最终处理好的数据。


```
"html_url": "https://github.com/Distributed-System-Course/Student-Grade-Accessment-System",
"description": "学生项目评价系统",
"fork": false,
"url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System",
"forks_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/forks",
"keys_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/keys{/key_id}",
"collaborators_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/collaborators{/collaborator}",
"teams_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/teams",
"hooks_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/hooks",
"issue_events_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/issues/events{/number}",
"events_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/events",
"assignees_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/assignees{/user}",
"branches_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/branches{/branch}",
"tags_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/tags",
"blobs_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/git/blobs{/sha}",
"git_tags_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/git/tags{/sha}",
"git_refs_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/git/refs{/sha}",
"trees_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/git/trees{/sha}",
"statuses_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/statuses/{sha}",
"languages_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/languages",
"stargazers_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/stargazers",
"contributors_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/contributors",
"subscribers_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/subscribers",
"subscription_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/subscription",
"commits_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/commits{/sha}",
"git_commits_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/git/commits{/sha}",
"comments_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/comments{/number}",
"issue_comment_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/issues/comments{/number}",
"contents_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/contents/{+path}",
"compare_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/compare/{base}...{head}",
"merges_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/merges",
"archive_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/{archive_format}{/ref}",
"downloads_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/downloads",
"issues_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/issues{/number}",
"pulls_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/pulls{/number}",
"milestones_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/milestones{/number}",
"notifications_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/notifications{?since,all,participating}",
"labels_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/labels{/name}",
"releases_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/releases{/id}",
"deployments_url": "https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/deployments",
"created_at": "2021-10-13T03:22:05Z",
"updated_at": "2021-11-18T07:12:41Z",
"pushed_at": "2021-11-18T07:17:16Z",
"git_url": "git://github.com/Distributed-System-Course/Student-Grade-Accessment-System.git",
"ssh_url": "git@github.com:Distributed-System-Course/Student-Grade-Accessment-System.git",
```


图　访问组织的 api 界面

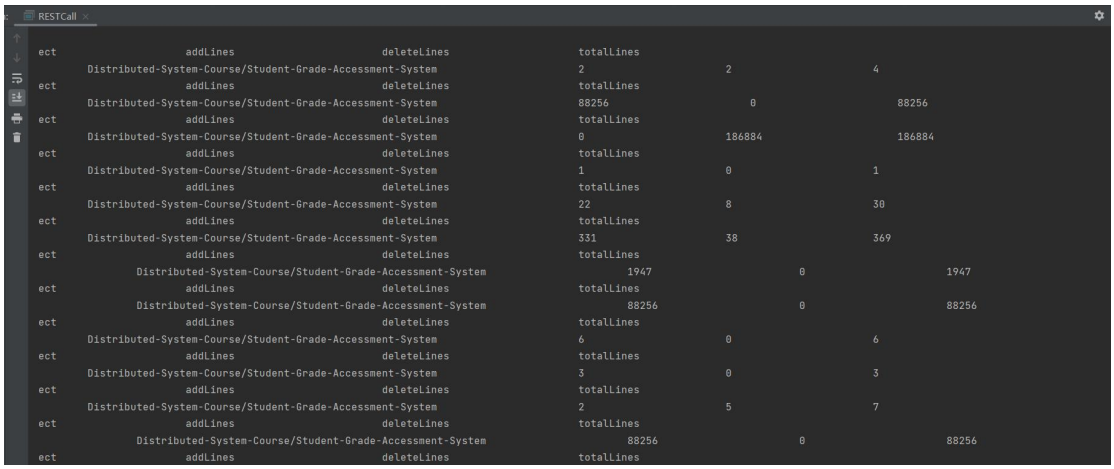


图　Json 解析后的数据展示

在学生成绩评价系统中，老师和学生可以通过 Web 网页，发出 Request 请求，从 mysql 数据库中查询到学生提交 commits 总数、lines of code changed per week per user per project，并且将所有的数据绘制成 echarts 图表，更为形象立体地展示整体数据。

下面，将展示几张 echarts 图表：

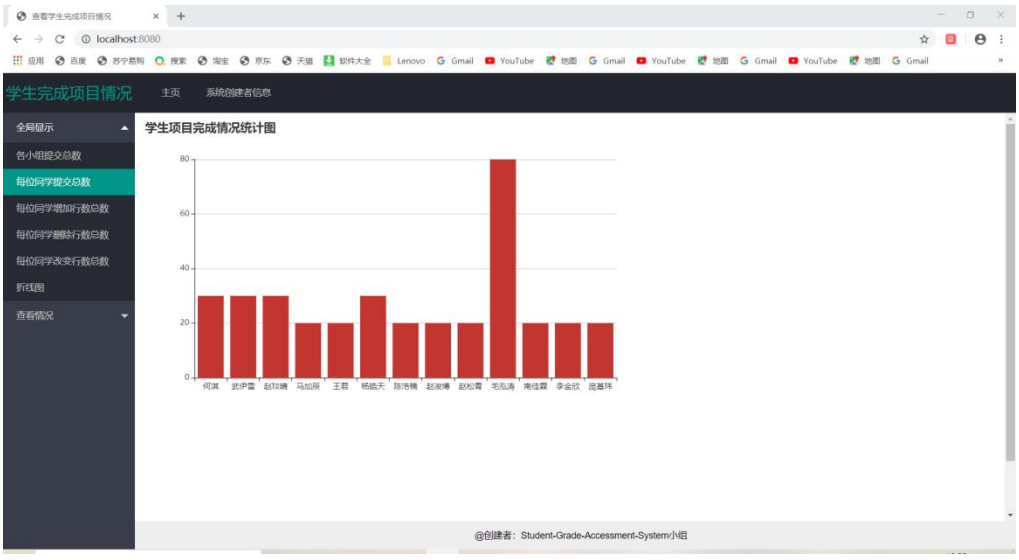


图　学生项目完成情况统计图

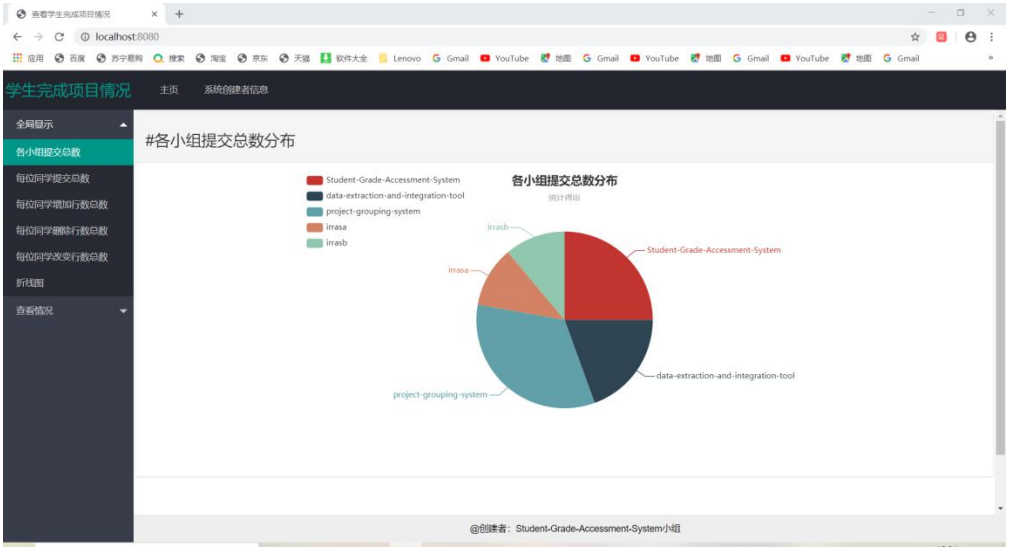


图　各小组提交总数分布统计图

（二）Container diagram

1. Container diagram

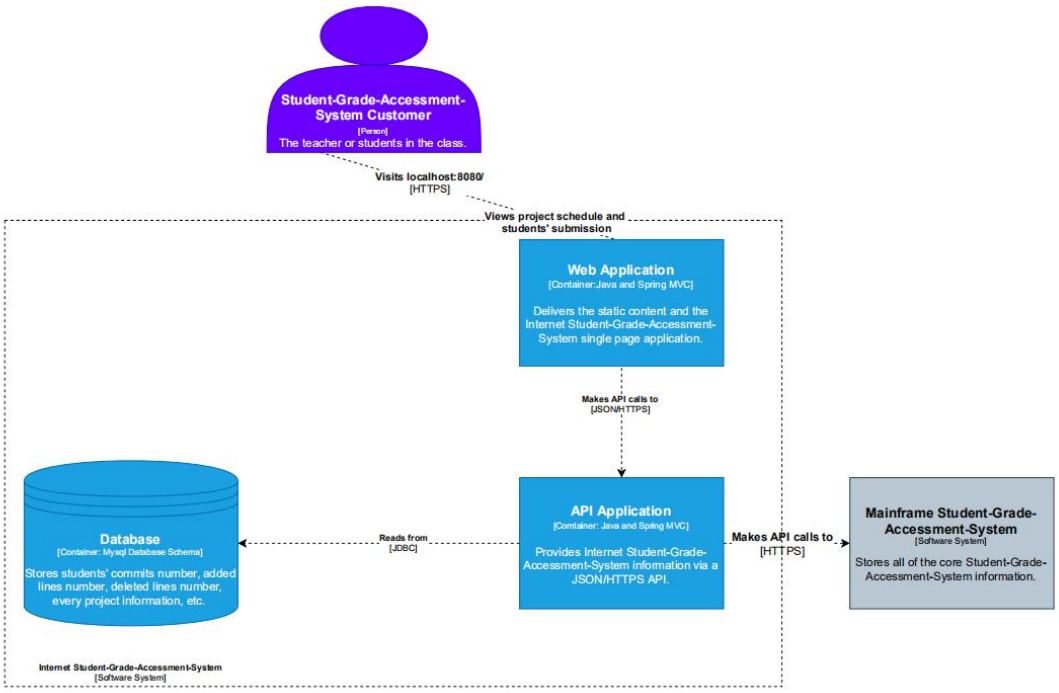


图 Container diagram

2. Container diagram 详细介绍

学生成绩评价系统的用户是老师以及课堂中的其他同学。老师想要查看每位同学以及每个小组的项目完成情况，同学想要查看自己的项目完成情况。

学生成绩评价系统（虚线框内）由四个容器组成：服务器端 Web 应用程序、单页应用程序、服务器端 API 应用程序和数据库。

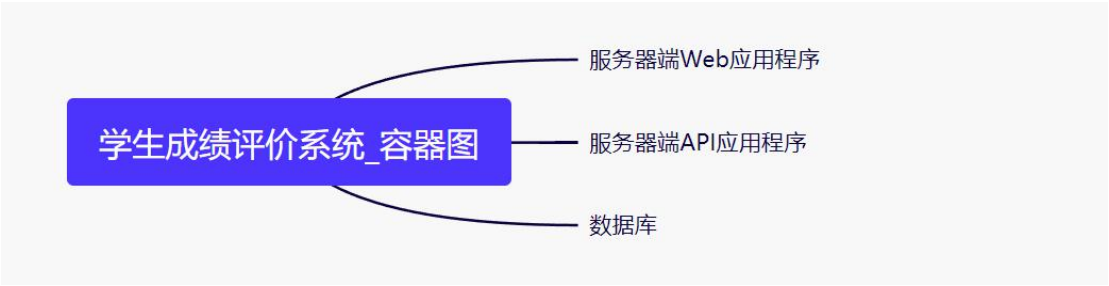


图　学生成绩评价系统_容器组成

容器组成详细介绍如下：

（1）Web 应用程序是一个 Java/Spring MVC Web 应用程序，它只提供静态内容（HTML/CSS 和 JavaScript），包括构成单页应用程序的内容。

（2）API 应用程序还使用 HTTP 接口与 Github 上每个小组的提交情况通信，以获取有关的完成情况信息。

（三）Component diagram

1. Component diagram

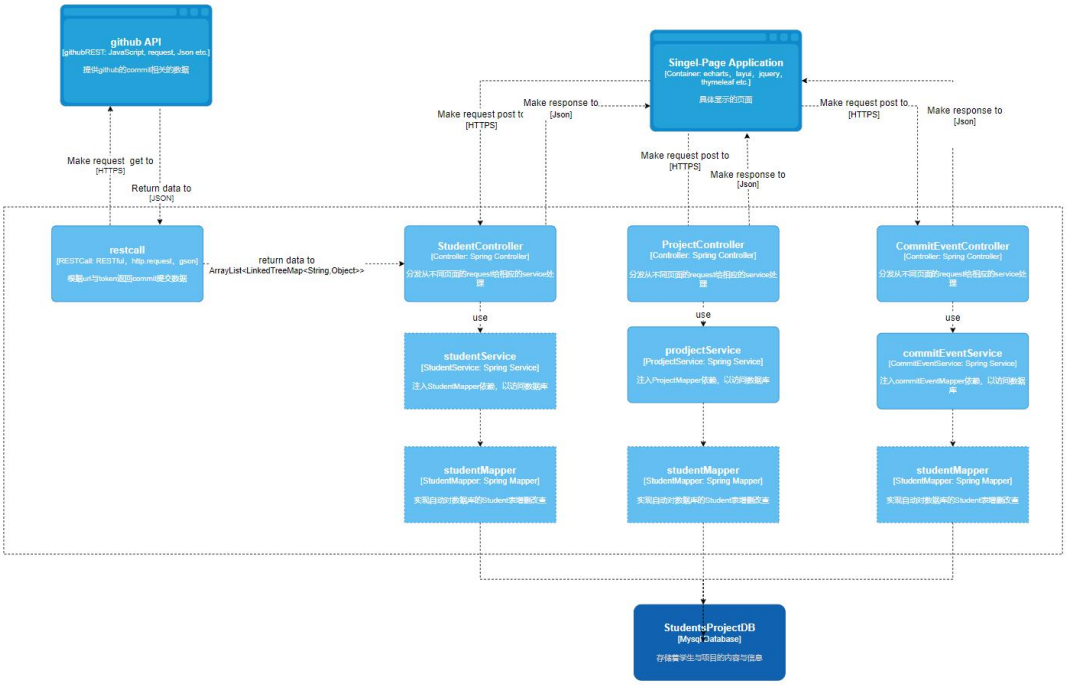
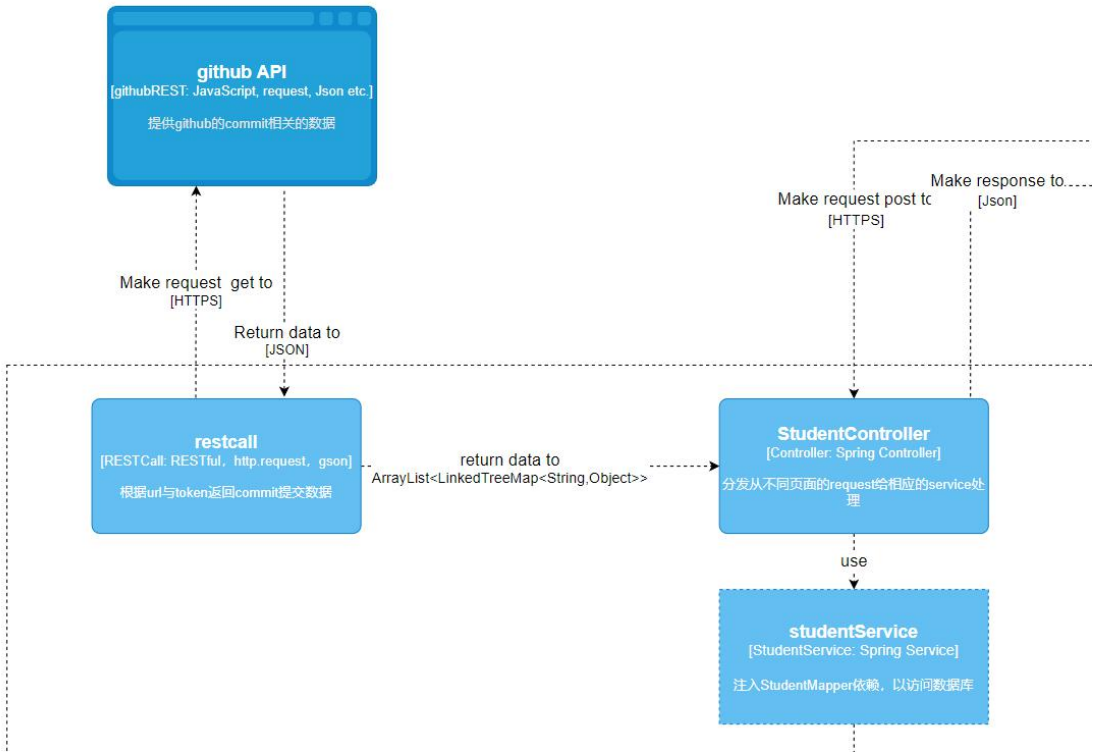


图 Component diagram

如上图所示，组件图大体上可以划分为后端、前端两个部分。

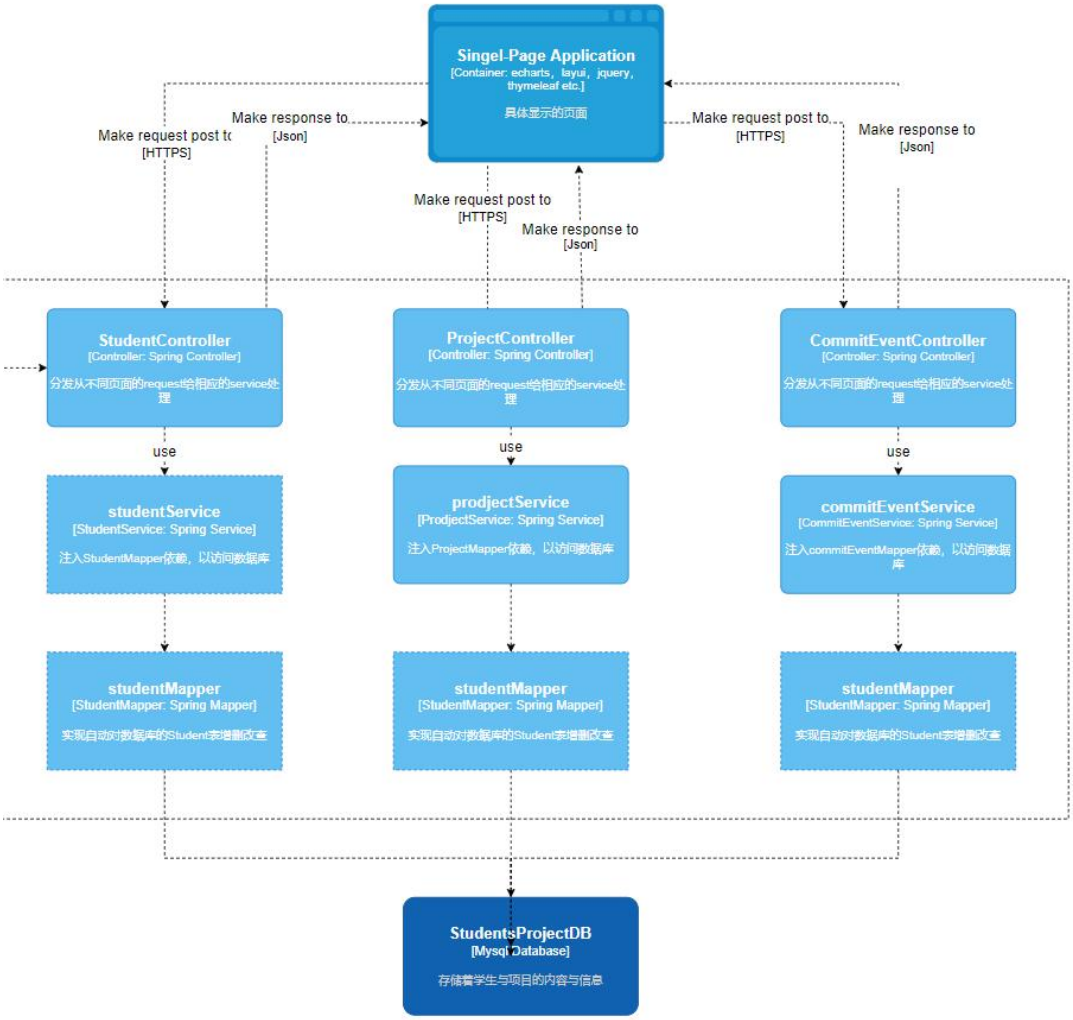
2. Component diagram 详细介绍

（1）后端



后端主要凭借 Github token，由 makeRESTCall()函数，向 Github API 发出数据请求。Github 将请求的数据返回给 RESTCall，并将数据从 StudentController 返回 Spring boot 容器。StudentController 通过使用方法 StudentService，将数据写回数据库，并保存。

（2）前端

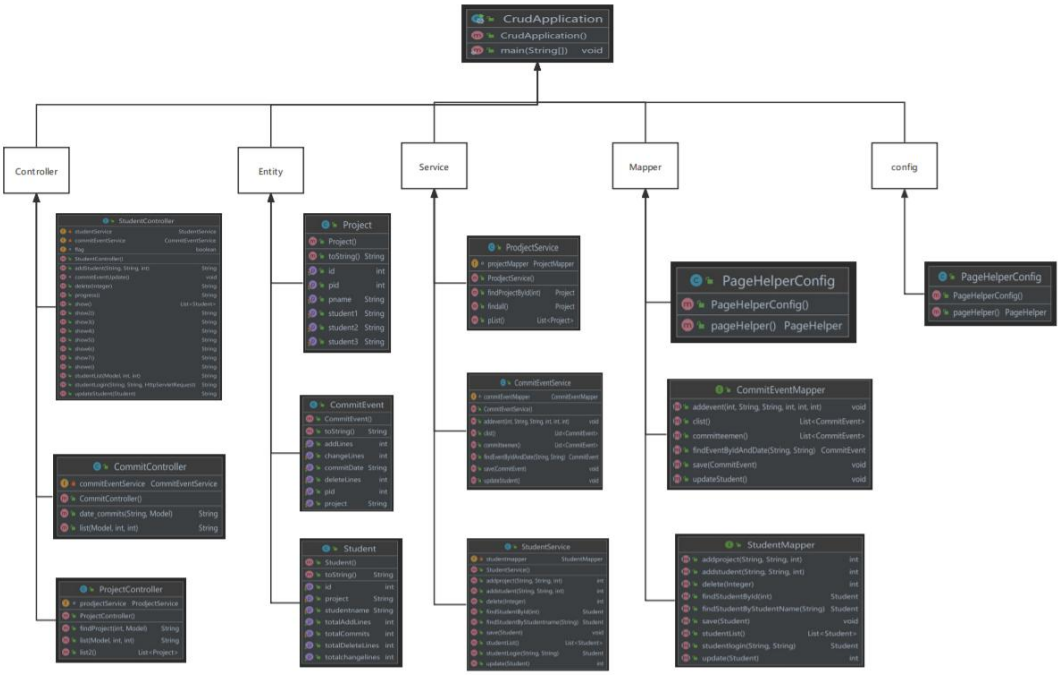


前端主要是 3 个实体，分别为 Student、Project、commitEvent，3 个实体配套会产生相应的 bean、service、controller、mapper。

用户从网页 html 中发出请求，使用 ajax 和 url 等方式将请求发送给 controller 控制器，控制器经简单处理后，调用 service 和 serviceImp 将处理进行具体处理。由于 service 中会注入 mapper，这样，就同时实现了对数据库和 xml 文件的调用和数据访问。

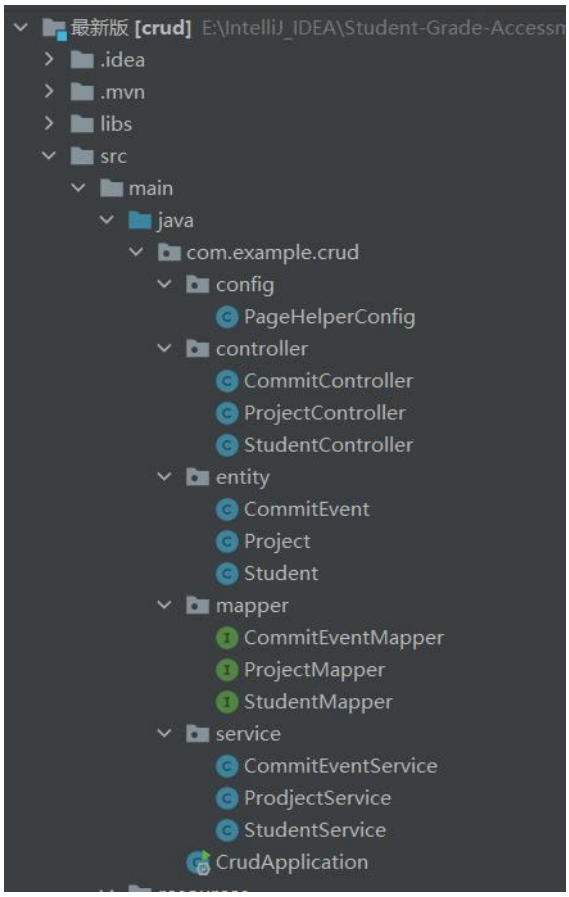
（四）Code

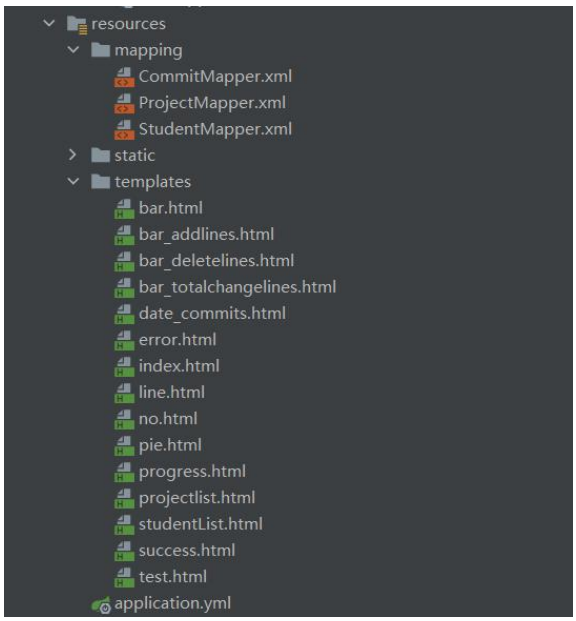
1. UML 图



最后，放大每个组件以显示它是如何作为代码实现的；使用 UML 类图、实体关系图或类似的。

2. Spring boot 目录展示

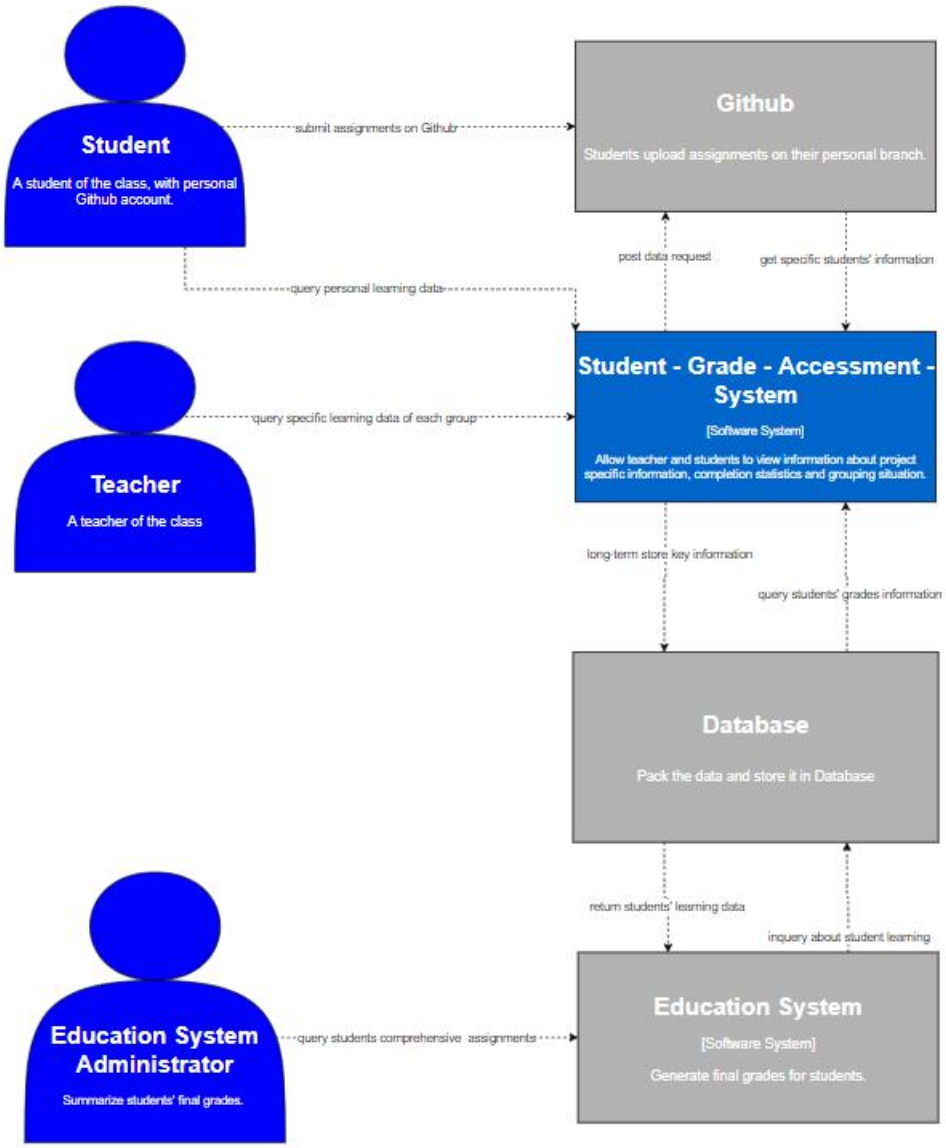




三、补充图表

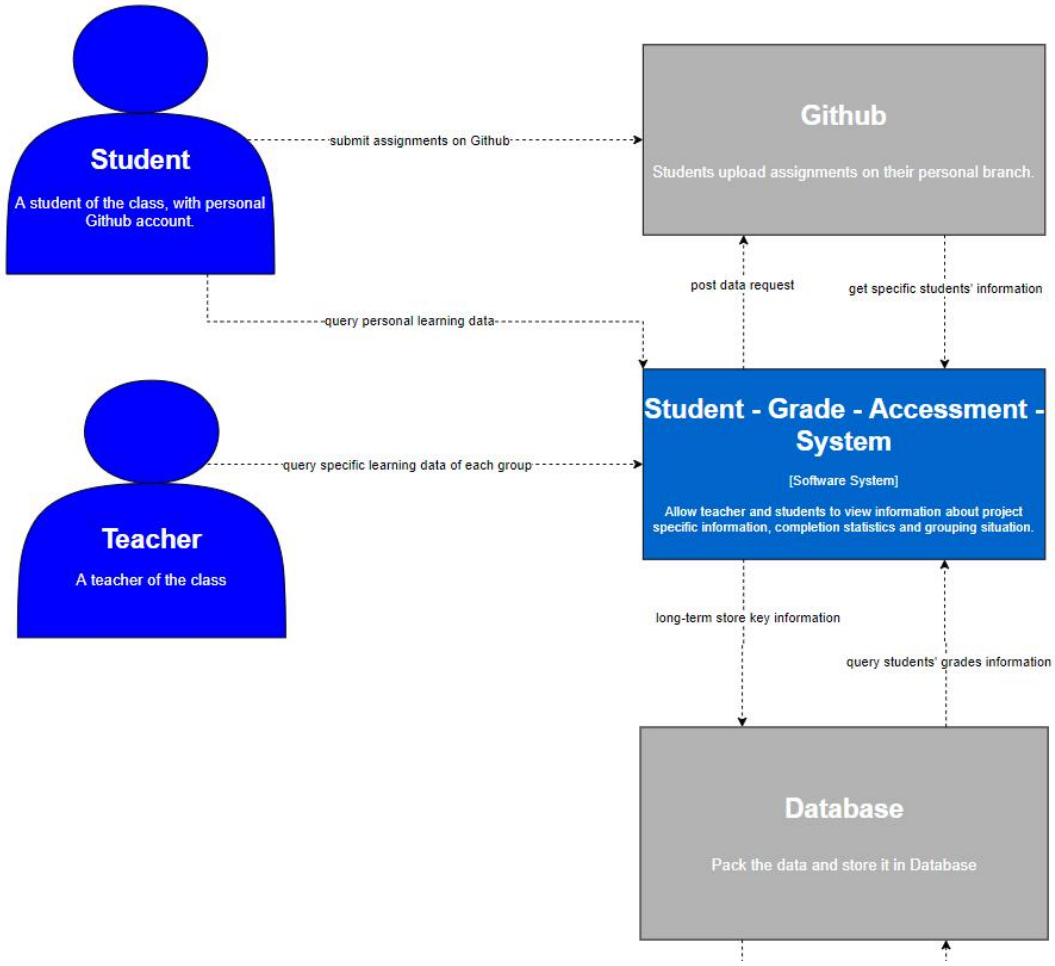
（一）System Landscape diagram

1. System Landscape diagram

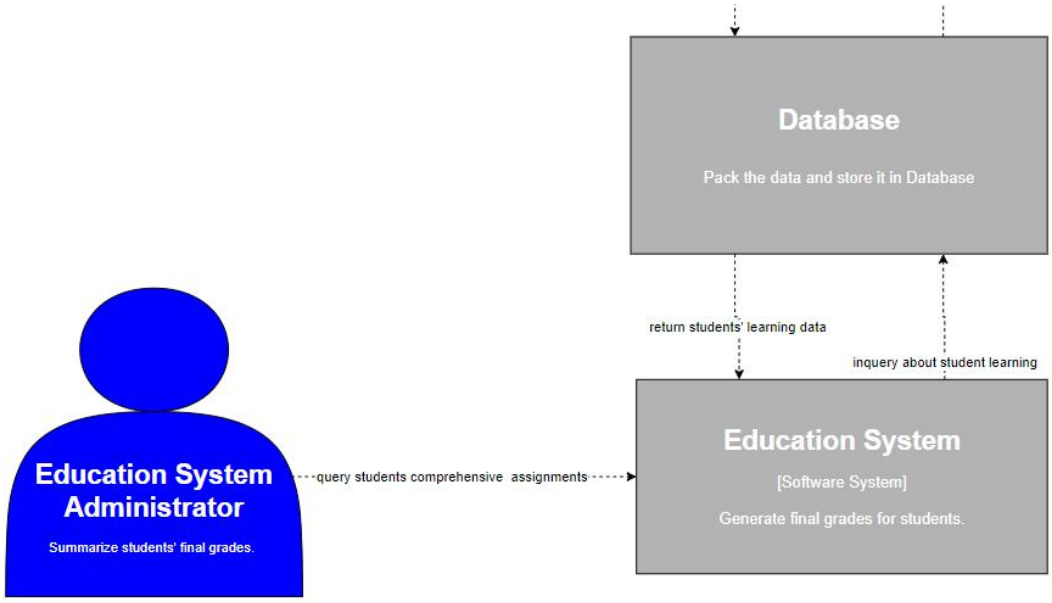


2. System Landscape diagram 详细介绍

（1）学生成绩管理系统与学生、老师、数据库的交互

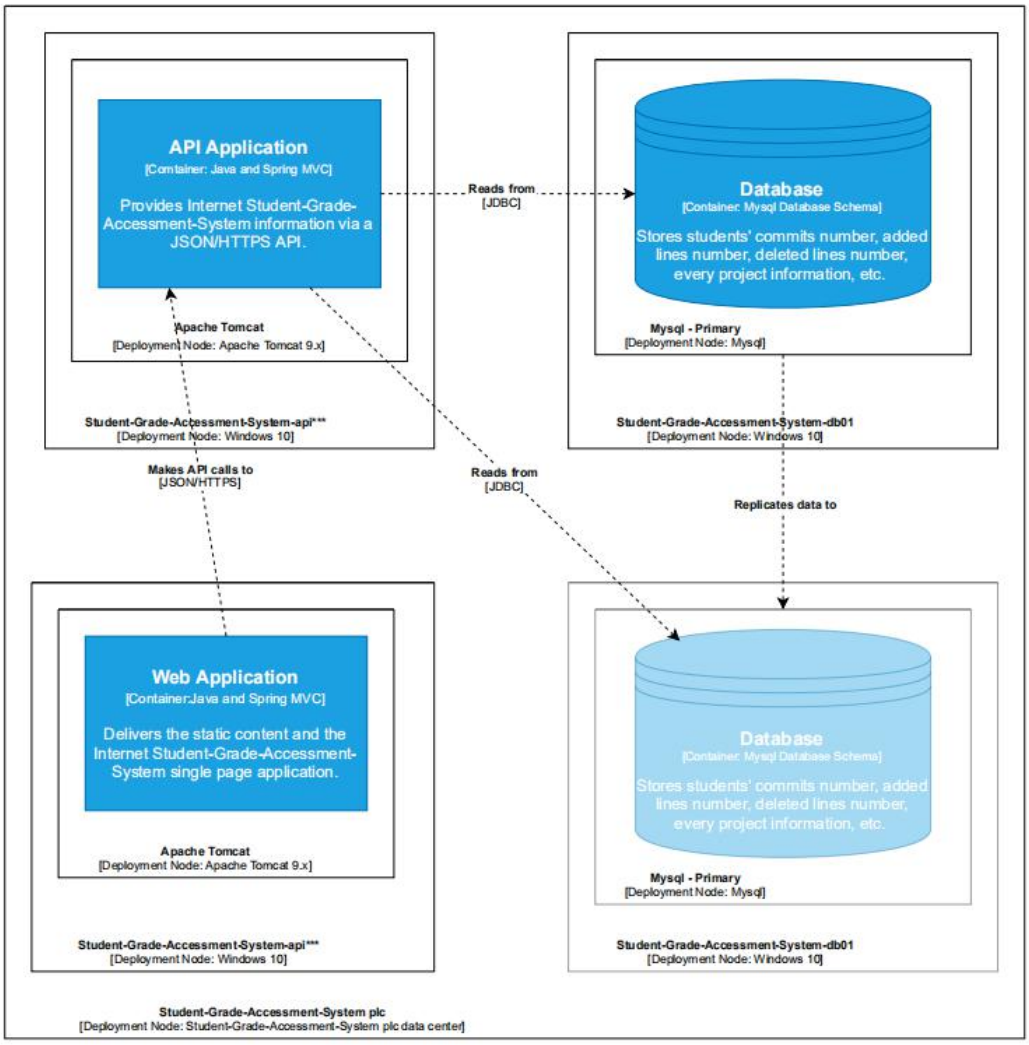


（2）学生成绩评价系统还可以与教务系统相关联，通过学生成绩评价系统，教务系统管理员可以查询学生学习的具体信息。管理员通过对学生学习情况综合测评，最终生成学生的综合测评成绩。



（二）Deployment diagram

1. Deployment diagram



2. Deployment diagram 详细介绍

从部署图中可以看到，每个容器都被两个框所框住，然后右半部分的四个容器被一个大框框住，形成了整个部署图。

API Application 这个容器使用的是 Java 以及 Spring MVC，提供了学生完成项目的信息，使用了 Apache-Tomcat 9.0。可以被单页应用程序所要求响应，同时也可以从数据库中读入数据。

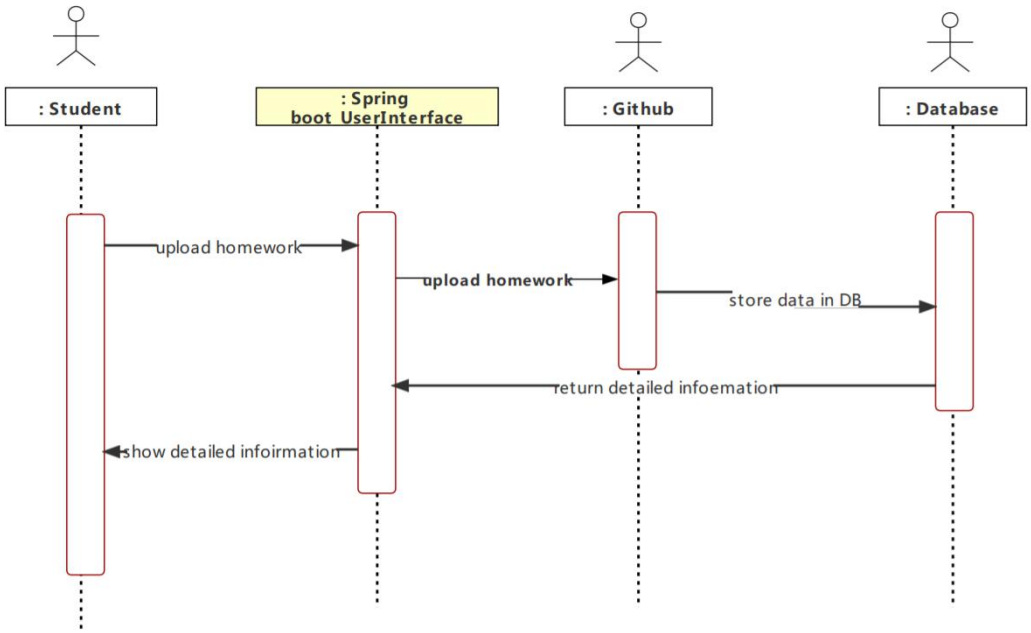
这里画了两个数据库，一个是真正使用的数据库，还有一个备份数据库，备份数据库将数据库中的信息复制到备份数据库中。数据库使用的是 Mysql 数据库。

Web 应用程序同样使用 Java 以及 Spring MVC 来运行，提供了静态信息和单页应用程序中的信息。

（三）Dynamic diagram

1. Dynamic diagram

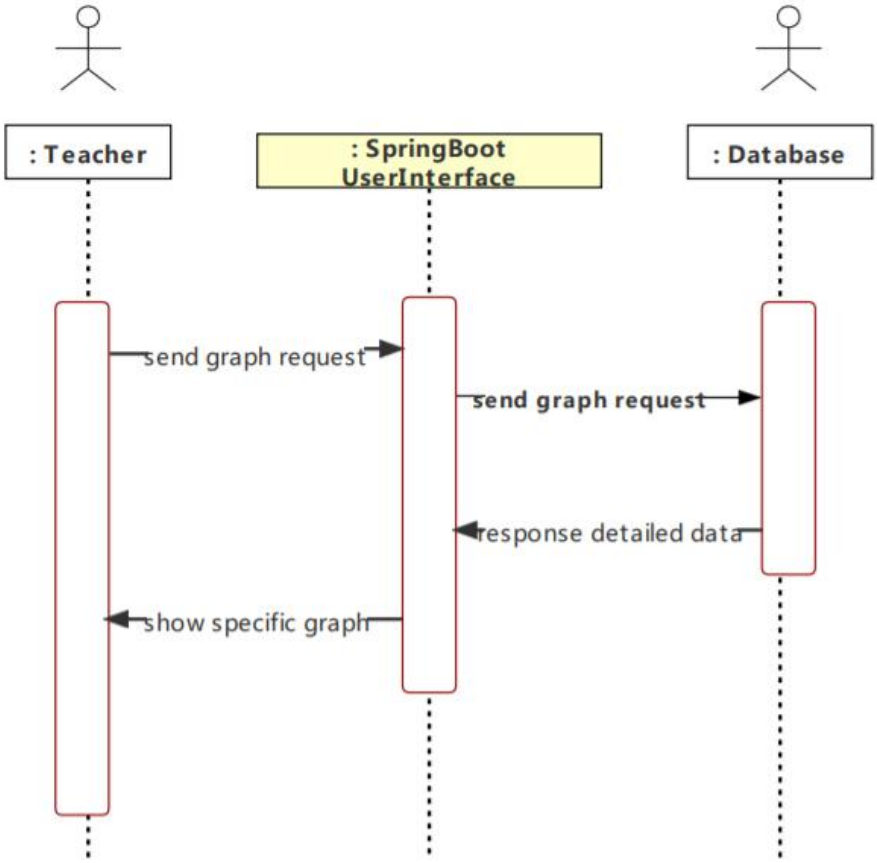
（1）Student 的序列图



① 学生主要进行的行为分为两种，如下图所示：

- 学生的行为
  - 使用Git上传作业到个人branch
    - 增加个人commit总数
    - 增加个人代码行数总数 / 删除行数总数 / 改变行数总数
    - 提高个人作业完成情况
  - 查看个人学习情况
    - 查看个人commit总数
    - 查看个人代码行数总数 / 删除行数总数 / 改变行数总数
    - 查看个人作业完成情况
    - 查看小组成员作业完成情况

（2） Teacher 的时序图



老师主要进行一种行为，如下图所示：

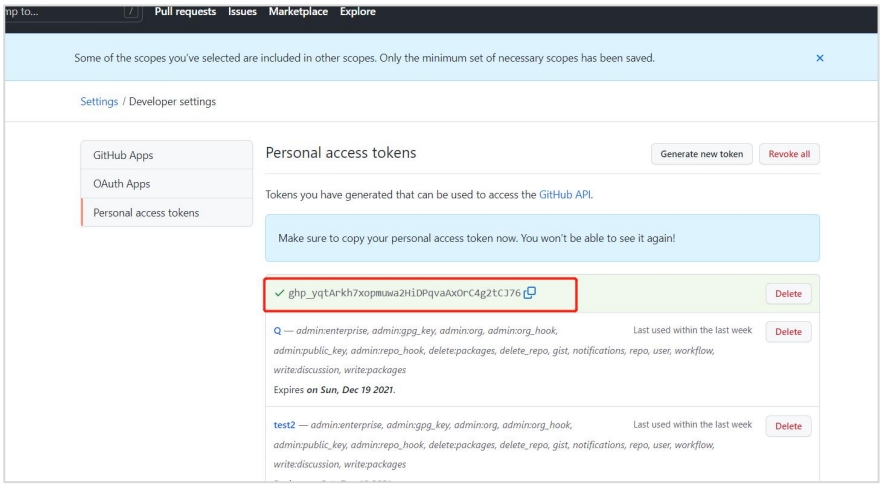
四、spring boot 程序详细介绍

五、RESTCall 程序详细介绍

（一）使用个人访问令牌（token）——进行身份验证

未经身份验证的客户端每小时可以发出 60 个请求。 要每小时发出更多请求，我们需要进行*身份验证*。 事实上，使用 GitHub API 进行任何交互都需要身份验证。

所以，创建个人 token 结果如下：



（二）查找仓库 API

1. 为何选择 GitHub Search API

GitHub Search API 可以实现高效地搜索特定项目。

Search API 可搜索要查找的特定项目。例如，在存储库中查找用户或特定文件。当在谷歌上执行搜索的方式。它旨在帮助用户找到正在寻找的一个结果（或者可能是正在寻找的几个结果）。就像在 Google 上搜索一样，有时希望查看几页搜索结果，以便找到最能满足用户需求的项目。为满足这一需求，GitHub Search API 为每次搜索提供多达 1,000 个结果。而搜索 API 依赖于需要作为 URL 请求参数发送的搜索“查询”

2. github-api 查看具体项目信息

以 Distributed-System-Course/Student-Grade-Accessment-System 为例

（1）小组网址

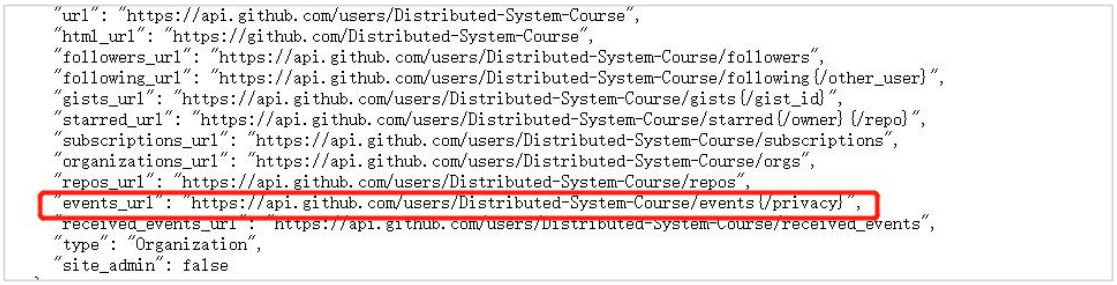
已知 GitHub-api： https://api.github.com/repos/{:owner}/{:repository}

所以，本组的仓库 API 为 https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System，其中，只要替换“Student-Grade-Accessment-System”为其他组的项目名称，就可以对别的组进行查询。

（2）根据 events 查找具体信息

打开下面的网址，

https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System



这里本组根据 events 进行具体查询：

最终代码如下：

```
String prefix="https://api.github.com/repos/Distributed-System-Course/";
String suffix="/events";
String projectName="Student-Grade-Accessment-System";
String url=prefix+projectName+suffix;
String token="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";//personal
url="https://api.github.com/repos/Distributed-System-Course/Student-Grade-Accessment-System/events";
```

（三）REST 调用

1. REST 调用

这里，本组使用 Apache commons HTTP 客户端库和 Apache Fluent httpcomponents 库来对 GitHub API 端点进行 REST 调用。

代码如下：

```
String makeRESTCall(String restUrl, String acceptHeaderValue,String token)
```

```
            throws ClientProtocolException, IOException {


        //REST 调用
        //使用 Apache commons HTTP 客户端库和 Apache Fluent httpcomponents 库来对
GitHub API 端点进行 REST 调用。
        // 我们将使用 GSON 进行 JSON 响应解析。
        Request request = Request.Get(restUrl);
        //request.viaProxy("127.0.0.1:7891");
        //request.socketTimeout(30000);
        if (acceptHeaderValue != null && !acceptHeaderValue.isBlank()) {
            request.addHeader("Accept", acceptHeaderValue);
        }
        if(token!=null){
            request.addHeader("Authorization", "token "+token);
        }
        else
            System.out.println(restUrl);

        request.addHeader("User-Agent", "Mozilla/5.0");
        Content content = request.execute().returnContent();
        String jsonString = content.asString();
        //System.out.println("content = " + jsonString);

        // To print response JSON, using GSON. Any other JSON parser can be used here.
        //Map jsonMap = gson.fromJson(jsonString, Map.class);
        return jsonString;
    }

    String makeRESTCall(String restUrl) throws ClientProtocolException, IOException {
        return makeRESTCall(restUrl, null,null);
```

```
}
```

（四）JSON 解析

当 Json 字符串转换成 Java 对象时，转换成的是 JsonObject，并不是想要的 Class 类型的对象，操作起来就很不方便。因此，我们使用 Google Gson,从而实现 Json 结构的相互转换

1. 需要把 Google 的 Gson 的 Jar 包导入到项目中

2. 具体解析流程

成员变量和构造函数如下（使用线程池，多线程提高运行速率）：

```
ArrayList<LinkedTreeMap<String,Object>> table;
    ThreadPoolExecutor poolExecutor;
    public RESTCall(){
        table= new ArrayList<LinkedTreeMap<String,Object>>();
        poolExecutor = (ThreadPoolExecutor) Executors.newFixedThreadPool(10);
    }
```

（1） 将 String 类型的 JSON 转化为 JsonArray

```
        //Json 的解析类对象
        JsonParser parser = new JsonParser();
        //将 JSON 的 String 转成一个 JsonArray 对象
        JsonArray jsonArray = parser.parse(jsonString).getAsJsonArray();
```

（2） 泛型的使用

由于这里 root 有自定义泛型，所以我们使用 foreach 语句和多线程，将 JSON 字符串转化为对象，代码如下：

```
            for(JsonElement jsonElement:jsonArray){
                POJO.Event.Root root = gson.fromJson(jsonElement , POJO.Event.Root.class);
                if (root==null) break;



                LinkedTreeMap<String,Object> row= new LinkedTreeMap<String,Object>();
                row.put("author",root.getActor().getLogin());
                row.put("date",parseDate(root.getCreated_at()));
                row.put("project",root.getRepo().getName());


if(root.getPayload().getCommits()!=null&& !root.getPayload().getCommits().isEmpty()){
                    String commitUrl=root.getPayload().getCommits().get(0).getUrl();
                    FutureTask<LinkedTreeMap<String,Object>> task = new
FutureTask<LinkedTreeMap<String,Object>>((
                    )->{
                        parseCommit(commitUrl,token,row);
                        return row;
                    });
                    results.add(task);
                    poolExecutor.execute(task);
                }
```

然后，使用 foreach 语句，将解析后的结果存入成员变量中：

```
            for(FutureTask<LinkedTreeMap<String, Object>> result:results) {
                    try {
                  table.add(result.get());
                  } catch (InterruptedException e) {
                  e.printStackTrace();
```

```
                    } catch (ExecutionException e) {
                    e.printStackTrace();
                    }
            }
```

这里，对于 date 和 commit 进行了详细的解析，代码分别如下：

parseDate:

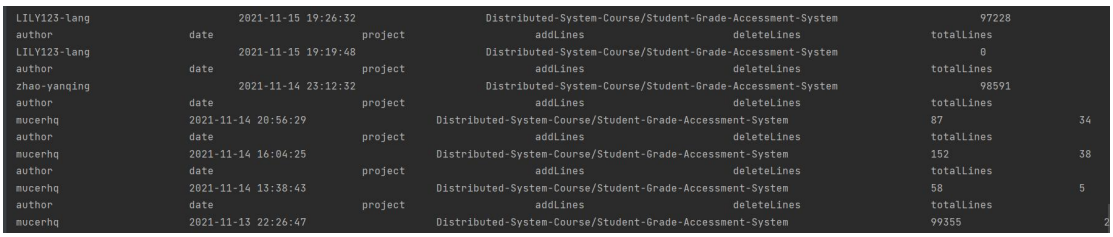
```
String parseDate(String raw){
            String str=raw;
            str=str.replaceAll("T"," ");
            str=str.replaceAll("Z","");
            SimpleDateFormat beijingSDF = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            beijingSDF.setTimeZone(TimeZone.getTimeZone("Asia/Shanghai"));//北京时区
            SimpleDateFormat gitSDF = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");    //git
时间格式
            gitSDF.setTimeZone(TimeZone.getTimeZone("GMT+00:00"));    // 设置 git 时区
            Date date = null;    // 将字符串时间按 git 时间解析成 Date 对象
            try {
                date = gitSDF.parse(str);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            return beijingSDF.format(date).toString();
    }
```

parseCommit:

```
void parseCommit(String url,String token,LinkedTreeMap<String,Object>row){
            String jsonString = null;
            try {
```

```
            jsonString = makeRESTCall(url,null,token);
        } catch (IOException e) {
            e.printStackTrace();
        }
        Gson gson = new Gson();
        POJO.Commits.Root root=gson.fromJson(jsonString,POJO.Commits.Root.class);
        row.put("addLines",root.getStats().getAdditions());
        row.put("deleteLines",root.getStats().getDeletions());
        row.put("totalLines",root.getStats().getTotal());
    }
```

（3） 如果将结果输出在控制台上，结果如下：



六、实验中的思考

（一） RESTFUL——RPC 对比

1. RPC

（1）概念介绍

RPC 即远程过程调用（Remote Procedure Call Protocol，简称 RPC），像调用本地服务(方法)一样调用服务器的服务(方法)。通常的实现有 XML-RPC，JSON-RPC，通信方式基本相同，所不同的只是传输数据的格式.

（2）RPC 是分布式架构的核心，按响应方式分如下两种：

① 同步调用：客户端调用服务方方法，等待直到服务方返回结果或者超时，再继续自己的操作

② 异步调用：客户端把消息发送给中间件，不再等待服务端返回，直接继续自己的操作。

2. REST

（1）REST

REST 即表述性状态传递（Representational State Transfer，简称 REST），是一种软件架构风格。REST 通过 HTTP 协议定义的通用动词方法(GET、PUT、DELETE、POST），以 URI 对网络资源进行唯一标识，响应端根据请求端的不同需求，通过无状态通信，对其请求的资源进行表述。

（2）Rest 架构的主要原则：

①网络上的所有事物都被抽象为资源

②每个资源都有一个唯一的资源标识符

③ 同一个资源具有多种表现形式(xml,json 等)

④ 对资源的各种操作不会改变资源标识符

⑤ 所有的操作都是无状态的

（3）RESTful 风格的体现

使用了 get 请求，就是查询；使用 post 请求,就是新增的请求；使用 put 请求，就是修改的请求；使用 delete 请求，就是删除的请求。这样做就完全没有必要对 crud 做具体的描述。

（4）如何在 spring 中使用

http 方法	资源操作	幂等	安全

GET	SELECT	是	是
POST	INSERT	否	否
PUT	UPDATE	是	否
DELETE	DELETE	是	否

① 幂等性：对同一 REST 接口的多次访问，得到的资源状态是相同的。

② 安全性：对该 REST 接口访问，不会使服务器资源的状态发生改变。

③ Spring boot 原生态支持了 REST 风格的架构风格设计,涉及的注解例如：@RequestMapping、@ResponseBody 等。

（二）POJO 和 JavaBean 的区别

1. POJO

（1）POJO 基本概念

POJO（Plain Ordinary Java Object），简单普通的 java 对象。主要用来指代那些没有遵循特定的 java 对象模型，约定或者框架的对象。没有业务逻辑，有时可以作为 VO(value-object)或 dto(Data Transform Object)来使用.当然,如果你有一个简单的运算属性也是可以的,但不允许有业务方法,也不能携带有 connection 之类的方法。

（2）POJO 的基本内容:

①有一些 private 的参数作为对象的属性,然后针对每一个参数定义 get 和 set 方法访问的接口。

② 没有从任何类继承、也没有实现任何接口，更没有被其它框架侵入的 java 对象。

2. JavaBean

（1）JavaBean 基本概念

JavaBean 是一种 JAVA 语言写成的可重用组件。JavaBean 符合一定规范编写的 Java 类，不是一种技术，而是一种规范。大家针对这种规范，总结了很多开发技巧、工具函数。符合这种规范的类，可以被其它的程序员或者框架使用。它的方法命名，构造及行为必须符合特定的约定：

（2）JavaBean 特点

①所有属性为 private。

②这个类必须有一个公共的缺省构造函数。即是提供无参数的构造器。

③ 这个类的属性使用 getter 和 setter 来访问，其他方法遵从标准命名规范。

④ 这个类应是可序列化的。实现 serializable 接口。

（3）两者的区别

①POJO 其实是比 javabean 更纯净的简单类或接口。POJO 严格地遵守简单对象的概念，而一些 JavaBean 中往往会封装一些简单逻辑。

②POJO 主要用于数据的临时传递，它只能装载数据， 作为数据存储的载体，而不具有业务逻辑处理的能力。

④ Javabean 虽然数据的获取与 POJO 一样，但是 Javabean 当中可以有其它的方法。

七、实验总结

在本次实验中，我们深入了解了 spring boot 的工作原理，也对 Github API 的使用更

为熟悉。我们不仅学到了专业的知识，也对增强了我们动手做项目的能力。由于整个作业是面向应用的，最后做出来的结果也是面向用户的，我们能将理论知识与实践相结合，极大地提高了我们的创新实践能力。我们受益匪浅！

参考文献：

https://docs.github.com/cn/rest/guides/getting-started-with-the-rest-api

https://itsallbinary.com/github-rest-api-search-files-content-pull-requests-commits-programmatically-using-java-without-cloning/

https://blog.csdn.net/qxs965266509/article/details/42774691

https://blog.csdn.net/qq_32505207/article/details/108921869?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216372254371678025526420 2%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=163722543716780255264202&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-1-108921869.pc_search_result_cache&utm_term=POJO&spm=1018.2226.3001.4187