# 1引言：虚拟校园概述

## 本项目基于Web 3D与虚拟现实的最新技术，设计和开发一款三维网络教学平台，旨在使教师授课更真实，让课堂更生动，资源共享更便捷从而改善用户的学习体验，提高在线学习效果。下面将从三个方面入手介绍本项目：项目进度汇报，技术分析及实现和测试环境说明。

# 2项目进度汇报：目前实现的功能

## 2.1前端

## 2.2后端

# 3技术分析及实现

## 3.1前端

## 3.2后端

### 3.2.1预期达成的效果

### 前端页面中展示的信息，都由后台的 SpringBoot 微服务提供，如地图数据，地标数据，学院信息，工作室信息，虚拟人员信息，工作室内的短视频，

### 阶段任务和子任务信息，评定信息，任务进阶点信息等， 而本题组主要完成前

### 端场景的开发、虚拟道具的开发、故事场景的切换。根据要求，目前需要与后台完成在进入链接虚拟学院时的登录。

### 3.2.2逻辑组成

### Spring MVC中的三层分别是Model, View, Controller. 当用户向服务器发送了请求后, 如HTTP请求, Controller先将请求映射到某个方法上, 方法根据该请求进行相应处理, 返回Model, 它封装储存了所有的页面数据。Model会被送到View层, 也就是用户界面层, View层将自己本身的模板和Model内部的数据结合成为完整的一个页面, 作为response返还给用户, 用户便看到了页面。本项目前后端分离,前端采用Web3D技术，在此的 View层的意义已经不大,本节作为后端, 主要专注于Model和Controller。

### 所有封装完毕的页面数据通过虚拟接口获取，定义标准 JSON 数据格式。接口采用REST组件并沿用其规范，分离接口和功能。

### 3.2.3软件组成

#### 3.2.3.1项目构建及依赖管理

首先根据maven工程，使用自己的Group ID和Artifact ID创建项目。完毕后在pom.xml中配置项目依赖：

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.4.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.howlowhello</groupId>  
 <artifactId>spring-boot-service-unavailable</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>springdemo2</name>  
 <description>Demo project for Spring Boot</description>  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

依赖说明：

Thymeleaf — 辅助生成用于测试的HTML界面

JPA — 主要借助其包含的Hibernate实现将对象映射到数据库

MySQL — 是实现Java到MySQL连接的一个中间件

Spring Web — 包含Tomcat容器，Spring MVC，REST等核心组件

### 

#### 3.2.3.2代码实现

##### 3.2.3.2.1在application.properties文件中配置数据库

application.properties文件的作用是去配置这些依赖，目前需要配置的是从Spring Boot到我们项目数据库的连接

该文件可以在maven工程的src/main/java/resources/applicatio.properties下找到

该文件中有如下属性:

spring.datasource.username=How\_Low\_Hello

spring.datasource.password=\*\*\*\*\*\*\*\*\*\*

spring.datasource.url=jdbc:mysql://localhost:3306/springdb

spring.jpa.hibernate.ddl-auto=create

第三行的url就是我们数据库的地址, 3306是MySQL默认的本地端口, 而springdb是之前创建的数据库.

第一行和第二行声明了用户名密码和账户名,

第四行的create为开发模式, 就是每次应用启动都重新创建一个新表, 原有的表会清空, 开发结束后将其设置为none.

声明了这些之后, 启动应用后, Spring会自动用上述的信息连接并登陆数据库

##### 3.2.3.2.2项目Entities

Entity 即被映射到数据库的java object，这里以项目中的User类为例，其中储存了用户的登录数据

**import** javax.persistence.Entity**;**

**import** javax.persistence.GeneratedValue**;**

**import** javax.persistence.GenerationType**;**

**import** javax.persistence.Id**;**

@Entity

**public** **class** **User** **{**

@Id

@GeneratedValue**(**strategy**=**GenerationType**.**AUTO**)**

**private** Integer id**;**

**private** String name**;**

**private** String email**;**

**private** String password**;**

@Override

**public** String **toString()** **{**

**return** "{" **+**

" id='" **+** getId**()** **+** "'" **+**

", name='" **+** getName**()** **+** "'" **+**

", email='" **+** getEmail**()** **+** "'" **+**

", password='" **+** getPassword**()** **+** "'" **+**

"}"**;**

**}**

**public** String **getPassword()** **{**

**return** **this.**password**;**

**}**

**public** **void** **setPassword(**String password**)** **{**

**this.**password **=** password**;**

**}**

**public** Integer **getId()** **{**

**return** **this.**id**;**

**}**

**public** **void** **setId(**Integer id**)** **{**

**this.**id **=** id**;**

**}**

**public** String **getName()** **{**

**return** **this.**name**;**

**}**

**public** **void** **setName(**String name**)** **{**

**this.**name **=** name**;**

**}**

**public** String **getEmail()** **{**

**return** **this.**email**;**

**}**

**public** **void** **setEmail(**String email**)** **{**

**this.**email **=** email**;**

**}}**

通过Spring Boot的@Entity注解, Spring框架会得知这个类是一个Entity, Hibernate会把这个类映射到数据库中的关系表, 这个关系表的列与User的所有属性一一对应

通过创建更多的Entity，可以较为轻易地实现项目需要的数据存储及管理。如地图数据，地标数据，学院信息，工作室信息，虚拟人员信息，工作室内的短视频，阶段任务和子任务信息，评定信息，任务进阶点信息等

##### 3.2.3.2.3Repository接口

这些声明的接口代表的是存储数据类的“库”，这里拿之前展示的User类为例，我们创建了一个UserRepository接口，代表的是储存这些用户信息的“库”

**import** java.util.List**;**

**import** org.springframework.data.repository.CrudRepository**;**

**public** **interface** **UserRepository** **extends** CrudRepository**<**User**,** Integer**>** **{**

List**<**User**>** **findByEmail(**String email**);**

**void** **deleteByEmail(**String email**);}**

即使只是声明一个接口, 而并没有具体实现, Spring也会自动完成对User信息的储存.

该类继承了CrudRepository, 也就是支撑"增删改查"的一个Repository, 当未来我们需要在别的地方调用这个接口的方法时, 只需按照方法名的规范创建相应方法即可, 这里我们拿邮箱查找为例, 使用findByEmail(String email)方法, Spring会自动将findByEmail转换成SQL语言中的`SELECT \* FROM UserRepository WHERE email = 'email'

##### 3.2.3.2.4项目Controller

Controller是接受用户的url请求, 将url映射到某个具体的方法, 处理url请求, 再返回一个Model, 是Spring MVC中的C层.

当用户向服务器发送请求后,  Controller先将请求映射到某个方法上, 方法根据该请求进行相应处理, 返回Model

示例文件: UserController.java:

**import** java.util.List**;**

**import** org.slf4j.Logger**;**

**import** org.slf4j.LoggerFactory**;**

**import** org.springframework.beans.factory.annotation.Autowired**;**

**import** org.springframework.stereotype.Controller**;**

**import** org.springframework.ui.Model**;**

**import** org.springframework.web.bind.annotation.GetMapping**;**

**import** org.springframework.web.bind.annotation.RequestParam**;**

**import** org.springframework.web.bind.annotation.ResponseBody**;**

@Controller**public**

**class** **UserController** **{**

**private** **static** **final** Logger log **=** LoggerFactory**.**getLogger**(**UserController**.**class**);**

@Autowired

**private** UserRepository userRepository**;**

@GetMapping**(**path**=**"/add"**)** *// Map ONLY GET REQUESTs.* **public** @ResponseBody String **addNewUser** **(**@RequestParam String name

**,** @RequestParam String email**,** @RequestParam String password**,** User user**)** **{**

*// @ResponseBody means the returned String is a response, not a view name.* user**.**setName**(**name**);**

user**.**setEmail**(**email**);**

user**.**setPassword**(**password**);**

userRepository**.**save**(**user**);**

log**.**info**(**user**.**toString**()+**" saved to the repo"**);**

**return** "Saved"**;**

**}**

*/\*\* \* 登陆方法, 用户输入邮箱和密码, 查询数据库检验是否有该账户,如果有, \* 返回原先页面 ,登陆成功。 \* @param email 用户邮箱 \* @param password 用户密码 \* @param model Spring MVC中的Model，用来储存经过controller处理后的信息，再由View层渲染 \* 得到前端页面。 \* @return \*/*

@GetMapping**(**path **=** "/login"**)**

**public** String **login(**@RequestParam String email**,** @RequestParam String password**,** Model model**)** **{**

List**<**User**>** users **=** userRepository**.**findByEmail**(**email**);**

*// 如果数据库中未查到该账号:* **if** **(**users **==** **null)** **{**

log**.**warn**(**"attempting to log in with the non-existed account"**);**

**return** "该用户不存在"**;**

**}** **else** **{**

User user **=** users**.**get**(**0**);**

**if** **(**user**.**getPassword**().**equals**(**password**))** **{**

*// 如果密码与邮箱配对成功:* model**.**addAttribute**(**"name"**,** user**.**getName**());**

log**.**warn**(**user**.**toString**()+** " logged in"**);**

**}** **else** **{**

*// 如果密码与邮箱不匹配:* model**.**addAttribute**(**"name"**,** "logging failed"**);**

log**.**warn**(**user**.**toString**()+** " failed to log in"**);**

**}**

**return** "index"**;**

**}**

**}**

*/\*\* \* 查看所有用户的注册信息，按照Spring Boot的设定，以Json的形式输送给用户端。 \* @return \*/*

@GetMapping**(**path**=**"/all"**)**

**public** @ResponseBody Iterable**<**User**>** **getAllUsers()** **{**

**return** userRepository**.**findAll**();**

**}**

*/\*\* \* 域名的根目录，然后返回的“index”会映射到 \* java/resources/templates/index.html文件。 \* @param name \* @return \*/*

@GetMapping**(**path**=**"/"**)**

**public** String **welcomePage(**@RequestParam**(**name**=**"name"**,** required**=false,** defaultValue**=**"World"**)**

String namel**){**

**return** "index"**;**

**}**

**}**

@AutoWired 被注有它的注解会被Spring得知, Spring会自动注入依赖. 如上述的userRepository, Spring在运行过程中会通过IoC容器注入一个UserRepository的实例给userRepository.

@GetMapping 将使用Get方法的HTTP请求映射到被注的方法.

@RequestParam 将HTTP请求中用户传入的参数映射到变量中括号内指定变量中.

@ResponseBody 它表示该方法返回的值就是Response本身, 不需传递至View被渲染, 用户直接得到该方法的返回值.