

南京航空航天大学

**软件设计模式与体系结构**

**课程报告**

**班级：\_\_\_\_\_\_\_1619303\_\_\_\_\_\_**

**学号：\_\_\_\_\_\_061920125\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_郑伟林\_\_\_\_\_\_**

**指导老师：\_\_\_\_\_周宇\_\_\_\_\_\_\_\_**

**2021年6月**

目录

[一、Spring框架架构 3](#_Toc75704675)

[1.1 什么是框架？ 3](#_Toc75704676)

[1.2 Spring 框架结构 3](#_Toc75704677)

[二、MVC 4](#_Toc75704678)

[2.1 MVC结构 4](#_Toc75704679)

[2.2 MVC特点 5](#_Toc75704680)

[三、实例应用开发 5](#_Toc75704681)

[3.1 前言 5](#_Toc75704682)

[3.2 技术与原理 5](#_Toc75704683)

[3.3 需求建模 6](#_Toc75704684)

[3.4 后端设计 7](#_Toc75704685)

[3.4.1 pojo层 7](#_Toc75704686)

[3.4.2 DAO层 11](#_Toc75704687)

[3.4.3 Service层 11](#_Toc75704688)

[3.4.4 Controller层 13](#_Toc75704689)

[3.5 前端设计 15](#_Toc75704690)

[3.6 项目效果 23](#_Toc75704691)

[3.6.1 登陆界面 23](#_Toc75704692)

[3.6.2 主界面 23](#_Toc75704693)

[四、课程设计感受 26](#_Toc75704694)

# 一、Spring框架架构

## 1.1 什么是框架？

框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象模块及模块实例间交互的方法；或者可以认为，框架是可被应用开发者定制的应用骨架。

框架是为解决一类问题而产生的产品。框架用户只需使用框架提供的类或函数即可实现所需功能。框架为系统提供了大环境，配置好了一系列的关系，使用户只需专注于核心业务功能的实现

## 1.2 Spring 框架结构

Spring Framework 是一个开源的Java／Java EE全功能栈（full-stack）的应用程序框架。Spring共由十多个模块组成。

核心容器包括Core、Beans、Context和Expression Language。Core模块包含Spring框架基本的核心工具类，Core模块是其它组件的基本核心。BeanFactory提供对工厂模式的实现。

Context模块：上下文构建于Core和Beans模块上，是一个配置文件，向Spring框架提供上下文信息，提供了国际化、资源绑定、事件传播和资源加载等支持。

Spring AOP：为基于Spring的应用程序中的对象提供了事务管理服务，直接将面向方面的编程功能集成到Spring。

Spring DAO：DAO是用于访问数据的对象。提供DAO层抽象可以很容易的构造模拟对象方便测试。Spring DAO对多个持久化技术提供支持，同时制定了通用的异常体系，实现业务层与具体的持久化技术解耦。

JDBC：包含了Spring对JDBC数据访问进行封装的所有类。

ORM: 对象-关系映射API，如JPA、JDO、Hibernate、iBatis等，提供了一个交互层。

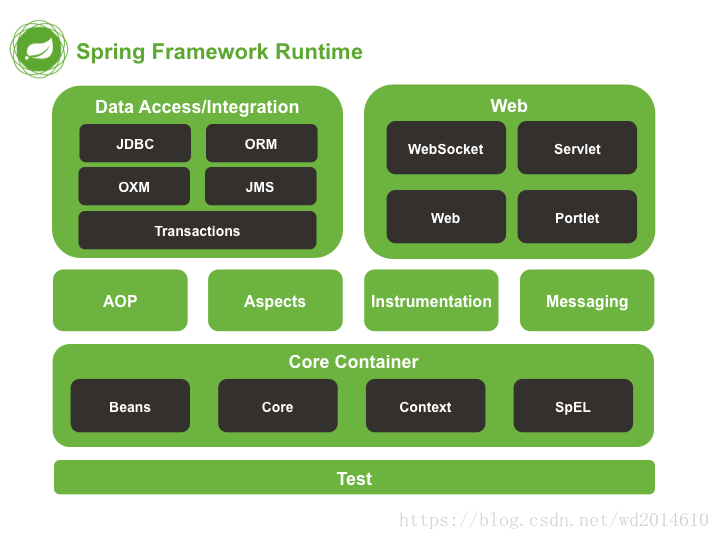
OXM: 提供了一个对Object/XML映射实现的抽象层。

JMS: 包含一些制造和消费消息的特性。

Transaction: 支持编程和声明性的事务管理，这些事务必须实现特定的接口，并且对所有的POJO都适用。

Spring Web模块：Web上下文基于应用程序上下文模块，为Web程序提供了上下文。

SpringMVC：是Spring框架提供的一个实现webMVC设计模式的轻量级框架，基于MVC框架。它是Spring框架的一部分，可以直接使用Spring提供的功能。



# 二、MVC

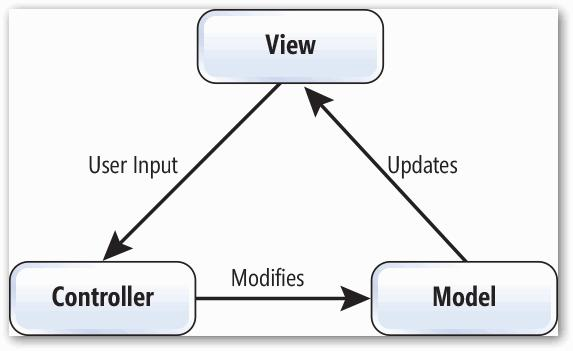
## 2.1 MVC结构

MVC是指Model View Controller，每个部分有他的职责。

Model：数据模型，是JavaBean对象，用来封装数据

View：视图，是指HTML等，用来展示数据给用户

Controller：控制器，用来接收用户请求，控制整个流程



## 2.2 MVC特点

MVC实现了显示模块与功能模块的分离，体现了解耦的思想，提高了程序的可维护性、可移植性、可拓展性和可重用性。

它的优点在于降低代码的耦合度，三个模块各司其职，需求变化时只需改某一模块的代码而不影响其它层。同时还利于程序开发的分工，实现前后端分离。它也有利于组件的重用。

不足之处在于其可能会增加系统结构的复杂性，且视图对模型的访问流程可能较长，导致效率降低。

# 三、实例应用开发

## 3.1 前言

为加深对设计模式和MVC框架的理解，我利用Spring Boot + Vue来实际开发一个简单的Web程序，其包含基于MVC的后端服务和基于Vue的前端界面，实现前后端分离解耦。

我所选择的主题是学生培养计划管理系统，系统目标是用户可以登录系统查看个人的培养计划信息，了解自己课程的学习状况与进度，具备课程的增删查改基本的功能。

## 3.2 技术与原理

前端技术：Vue.js + ElementUI、axios

使用Vue构建前端用户界面，配合ElementUI可以快速、简单的完成界面开发。前端通过RESTful API与后端传递JSON数据，而axios模块可以实现反向代理，将前端请求通过前端服务器转发给后端。

后端技术：Spring Boot、Spring Date JPA

后端采用Spring Boot的解决方案，以三层架构：DAO +Service +Controller分工合作。DAO层与数据库直接交互，增删查改；Service层提供业务逻辑；Controller层负责数据交互，调用Service层返回数据，并接受前端发送的请求。

数据库：MySQL

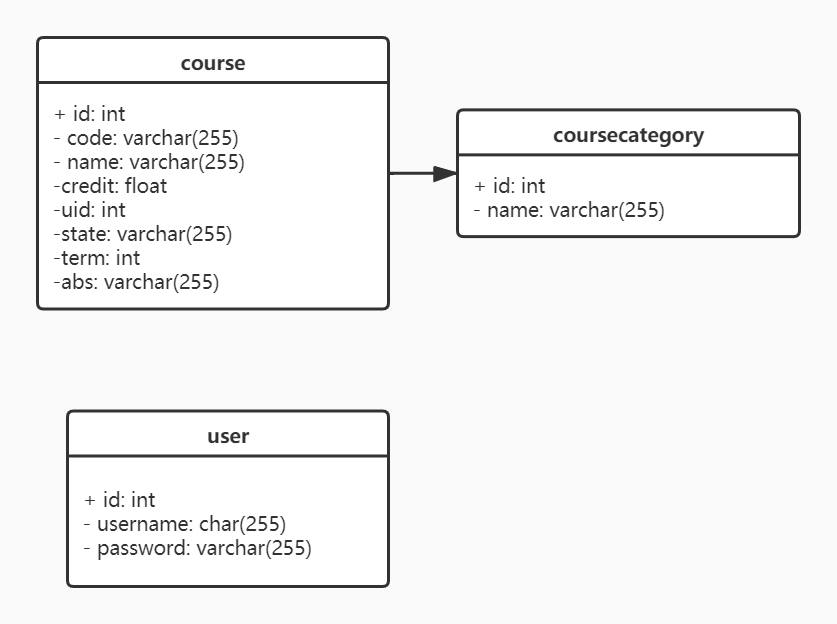
具体开发环境：

**Intellij IDEA 2021.1.2、Navicat 15、Chrome  91.0.4472.77**

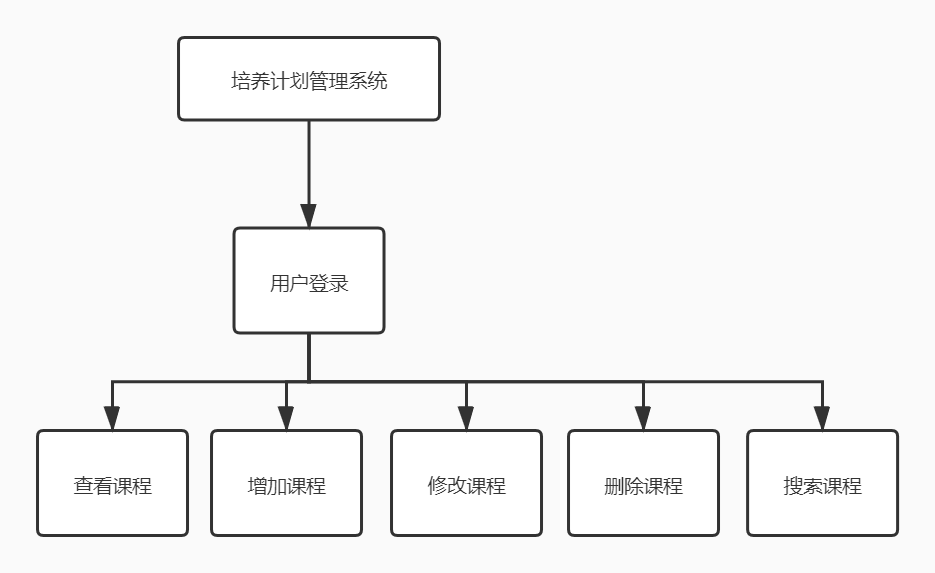
JAVA版本：1.8.0\_211 Vue版本：2.9.6 MySQL版本：5.7

## 3.3 需求建模

持久化数据表结构：



用户流程：



需求分析：

用户登录系统，简单的控制普通用户进入系统，阻止非法访问核心页面。

展示已有课程信息，修改课程信息，删除课程，添加课程，搜索课程

对课程进行分类

## 3.4 后端设计

分成四个部分：pojo（模型）、DAO（数据访问对象）、Service（业务逻辑）、Controller（控制器）。

模型层为DAO提供了基本模型对象，在DAO中实例化对象，生产数据访问对象对数据库进行实际操作，而Service层是具体业务逻辑的实现，实现方法即调用DAO中的函数。在Controller层，其接受前端的请求、数据，调用Service业务逻辑进行数据处理，然后返回结果。

### 3.4.1 pojo层

本层是基本模型类，包含User类、Course类和Category类。分别对应数据库中的三个表。

**User类：**

1. @Entity
2. @Table(name = "user")
3. @JsonIgnoreProperties({"handler","hibernateLazyInitializer"})
4. **public** **class** User {
5. @Id
6. @GeneratedValue(strategy = GenerationType.IDENTITY)
7. @Column(name = "id")
9. Integer id;
10. String username;
11. String password;
13. **public** Integer getId() {
14. **return** id;
15. }
17. **public** **void** setId(Integer id) {
18. **this**.id = id;
19. }
21. **public** String getUsername() {
22. **return** username;
23. }
25. **public** **void** setUsername(String username) {
26. **this**.username = username;
27. }
29. **public** String getPassword() {
30. **return** password;
31. }
33. **public** **void** setPassword(String password) {
34. **this**.password = password;
35. }
36. }

**Course类:**

1. @Entity
2. @Table(name = "course")
3. @JsonIgnoreProperties({"handler","hibernateLazyInitializer"})
4. **public** **class** Course {
5. @Id
6. @GeneratedValue(strategy = GenerationType.IDENTITY)
7. @Column(name = "id")
8. **int** id;
10. @ManyToOne
11. @JoinColumn(name="uid")
12. **private** Category category;
14. String code;
15. String name;
16. Float credit;
17. String state;
18. **int** term;
19. String abs;
21. **public** **int** getId() {
22. **return** id;
23. }
25. **public** **void** setId(**int** id) {
26. **this**.id = id;
27. }
29. **public** Category getCategory() {
30. **return** category;
31. }
33. **public** **void** setCategory(Category category) {
34. **this**.category = category;
35. }
37. **public** String getCode() {
38. **return** code;
39. }
41. **public** **void** setCode(String code) {
42. **this**.code = code;
43. }
45. **public** String getName() {
46. **return** name;
47. }
49. **public** **void** setName(String name) {
50. **this**.name = name;
51. }
53. **public** Float getCredit() {
54. **return** credit;
55. }
57. **public** **void** setCredit(Float credit) {
58. **this**.credit = credit;
59. }
61. **public** **int** getTerm() {
62. **return** term;
63. }
65. **public** **void** setTerm(**int** term) {
66. **this**.term = term;
67. }
69. **public** String getState() {
70. **return** state;
71. }
73. **public** **void** setState(String state) {
74. **this**.state = state;
75. }
77. **public** String getAbs() {
78. **return** abs;
79. }
81. **public** **void** setAbs(String abs) {
82. **this**.abs = abs;
83. }
84. }

Category类：

1. @Entity
2. @Table(name = "coursecategory")
3. @JsonIgnoreProperties({"handler","hibernateLazyInitializer"})
5. **public** **class** Category {
6. @Id
7. @GeneratedValue(strategy = GenerationType.IDENTITY)
8. @Column(name = "id")
9. **int** id;
11. String name;
13. **public** **int** getId() {
14. **return** id;
15. }
17. **public** **void** setId(**int** id) {
18. **this**.id = id;
19. }
21. **public** String getName() {
22. **return** name;
23. }
25. **public** **void** setName(String name) {
26. **this**.name = name;
27. }
28. }

### 3.4.2 DAO层

本层负责与数据库直接交互，定义增删查改操作，包括UserDAO接口、CourseDAO接口和CategoryDAO接口。

**UserDAO接口:**

1. **public** **interface** UserDAO **extends** JpaRepository<User,Integer>{
2. User findByUsername(String username);
4. User getByUsernameAndPassword(String username,String password);
5. //使用 JPA，无需手动构建 SQL 语句，而只需要按照规范提供方法的名字即可实现对数据库的增删改查。
6. }

**CourseDAO接口:**

1. **public** **interface** CourseDAO **extends** JpaRepository<Course,Integer> {
2. List<Course> findAllByCategory(Category category);
3. List<Course> findAllByNameLikeOrCodeLike(String keyword1, String keyword2);
5. }

**CategoryDAO接口：**

1. **public** **interface** CategoryDAO **extends** JpaRepository<Category, Integer> {
2. }

### 3.4.3 Service层

Service层是负责具体的业务逻辑，此处编写调用各种方法对从DAO中取得的数据操作，包括UserService类、CourseService类和CategoryService类。

**UserService类：**

1. @Service
2. **public** **class** UserService {
3. @Autowired
4. UserDAO userDAO;
6. **public** **boolean** isExist(String username) {
7. User user = getByName(username);
8. **return** **null**!=user;
9. }
11. **public** User getByName(String username) {
12. **return** userDAO.findByUsername(username);
13. }
15. **public** User get(String username, String password) {
16. **return** userDAO.getByUsernameAndPassword(username,password);
17. }
19. **public** **void** add(User user) {
20. userDAO.save(user);
21. }
23. }

**CourseService类:**

1. @Service
2. **public** **class** CourseService {
4. @Autowired
5. CourseDAO courseDAO;
6. @Autowired
7. CategoryService categoryService;
9. **public** List<Course> list() {
10. Sort sort = **new** Sort(Sort.Direction.DESC,"id");
11. **return** courseDAO.findAll(sort);
12. }
14. **public** List<Course> Search(String keywords) {
15. **return** courseDAO.findAllByNameLikeOrCodeLike('%' + keywords + '%','%'+ keywords + '%');
16. }
17. **public** **void** addOrUpdate(Course course) {
18. courseDAO.save(course);
19. //这里 save() 方法的作用是，当主键存在时更新数据，当主键不存在时插入数据。
20. }
22. **public** **void** deleteById(**int** id) {
23. courseDAO.deleteById(id);
24. }
26. **public** List<Course> listByCategory(**int** cid) {
27. Category category = categoryService.get(cid);
28. **return** courseDAO.findAllByCategory(category);
29. }
30. }

**CategoryService类：**

1. @Service
2. **public** **class** CategoryService {
3. @Autowired
4. CategoryDAO categoryDAO;
6. **public** List<Category> list() {
7. Sort sort = **new** Sort(Sort.Direction.DESC,"id");
8. **return** categoryDAO.findAll(sort);
9. }
11. **public** Category get(**int** id) {
12. Category c = categoryDAO.findById(id).orElse(**null**);
13. **return** c;
14. }
15. }

### 3.4.4 Controller层

本层负责数据交互，接受前端的数据，调用Service中的函数处理并返回结果。包括LibraryController类和LoginController类。

**LoginController类：**

1. @Controller
2. **public** **class** LoginController {
4. @Autowired
5. UserService userService;
7. @CrossOrigin
8. @PostMapping(value = "api/login")
9. @ResponseBody
10. **public** Result login(@RequestBody User requestUser){
11. String username = requestUser.getUsername();
13. User user = userService.get(username,requestUser.getPassword());
14. **if** (**null** == user)
15. **return** **new** Result(400);
16. **else**
17. **return** **new** Result(200);
18. }
19. }

**LibraryController类：**

1. @RestController
2. **public** **class** LibraryController {
3. @Autowired
4. CourseService courseService;
6. @CrossOrigin
7. @GetMapping("/api/courses")
8. **public** List<Course> list() **throws** Exception {
9. **return** courseService.list();
10. }
12. @CrossOrigin
13. @PostMapping ("/api/courses")
14. **public** Course addOrUpdate(@RequestBody Course course) **throws** Exception {
15. courseService.addOrUpdate(course);
16. System.out.println("AddOrUpdating");
17. **return** course;
18. }
20. @CrossOrigin
21. @PostMapping("/api/delete")
22. **public** **void** delete(@RequestBody Course course) **throws** Exception {
23. courseService.deleteById(course.getId());
24. }
26. @CrossOrigin
27. @GetMapping("/api/categories/{cid}/courses")
28. **public** List<Course> listByCategory(@PathVariable("cid") **int** cid) **throws** Exception {
29. **if** (0 != cid) {
30. **return** courseService.listByCategory(cid);
31. }
32. **else** {
33. **return** list();
34. }
35. }
37. @CrossOrigin
38. @GetMapping("/api/search")
39. **public** List<Course> searchResult(@RequestParam("keywords") String keywords) {
40. // 关键词为空时查询出所有课程
41. System.out.println("关键字："+keywords);
42. **if** ("".equals(keywords)) {
43. **return** courseService.list();
44. } **else** {
45. **return** courseService.Search(keywords);
46. }
47. }
49. }

## 3.5 前端设计

前端分为登录页面和主页面，在登陆页面中输入正确的帐号密码后，跳转到主页面。其中实现了登录拦截器避免未登录直接进入主页面。

主页面左侧显示课程分类，中间主体是课程的详细信息表，上方还有搜索栏。数据下方有加号按键可以添加，最下方有页数索引。

在main.js中设置反向代理，将请求发送到指定域。同时为了解决跨域问题，在config\index.js中进行配置

**Main.js：**

1. import Vue from 'vue'
2. import App from './App'
3. import router from './router'
4. import ElementUI from 'element-ui'
5. import store from './store'
6. import 'element-ui/lib/theme-chalk/index.css'
7. // 设置反向代理，前端请求默认发送到 http://localhost:8443/api
8. var axios = require('axios')
9. axios.defaults.baseURL = 'http://localhost:8443/api'
10. // 全局注册，之后可在其他组件中通过 this.$axios 发送数据
11. Vue.prototype.$axiosaxios = axios
12. Vue.config.productionTip = false
14. Vue.use(ElementUI)
15. // 使用Element
17. router.beforeEach((to, from, next) =**>** {
18. if (to.meta.requireAuth) {
19. // eslint-disable-next-line no-undef
20. if (store.state.user.username) {
21. next()
22. } else {
23. next({
24. path: 'login',
25. query: {redirect: to.fullPath}
26. })
27. }
28. } else {
29. next()
30. }
31. })
33. /\* eslint-disable no-new \*/
34. new Vue({
35. el: '#app',
36. render: h =**>** h(App),
37. router,
38. store,
39. components: { App },
40. template: '**<App/>**'
41. })

**Config\index.js中在对应地方修改：**

1. proxyTable: {
2. '/api': {
3. target: 'http://localhost:8443',
4. changeOrigin: true,
5. pathRewrite: {
6. '^/api': ''
7. }
8. }
9. },

**登录界面组件：**

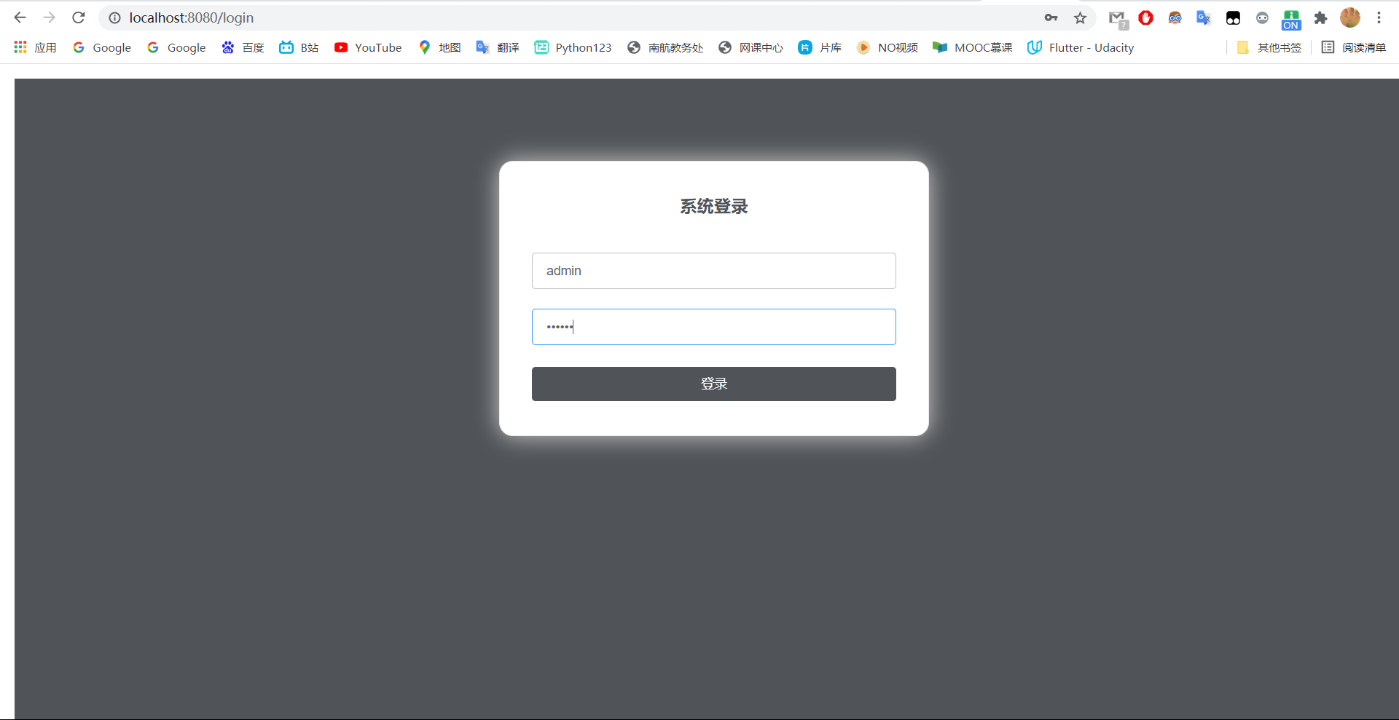
1. **<template>**
2. **<body** id="poster"**>**
3. **<el-form** class="login-container" label-position="left"
4. label-width="0px"**>**
5. **<h3** class="login\_title"**>**系统登录**</h3>**
6. **<el-form-item>**
7. **<el-input** type="text" v-model="loginForm.username"
8. auto-complete="off" placeholder="账号"**></el-input>**
9. **</el-form-item>**
10. **<el-form-item>**
11. **<el-input** type="password" v-model="loginForm.password"
12. auto-complete="off" placeholder="密码"**></el-input>**
13. **</el-form-item>**
14. **<el-form-item** style="width: 100%"**>**
15. **<el-button** type="primary" style="width: 100%;background: #505458;border: none" v-on:click="login"**>**登录**</el-button>**
16. **</el-form-item>**
17. **</el-form>**
18. **</body>**
19. **</template>**
21. **<script>**
23. export default {
24. name: 'Login',
25. data () {
26. return {
27. loginForm: {
28. username: '',
29. password: ''
30. },
31. responseResult: []
32. }
33. },
34. methods: {
35. login () {
36. var \_this = this
37. console.log(this.$store.state)
38. this.$axios
39. .post('/login', {
40. username: this.loginForm.username,
41. password: this.loginForm.password
42. })
43. .then(successResponse =**>** {
44. if (successResponse.data.code === 200) {
45. // var data = this.loginForm
46. \_this.$store.commit('login', \_this.loginForm)
47. var path = this.$route.query.redirect
48. this.$router.replace({path: path === '/' || path === undefined ? '/index' : path})
49. }
50. })
51. .catch(failResponse =**>** {
52. })
53. }
55. }
56. }
57. **</script>**
59. **<style>**
60. .login-container{
61. border-radius: 15px;
62. background-clip: padding-box;
63. margin: 90px auto;
64. width: 400px;
65. padding: 35px 35px 15px 35px;
66. background: #fff;
67. border: 1px solid #eaeaea;
68. box-shadow: 0 0 25px #cac6c6;
69. }
70. .login\_title{
71. margin: 0px auto 40px auto;
72. text-align: center;
73. color: #505458;
74. }
75. #poster {
76. background: #505458;
77. background-position: center;
78. height: 100%;
79. width: 100%;
80. background-size: cover;
81. position: fixed;
82. }
83. **</style>**

**主界面课表组件：**

1. **<template>**
2. **<div>**
3. **<el-row** style="height: 840px;width: auto"**>**
4. **<search-bar** @onSearch="searchResult" ref="searchBar"**></search-bar>**
5. **<el-table**
6. :data="courses"
7. style="width: 100%"
8. fit**>**
9. **<el-table-column**
10. label="课程号"
11. width="180"**>**
12. **<template** slot-scope="scope"**>**
13. **<span** style="margin-left: 10px"**>**{{ scope.row.code }}**</span>**
14. **</template>**
15. **</el-table-column>**
16. **<el-table-column**
17. label="课程名"
18. width="250"**>**
19. **<template** slot-scope="scope"**>**
20. **<el-popover** trigger="hover" placement="top" open-delay="500"**>**
21. **<p>**备注: {{ scope.row.abs }}**</p>**
22. **<div** slot="reference" class="name-wrapper"**>**
23. **<el-tag** size="medium"**>**{{ scope.row.name }}**</el-tag>**
24. **</div>**
25. **</el-popover>**
26. **</template>**
27. **</el-table-column>**
28. **<el-table-column**
29. label="课程类别"
30. width="150"**>**
31. **<template** slot-scope="scope"**>**
32. **<span** style="margin-left: 10px"**>**{{ scope.row.category.name }}**</span>**
33. **</template>**
34. **</el-table-column>**
35. **<el-table-column**
36. label="学分"
37. width="100"**>**
38. **<template** slot-scope="scope"**>**
39. **<span** style="margin-left: 10px"**>**{{ scope.row.credit }}**</span>**
40. **</template>**
41. **</el-table-column>**
42. **<el-table-column**
43. label="修读学期"
44. width="100"**>**
45. **<template** slot-scope="scope"**>**
46. **<span** style="margin-left: 10px"**>**{{ scope.row.term }}**</span>**
47. **</template>**
48. **</el-table-column>**
49. **<el-table-column**
50. label="修读状态"
51. width="100"**>**
52. **<template** slot-scope="scope"**>**
53. **<span** style="margin-left: 10px"**>**{{ scope.row.state }}**</span>**
54. **</template>**
55. **</el-table-column>**
56. **<el-table-column** label="操作" fixed="right"**>**
57. **<template** slot-scope="scope"**>**
58. **<el-button**
59. size="mini"
60. @click="editCourse(scope.row)"**>**编辑**</el-button>**
61. **<el-button**
62. size="mini"
63. type="danger"
64. @click="deleteCourse(scope.row.id)"**>**删除**</el-button>**
65. **</template>**
66. **</el-table-column>**
67. **</el-table>**
68. **<edit-form** @onSubmit="loadCourses()" ref="edit"**></edit-form>**
69. **</el-row>**
70. **<el-row>**
71. **<el-pagination**
72. @current-change="handleCurrentChange"
73. :current-page="currentPage"
74. :page-size="pagesize"
75. :total="courses.length"**>**
76. **</el-pagination>**
77. **</el-row>**
78. **</div>**
79. **</template>**
81. **<script>**
82. import EditForm from './EditForm'
83. import SearchBar from './SearchBar'
84. export default {
85. name: 'Courses',
86. components: {EditForm, SearchBar},
87. data () {
88. return {
89. courses: [],
90. currentPage: 1,
91. pagesize: 17
92. }
93. },
94. mounted: function () {
95. this.loadCourses()
96. // 钩子函数，打开页面时触发代码进行发送请求并渲染本页面
97. },
98. methods: {
99. loadCourses () {
100. var \_this = this
101. this.$axios.get('/courses').then(resp =**>** {
102. if (resp && resp.status === 200) {
103. \_this.courses = resp.data
104. }
105. })
106. // 利用axios发送一个get请求，后端返回后把data更新为后端传来的数据
107. },
108. handleCurrentChange: function (currentPage) {
109. this.currentPage = currentPage
110. console.log(this.currentPage)
111. },
112. searchResult () {
113. var \_this = this
114. this.$axios
115. .get('/search?keywords=' + this.$refs.searchBar.keywords, {
116. }).then(resp =**>** {
117. if (resp && resp.status === 200) {
118. \_this.courses = resp.data
119. }
120. })
121. },
122. deleteCourse (id) {
123. this.$confirm('此操作将永久删除该课程, 是否继续?', '提示', {
124. confirmButtonText: '确定',
125. cancelButtonText: '取消',
126. type: 'warning'
127. }).then(() =**>** {
128. this.$axios
129. .post('/delete', {id: id}).then(resp =**>** {
130. if (resp && resp.status === 200) {
131. this.loadCourses()
132. }
133. })
134. }
135. ).catch(() =**>** {
136. this.$message({
137. type: 'info',
138. message: '已取消删除'
139. })
140. })
141. // alert(id)
142. },
143. editCourse (item) {
144. this.$refs.edit.dialogFormVisible = true
145. this.$refs.edit.form = {
146. id: item.id,
147. code: item.code,
148. name: item.name,
149. credit: item.credit,
150. state: item.state,
151. term: item.term,
152. abs: item.abs,
153. category: {
154. id: item.category.id.toString(),
155. name: item.category.name
156. }
157. }
158. }
159. }
161. }
162. **</script>**
164. **<style** scoped**>**
166. img {
167. width: 115px;
168. height: 172px;
169. /\*margin: 0 auto;\*/
170. }
172. a {
173. text-decoration: none;
174. }
176. a:link, a:visited, a:focus {
177. color: #3377aa;
178. }
179. **</style>**

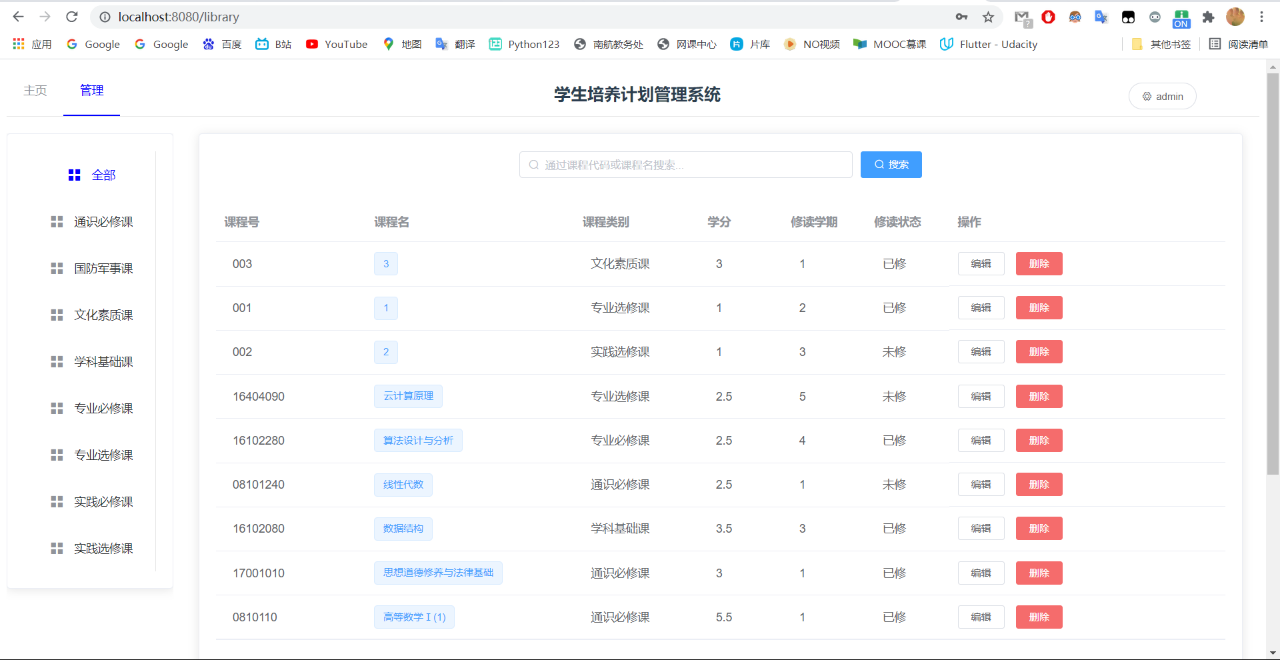
## 3.6 项目效果

### 3.6.1 登陆界面

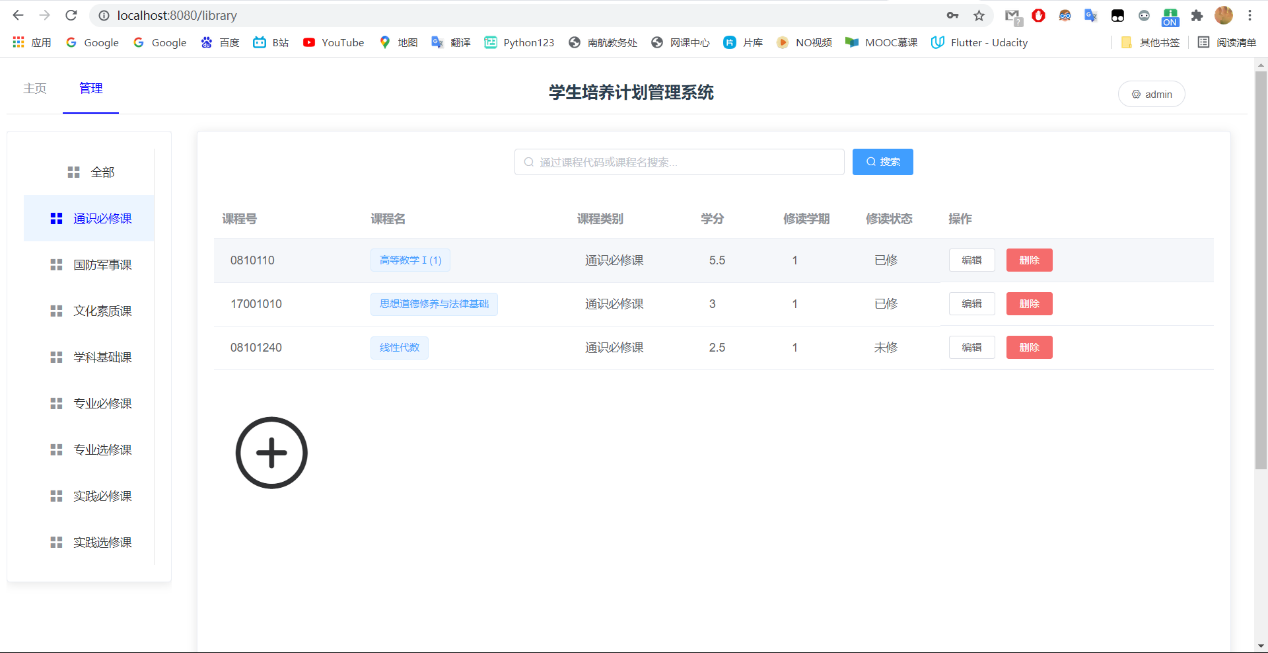


### 3.6.2 主界面

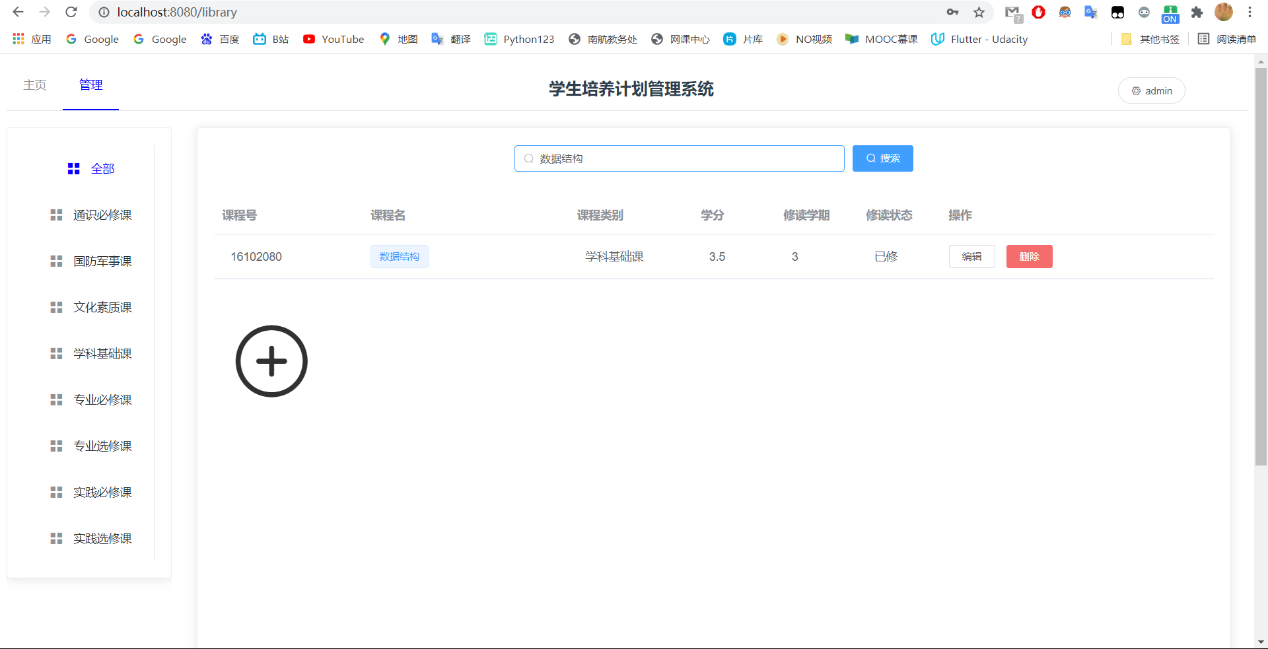
主页面：



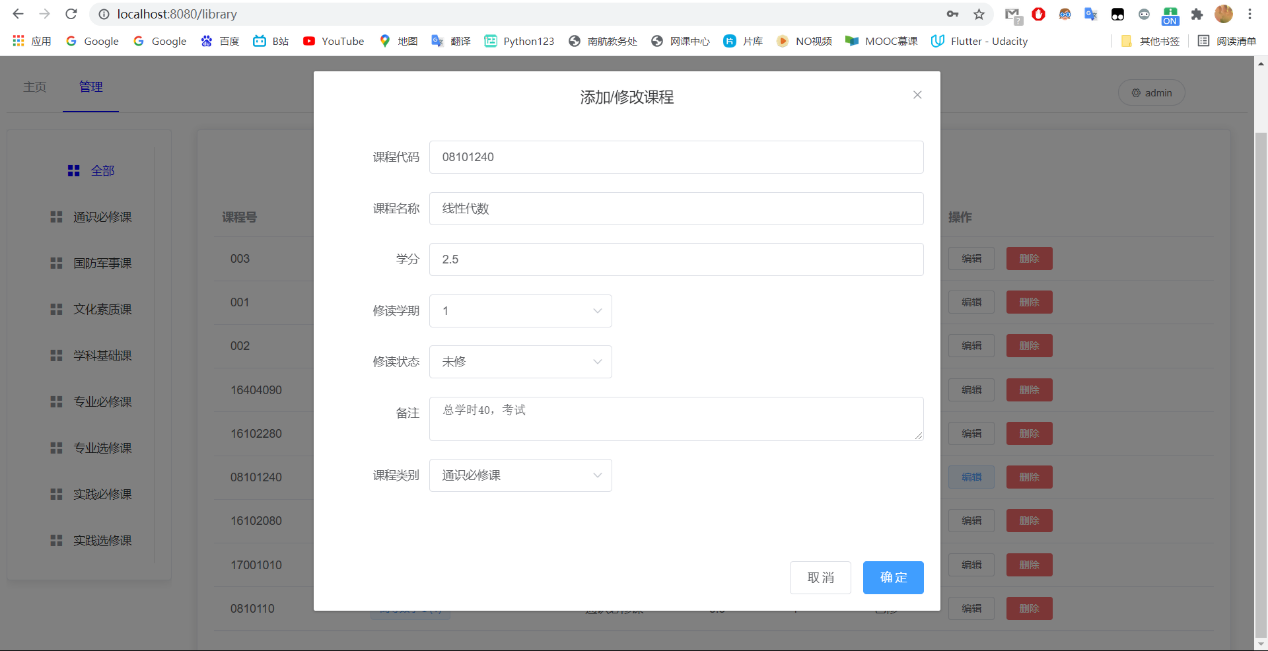
分类：



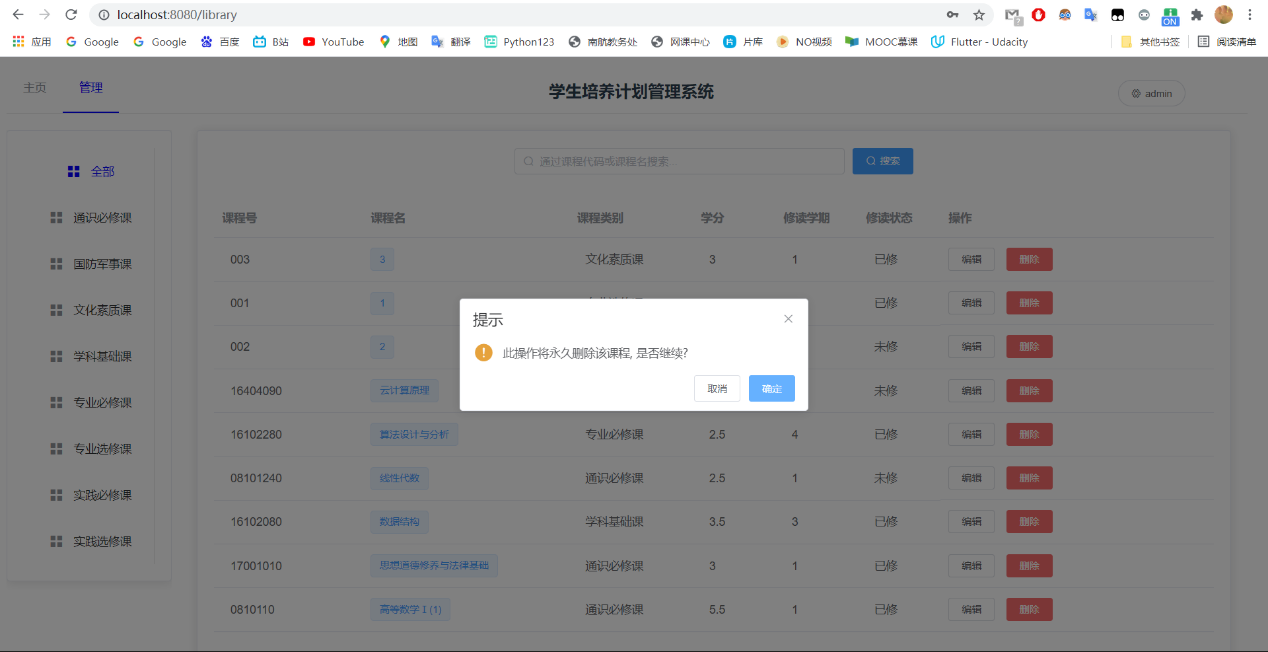
搜索：



编辑/增加：



删除：



# 四、课程设计感受

本次课程设计完成了一个简单了web系统，从前端到后端，再到数据库，亲手搭建起这一个简单的系统。

虽说老师提早许久就布置下了课设任务，但正如老师所料，我还是放在了最后几周来做课设，伴着复习和其它课设、实验，忙里抽空来做本课设。由于还从未涉及了解过前端的实现，因此在简单的完成了助教ppt中的demo后仍一头雾水。于是转向网络，找一些相关项目，来先学习一下前后端分离开发的整个流程。在看了许多文档和项目后，终于准备开始项目的工作。

在项目后端构建中，我不得不感叹整个MVC还挺巧妙的，三个角色分离，各司其职完成系统任务，搭建起整个系统时又很清晰的展示各层关系。然后我又想感叹一下框架真好用，尤其是Spring Boot，无需配置大量信息，只需一个引用，轻松构建起系统架构。在前端构建时，先去简单学习了Vue的方案，但项目过程中还是遇到许多问题，只得不断地查询网络来找解决办法。在克服种种bug之后，所搭建的页面出现在眼前，那是说不出来的高兴与成就感，虽然挺简陋。后来了解到了Element UI的插件，它直接内置了一下风格统一又好看的组件，页面美观直接提升好几个层次。而数据库，只简单的学了基本操作，然后用Navicat操作数据库。尴尬的是经常在数据库中打错字，导致在后端操作时报错，找半天发现打错一个字母、大小写混乱。

我选的web系统主题是培养计划管理，选这个的原因是想自己建一个管理系统，来管理自己当前的课程学习进度、学分。目前只简单的完成了课程的增删查改和分类，等暑假有空就把学分管理，以及提供更多有用的信息放在主页显示（主界面中的主页导航页暂未实现）。同时还可以拓展一下用户管理，实现用户权限分级，完成管理员后台的页面等。

经历了这次课设任务，真的学到了很多，尤其是前后端的分离，以及后端中MVC设计模式的实践，让我对课程知识有了更深一步的理解。最后，感谢老师与助教的教学与指导。