数据库课程设计

王少文 19307130251
谢子陶 19307130037

2021.6.11

摘要

本次课程实验我们设计编写了一个智能自习管理小程序 GoStudy，集成了一体式学习管理与自习推荐系统。前端依托于微信小程序，使用了部分 Vant-Weapp 组件库中的前端组件进行开发；通过部署 SSL 证书和用户 token 机制实现前后台交互的数据加密，保证用户数据安全性；后端采用 Go 语言开发，通过 Nginx 反向代理，完成服务转发和负载均衡，通过 Gin 搭建 API Server 提供前后端交互接口，插入数据库；在数据库层面，我们采用了 Redis 和 PostgreSql 的两级数据库部署模式，通过 Redis 加速数据索引，以应对高并发业务场景。在部署上，我们采用了容器 (Docker) 技术，实现了不依赖操作系统的灵活多机部署。同时我们通过 Django Admin 搭建了功能强大的数据库管理页面，支持对于数据库中各表数据的各类操作以及可视化数据显示。

关键词： 微信小程序; 容器 (Docker); PostgreSQL 数据库; Redis 数据库; Gin 框架; Django 框架

目录

1 实验题目 **4**
1.1 项目名字 4
1.2 项目背景 4
1.3 功能介绍 4

2 开发环境 **4**
2.1 后端开发环境 4
2.1.1 Docker 5
2.1.2 PostgreSQL 5
2.1.3 Apiservice 5
2.1.4 Nginx 5
2.1.5 Redis 5
2.1.6 Admin 6
2.2 前端开发环境 6
2.2.1 WeApp 6
2.2.2 Vant-WeApp 前端组件 6

3 数据库设计 6
3.1 表设计 . . . 6
3.2 数据库指标实现 . . . 7
3.2.1 约束 . . . 7
3.2.2 储存过程、触发器、视图和函数 . . . 7
3.3 数据库固化、备份和还原 . . . 7

4 系统设计 7
4.1 后端系统设计 . . . 7
4.1.1 爬虫模块设计 . . . 8
4.1.2 数据库通信模块设计 . . . 9
4.1.3 API 服务模块设计 . . . 10
4.1.4 管理服务模块设计 . . . 11
4.2 前端系统设计 . . . 12
4.2.1 整体风格设计 . . . 12
4.2.2 响应式刷新 . . . 12
4.2.3 页面生命周期 . . . 13
4.2.4 数据缓存机制 . . . 14
4.2.5 主页面 Index Page . . . 14
4.2.6 课程查询页面 ClassDB Page . . . 15
4.2.7 课表页面 Curriculum Page . . . 16
4.2.8 用户页面 User Page . . . 16
4.2.9 用户自习时长统计逻辑 . . . 17
4.2.10 前端页面通信 . . . 18

5 特色和创新点 18
5.1 后端特色和创新点 . . . 18
5.1.1 原生支持多服务器架构 . . . 18
5.1.2 使用 Redis 作为缓存 . . . 18
5.1.3 高安全性 . . . 18
5.1.4 易于部署 . . . 19
5.2 前端特色和创新点 . . . 19
5.2.1 智能教室推荐系统 . . . 19
5.2.2 自习时长管理系统 . . . 20
5.2.3 自习称号系统 . . . 20
5.2.4 课表管理系统 . . . 20
5.2.5 教室课程查询 . . . 20

6 实验分工 20

7 提交文件说明 20
7.1 后端提交文件说明 . . . 20
7.1.1 各文件/文件夹说明 . . . 20
7.1.2 运行方法 . . . 21

7.2 前端提交文件说明 21
7.2.1 页面层级组织 21
7.2.2 运行方法 21

8 实验总结 **21**

9 致谢 **22**

1 实验题目

1.1 项目名字

GoStudy(轻量级学习集成 APP)

图 1: GoStudy LOGO

1.2 项目背景

复旦提供了一个内网的 IP 地址 (http://10.64.130.6/) 用于方便教学楼管理员处理临借教室等情况，但该网站只能校内访问，前端页面没有对手机端优化，操作不便，也没有智能选取等功能，每次必须手动填写时间、教学楼、楼层等信息，十分不便。而学校的微信小程序空教室查询已经失效数天，恢复情况未知；且该小程序中的信息很简单，只能看到教室是否为空，而不能看到该教室具体在上什么课，对旁听的同学不方便。此外，学校的查课仍旧十分不方便，只能在选课阶段进行查询，对想要旁听的同学不甚友好。因此，一款用户友好的集成 APP 就显得十分必要。

1.3 功能介绍

我们将针对项目背景中的痛点实现以下功能:

1. 查询各教学楼的空闲教室，支持根据不同的筛选条件进行查询，也支持通过地理位置 + 用户偏好[1]进行智能推荐。
2. 查询各教室的课表；根据课程编号和课程名进行模糊搜索课程。
3. 将查询到的课程添加入自己的选课；也支持自定义添加课程。
4. 统计用户的自习时间，给出相应的鼓励；[2]

2 开发环境

2.1 后端开发环境

后端开发完全使用 Docker 化，与操作系统无关，最终部署于腾讯云轻量级服务器上 (Ubuntu20.04)，下面为我们使用的 Docker 及容器版本。[3]

[1] 目前用户偏好的算法还十分不完善，使用了启发式先天赋值算法，将在下一版本调整为使用用户频次或机器学习算法

[2] 在未来可能加入好友列表比一比功能

[3] 若需要正确运行爬虫模块，还需要在学校校园网覆盖范围内，或者在服务器上使用 easyconnect 配置全局 VPN 连接到学校

2.1.1 Docker

Docker 软件版本: 20.10.6

2.1.2 PostgreSQL

Docker 映像: Postgres:13

2.1.3 Apiservice

基于 Docker 映像：golang:1.16
使用 Go mod 安装下列包：

1. github.com/PuerkitoBio/goquery v1.6.1 // indirect
2. github.com/antchfx/htmlquery v1.2.3 // indirect
3. github.com/antchfx/xmlquery v1.3.6 // indirect
4. github.com/dgrijalva/jwt-go v3.2.0+incompatible
5. github.com/gin-gonic/gin v1.7.2
6. github.com/go-pg/pg/extra/pgdebug/v10 v10.9.3
7. github.com/go-pg/pg/v10 v10.9.3
8. github.com/go-redis/redis/v8 v8.10.0
9. github.com/gobwas/glob v0.2.3 // indirect
10. github.com/gocolly/colly v1.2.0
11. github.com/golang/protobuf v1.5.2 // indirect
12. github.com/kennygrant/sanitize v1.2.4 // indirect
13. github.com/saintfish/chardet v0.0.0-20120816061221-3af4cd4741ca // indirect
14. github.com/temoto/robotstxt v1.1.2 // indirect
15. golang.org/x/net v0.0.0-20210503060351-7fd8e65b6420
16. golang.org/x/sys v0.0.0-20210514084401-e8d321eab015 // indirect

2.1.4 Nginx

Docker 映像: Nginx:1.21.0

2.1.5 Redis

Docker 映像: Redis:6

2.1.6 Admin

基于 Docker 映像：python:3
使用 pip 安装下列包:

1. Django>=3.0,<4.0
2. psycopg2-binary>=2.8
3. django-simpleui
4. gunicorn>=20.1.0

2.2 前端开发环境

2.2.1 WeApp

微信小程序基础调试库版本: 2.17.0 主要测试机型: iPhoneX, iPhone6/7/8, Nexus5, iPad Pro

2.2.2 Vant-WeApp 前端组件

Vant-WeApp 前端组件版本: 1.6.9 (经过修改)

3 数据库设计

3.1 表设计

本数据库共含 9 张表，具体表的字段与数据类型在下图中可见，表内属性顾名思义不再详述。

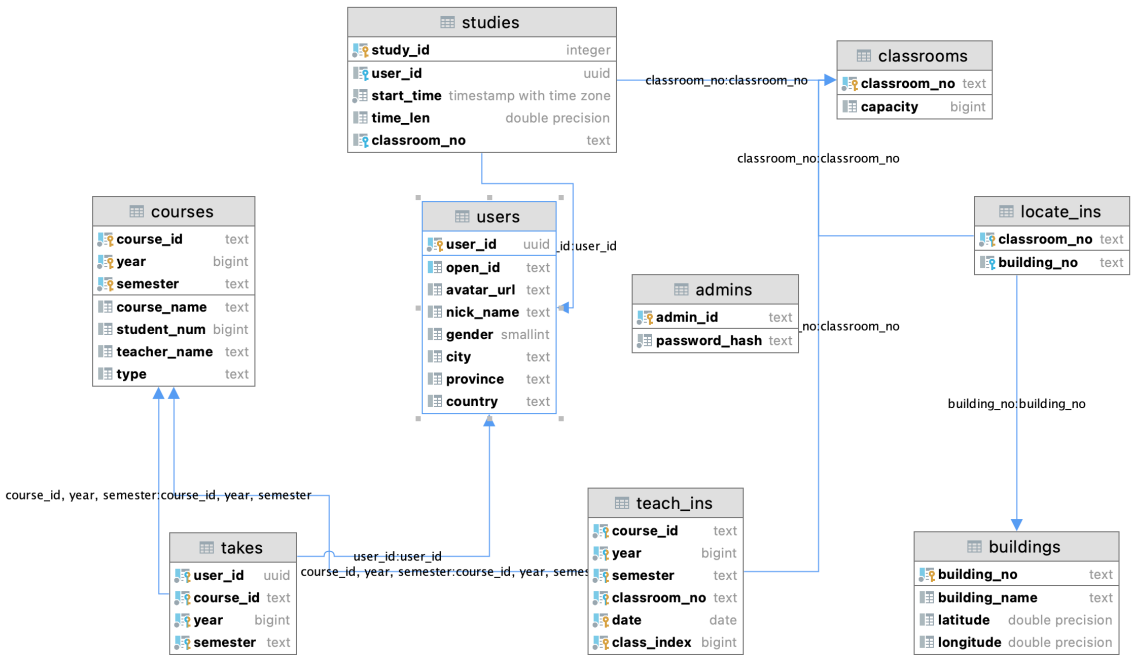

图 2: 数据库设计图

1. users 表储存了用户信息
2. admin 表储存了管理员的信息
3. studies 表储存了用户每次自习的记录
4. classrooms 表储存了学校的教室
5. locate_ins 表储存了学校的教室对应的教学楼
6. buildings 表储存教学楼的相关信息
7. courses 表储存了课程相关信息
8. teach_ins 表储存了课堂相关信息
9. takes 表储存了用户选课的信息

3.2 数据库指标实现

3.2.1 约束

除了图中所列出的外键和主键约束外，我们还加入了如下三个约束，来保证我们表数据的完整性

```
teachin表约束chk_index: class_index>=1 and class_index<=14
users表约束chk_gender: gender>=0 and gender<=2
courses表约束chk_semester:  semester IN ($$Spring$$,$$Autumn$$,$$Summer$$)
```

3.2.2 储存过程、触发器、视图和函数

本设计在 backend/script/db_init.sql 下实现了该指标[4]

3.3 数据库固化、备份和还原

我们的数据库是运行在 docker 容器内的，但通过文件映射的方式固化在了宿主机的文件系统内，同时在 backend/script 中实现了备份和恢复的脚本

4 系统设计

4.1 后端系统设计

后端架构如下[5][6]：我们使用了 docker 来隔离各个服务，使其尽可能的单元化。

1. Postgres Server 为主 (网络) 数据库;
2. Redis Server 为主 (缓存) 数据库[7];
3. API Servergroup 为实现核心 API 的服务器组;
4. APP Server 为实现管理页面的 Web 服务器;
5. Nginx 为负载均衡、反向代理服务器;

[4]但值得注意的是，现代的 web 应用更多不再使用数据库内部的函数，而在外部的代码中实现业务逻辑
[5]目前最右侧的下游应用中，只实现了 Web 与 Mini APP 部分，APP 部分暂未实现，仅作为示意
[6]并不是所有的数据都会经过 redis，只有被访问最频繁的 API 才会在 Redis 中缓存，图中仅为示意
[7]目前只有单个 Redis Server，但可以变为 Redis Server Group

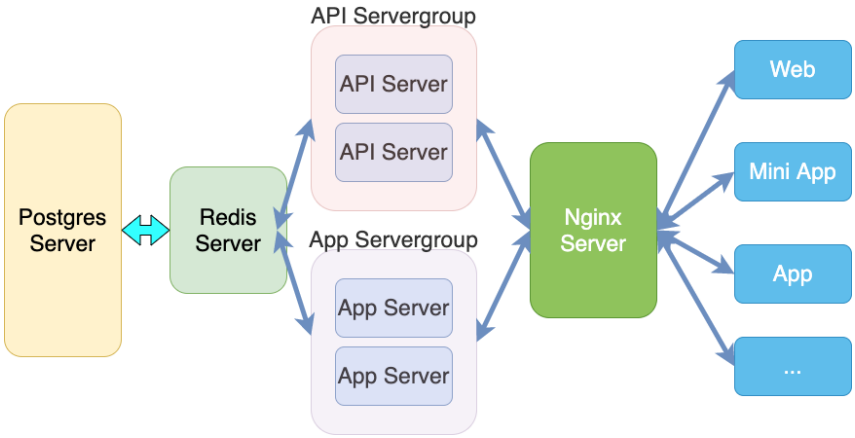

图 3: GoStudy 后端架构图

4.1.1 爬虫模块设计

爬虫模块使用 Go 爬虫引擎 colly，逐层解析学校查课网站中的 HTML 代码。学校的查课网站上，课程是以表格的形式展示的，因此可以进行逐行逐列的遍历。核心代码如下:

```
c.OnHTML("table[id='statusTable_0']", func(table *colly.HTMLElement) {
  table.ForEach("tr", func(row int, line *colly.HTMLElement) {
    //在此处运行解析教室的代码
    line.ForEach("td", func(idx int, class *colly.HTMLElement) {
    //在此处运行解析课程的代码
    }
  }
}
```

但按此方式遍历速度较慢，因此我们在爬虫中使用 GoRuntine 加入了并发爬取，代码如下:

```
func (con *Connection) CrawlClasses(dateArr []time.Time) error {
  con.rdb.FlushDB(con.ctx)
  log.Println("Flushing redis database")
  errChan := make(chan error)
  for _, date := range dateArr {
    date := date
    go func() {
      err := con.CrawlClassesDate(date)
      if err != nil {
        log.Println("Error in CrawlClasses", err)
      }
      errChan <- err
    }()
  }
  length := len(dateArr)
  for i := 0; i < length; i++ {
    err := <-errChan
    if err != nil {
      return err
    }
  }
```

```
  return nil
}
```

4.1.2 数据库通信模块设计

Go 语言与 PostgreSQL 数据库的通信模块使用 go-pg 库，与 redis 数据库的通信使用 go-redis 库，在数据库通信中，我们将所有的操作都封装入 Connection，在库外只允许调用 Connection 的方法，实现了对数据库的抽象。

```
type Connection struct {
  db  *pg.DB
  rdb *redis.Client
  ctx context.Context
}
```

go-pg 库提供了 orm 方法，允许我们在定义表结构体时，通过反射制定由结构体生成的数据库的表的名字、类型、约束等信息，以 Study 表为例，我们指定了 StudyId 的类型为 PostgreSQL 的自增型数据 serial，并且指定其为主键。

```
type Study struct {
  StudyId     int     `pg:"type:serial,pk"`
  UserId      string  `pg:"type:uuid"`
  StartTime   time.Time `pg:",notnull"`
  TimeLen     float64 `pg:"default:0"` // The unit is second
  ClassroomNo string
}
```

但因为 go-pg 库的缺陷，我们不能直接建表时指定外键，因此我手写了外键的执行代码（和其他某些未提供功能的代码）

```
func (con *Connection) addForeignKey(tableName, fkColumns, parentTable,
    parentColumns string) error {
  constraintName := "fk_" + strings.ReplaceAll(fkColumns, ",", "_")
  _, err := con.db.Model().Exec(fmt.Sprintf("SELECT create_constraint_if_not_exists
    ('%s','%s','ALTER TABLE %s ADD CONSTRAINT %s FOREIGN KEY(%s) REFERENCES %s(%s)
    ;') ", tableName, constraintName, tableName, constraintName, fkColumns,
    parentTable, parentColumns))
  if err != nil {
    return err
  }
  return nil
}
```

对于 go-pg 库支持的操作，我们可以使用其模板引擎帮助我们动态生成 SQL，如在查找空教室时，我们的代码如下：

```
var classroomNo []string
  allClassRooms := db.Model().
    Column("classroom_no").
    Distinct().
    Table("classrooms").
    Join("NATURAL JOIN \"locate_ins\"").
```

```
    Where("building_no=?", building)
  usedClassRooms := db.Model((*TeachIn)(nil)).
    Column("classroom_no").
    Distinct().
    Join("NATURAL JOIN \"classrooms\" NATURAL JOIN \"locate_ins\"").
    Where("date=?", dateString).Where("class_index=?", classIndex).
    Where("building_no=?", building)
  err = db.Model().
    TableExpr("(?) AS res", allClassRooms.Except(usedClassRooms)).
    Order("classroom_no asc").
    Select(&classroomNo)
```

而 Redis 作为一个键值对数据库，操作较为简单，只需调用 Get, Set 方法就能将数据从数据库中取出和存入，但需要注意的是 Redis 中的数据结构较为单一，本次实验中为了方便考虑，所有的数据均采用 Json 格式的字符串在 redis 中一 string 类型储存。同样以缓存空教室数据为例，核心代码如下:

```
  jsonVal, err := json2.Marshal(classroomVal{ClassroomNo: classroomNo})
  if err != nil {
    log.Println(err)
  }
  err = rdb.Set(ctx, string(json), string(jsonVal), 0).Err()
```

4.1.3 API 服务模块设计

本项目构建了一个易于添加、修改的 API 模块。

handler 所有的 API url 都需要绑定一个 handler，来让 gin 引擎在接收到对这个 url 的请求后，将请求转发该 handler 上。我们在 handler 中分别按需解析出本次请求的参数、头和数据段，然后调用数据库中的操作进行处理。以开始学习的 handler 为例，我们首先利用 token 反向解析出 userId, 再得到 classroom_no 参数，并在最后返回结果和错误码。

```
// API: /p_api/start_study?classroom_no=example
func StartStudyHandle(c *gin.Context, con *db.Connection) {
  userId := getUser(c)
  if userId == "" {
    return
  }
  classroomNo := c.Query("classroom_no")
  err := con.StartStudy(userId, classroomNo)
  if err != nil {
    reportError(c, DatabaseError, err.Error())
    return
  }
  c.JSON(http.StatusOK, gin.H{
    "code": Success,
  })
}
```

在完成 handler 之后，只需要在 register.go 里面注册该 handler 并绑定即可

```
  r.GET("/start_study", func(c *gin.Context) {
    StartStudyHandle(c, con)
  })
```

中间件 因为我们的多数操作多需要用户登陆，因此我们需要添加中间件进行验证 token，如果验证成功，将该 request 设置 userId 后转发给下一级的 handler，否则立即结束并返回错误码。

```
func WxAuthMiddleware() func(c *gin.Context) {
  return func(c *gin.Context) {
    authHeader := c.Request.Header.Get("token")
    //在此处验证token
    c.Set("user_id", claims.UserId)
    c.Next()
  }
}
```

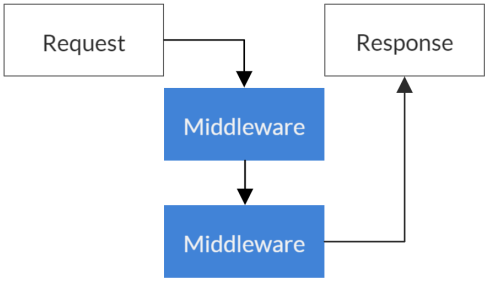

图 4: 中间件逻辑

4.1.4 管理服务模块设计

因为时间有限，我们使用了 django admin 框架与 Simple UI 模板，以便我们能最快的搭建出管理页面。[8]在使用 manage.py 搭建好项目后，我们在 models.py 中定义了我们所需要的 orm 模型，声明对应的数据类型、主键、外键、约束等成分，并连接上我们已有数据的数据库。以 locate_in 表为例

```
class LocateIn(models.Model):
    classroom_no = models.OneToOneField(
        Classroom, models.DO_NOTHING, db_column='classroom_no', primary_key=True,
    verbose_name='教室号')
    building_no = models.ForeignKey(
        Building, models.DO_NOTHING, db_column='building_no', blank=True, null=True
    ,verbose_name='楼号')
    class Meta:
        managed = False
        db_table = 'locate_ins'
        verbose_name = '地址'
        verbose_name_plural = '地址'
    def __str__(self):
        return "Loc"
```

此后，我们修改了 Simple UI 模板中的 html 代码，以使其符合本项目的需求。首页如下：

[8]这一部分提交的代码中,static 和 template 的 html 页面主要来源于 simpleUI 模板，我们只是基于该模板进行了定制，而并非从头搭建的网页。本项目的实现重点是 API 和小程序而并不是管理页面，因此在此特别说明

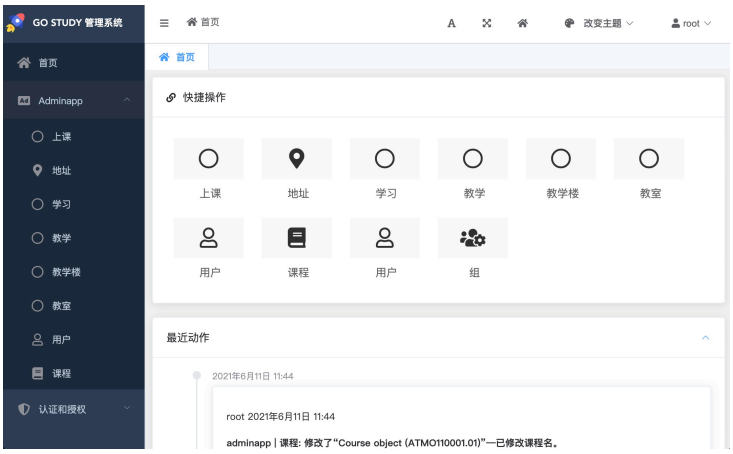

图 5: 管理页面主页

同时，我们也利用框架中的 filter 组件在部分页面加上了 filter 功能，如下图中搜索了 2021 年春季学期所有的博士生课程:

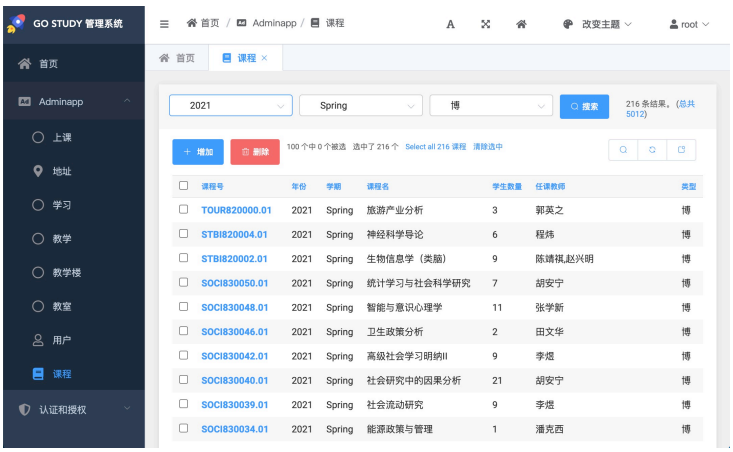

图 6: 搜索指定课程

最后，我们也提供了修改、添加、删除表的接口，例如修改课程的页面如下:

4.2 前端系统设计

4.2.1 整体风格设计

App 前端整体设计风格采用卡片式设计模式，结合多组件模式交互，尽量采取可视化组件和触控输入模式，减少用户复杂交互。交互逻辑简单、自然、符合直觉。

4.2.2 响应式刷新

动态页面刷新 前端开发采用响应式前端 DOM 树刷新机制，每个组件实例都对应一个 watcher 实例，它会在组件渲染的过程中把“接触”过的数据 property 记录为依赖。之后当依赖项的 setter 触发时，会通知 watcher，从而使它关联的组件的 DOM 标签进行重新渲染。
该过程是异步进行的，即在 js 数据改变时，数据改变推入 watcher 队列，对绑定的 html 标签进行异步刷新。

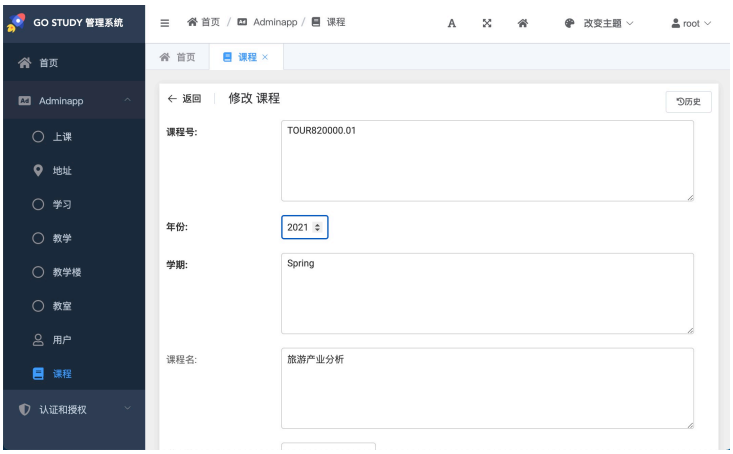

图 7: 修改课程的页面

页面数据绑定 微信小程序作为 MVVM 框架，数据实现双向绑定，js 端监听前端视图变化同步改变数据，实现数据的双向绑定。获取用户输入数据，并在触发函数中步入下一步业务逻辑。

4.2.3 页面生命周期

MVVM 模式提供了一整套页面生命周期函数回调，以供前端页面业务逻辑的编写。下图清晰展示了前端页面生命周期机制过程，

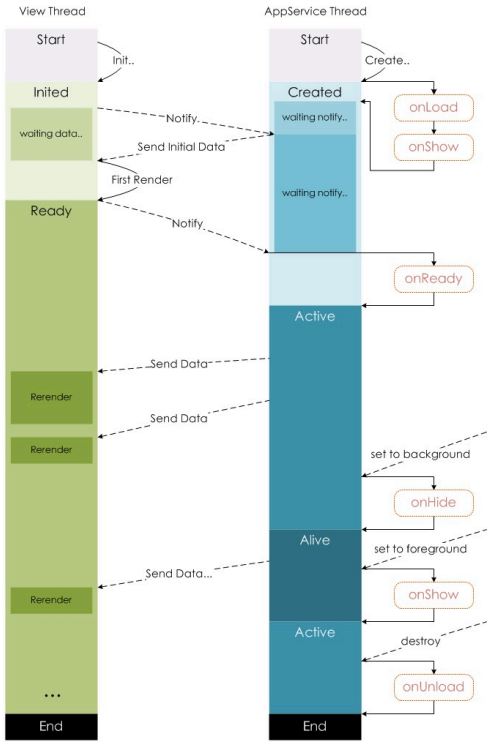

图 8: 页面生命周期

当页面被加载时，程序数据层和视图层同步开始加载，数据层完成两级 onLoad 和 onShow 生命周期回调，在 onLoad 中进行一些参数处理和初始化数据处理；前端视图初始化完成后，向数据层发送信号，数据层返回数据，前端视图层进行渲染。前端视图渲染完成发送信号到数据层，数据层回

调 onReady 函数。
当前端视图不再被浏览，不直接释放页面数据，而是仅在视图层隐藏页面 onHide 回调再次浏览时直接调用 onShow，加快页面访问。最后页面销毁的时候调用 onUnload 函数。

4.2.4 数据缓存机制

微信小程序提供了一套 Cache 机制，可以将数据暂时缓存于指定的 key 中。除非用户主动删除或因存储空间原因被系统清理，否则数据都一直可用。单个 key 允许存储的最大数据长度为 1MB，所有数据存储上限为 10MB。

```
wx.setStorage({
  key:"key",
  data:"value"
})
```

我们将例如当前空闲教室筛选、用户的 token 储存与缓存中。便于快速取用，每次先请求缓存，再 request，加快前端页面的刷新。

```
// 插入缓存
let key = _this.building + '&' + idx_query  + '&' + date_query;
wx.setStorageSync(key, res.data.empty_classroom);
```

4.2.5 主页面 Index Page

图 9: 主页面前端

图 10: 智能教室推荐页面

主页面 Main Page 主页面统计邯郸校区二三四五六教及光华楼西辅楼的空闲教室数量，采用圆环进度条式设计，让用户能够直观的查看空闲教室情况。头部卡片有智能推荐教室，和教室课程筛选按钮，点击可以路由到相应页面

智能教室推荐页面 Recommend Page 在当前页面步入时，会在 onLoad 生命周期函数中，请求获取用户手机当前地理位置。发送当前地理位置到后端 distance API 接口，返回当前位置到各大教学楼的距离[9]

```
success (res) {
  var resList = res.data.res;
  var dis = {};
  for (var i = 0; i < resList.length; i++) //将返回数据存入dis数组，
    dis[_this.data.buildingList[i]] = resList[i].distance;
  _this.setData({ distance: dis }) //设置并更新distance数据
  _this.generateRec();  // 教学楼打分
}
```

教室课表查询页面 CheckClsRoom Page 在教室课表查询页面可以通过搜索框直接搜索教室当天课程列表，卡片组件采用 Scroll View，使用列表渲染进行课程级别的 Cell 渲染以及相关数据绑定。

图 11: 教室课表查询页面

图 12: 空闲教室筛选页面

空闲教室筛选页面 EmptyClassRoom Page 空闲教室筛选页面，可以根据日期、时段以及节次进行空闲教室筛选，日期与后两者之间是“与”关系，后两者（时段、节次）之间互斥，选择一个就会将另一个清空至默认状态。

空闲教室 tag 同样采用列表渲染，并且对于长 tag 进行了优化，例如光华楼 HGX tag 长于其余 tag，在前端采用动态绑定不同的 css 样式。[10]

4.2.6 课程查询页面 ClassDB Page

[9]该页面点击空闲教室 tag 可以进入对应教室自习并开启自习时长计时

[10]该页面点击空闲教室 tag 可以进入对应教室自习并开启自习时长计时

图 13: 课程查询页面

图 14: 课表页面

空闲教室筛选页面 EmptyClassRoom Page 课程查询页面 GET 后端 empty classroom API 并进行数据解析和前端数据绑定，对每一个 CourseEntity 组织为以下模式

```
var CourseEntity = {
  CourseName: String, // 课程名称
  CourseId: String, // 课程代码
  Teacher: String,  // 教师名称
  CourseDayIdxList: Array,  // 课程日期列表 [int, int, ..., int] length 上限 7
  CourseTimeIdxList: Array, // 课程节次列表 [int, int, ... ,int]
  CourseTimeStr: String,    // 课程时间信息
  CourseTimeTag: Array(2)   // [int, int] 课程时间tag, 包括开始时间和结束时间
  __type__: String      //'Custom' or 'Course' 用以判断是自定义添加还是课程导入
};
```

4.2.7 课表页面 Curriculum Page

用户课表页面，显示用户当天课表，会从数据库 prefetch 用户历史添加的课表到队列中，按序插入课程表。对于课程表有时长互斥，对于学校课程有冲突检测

4.2.8 用户页面 User Page

用户登录 用户点击按钮登录，调用微信 open-type Login 来获取 code 发送到后端换取用户唯一指定 OpenId，接取后端返回 token 进行用户鉴权。token 24 h 过期后重新登录。

自习时长 用户登录后，自习称号和自习时长会在此页面中进行展示

```
setRate: function(time){
  var hour = time / 3600;
  if(hour < 1)
    this.setData({ Rate: '寝室自习王' });
```

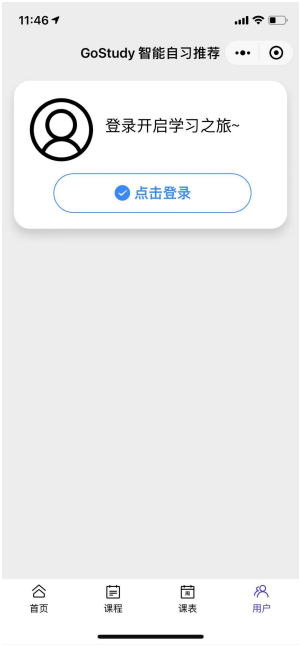

图 15: 用户登录前页面

图 16: 用户登录后页面

```
  else if(hour < 3)
    this.setData({ Rate: '拿A突击手' });
  else if(hour < 10)
    this.setData({ Rate: '黄金自习手' });
  else if(hour < 50)
    this.setData({ Rate: '三教久留' });
  else if(hour < 100)
    this.setData({ Rate: '三教王者' });
  else if(hour < 250)
    this.setData({ Rate: '轻松保研' });
  else
    this.setData({ Rate: 'GPA 4.0' });
}
```

4.2.9 用户自习时长统计逻辑

用户自习时长统计逻辑，是在用户在智能教室推荐页面或空闲教室筛选页面点击确认进入教室自习后触发的，其包括了前端的数据展示、数据路由、后端数据库数据交互等一系列业务逻辑。

简单流程概述 在用户点击进入教室自习后，后端通过点击事件，获得到对应 tag 的标签 Id，进而索引到 tag 取回对应教室信息，并路由到用户页面，设置全局用户学习状态（isStudy）为真，并获取当前时间，发送至后端数据库 start study API，表示用户开始学习。在 isStudy 为真时，用户界面会展示学习计时和停止学习按钮。当用户点按停止学习按钮，逻辑层将 isStudy 设为假，并向后端 end study API 停止学习并请求总学习时长，进行格式化后显示在用户前端学习时长中。

4.2.10 前端页面通信

page 页面数据传递 page 页面的数据传递是通过 navigateTo 中 url 携带参数完成的，路由页面在 onLoad 的参数中获取 url 传递的参数，并进行解码。

tabBar 页面数据传递 微信小程序并不支持 tabBar 页面之间的传参，因而 tabBar 页面之间的数据通信是通过设置全局变量来完成的。这就涉及到空间花费问题，为了不过多的浪费空间前端 tabBar 页面设置的页面通信都不传递整个数据对象，而是设置一个全局变量 key 来完成的。而完整的数据对象会在路由前被存入缓存，再在路由后通过之前设置的全局变量 key 来获取缓存，从而达成 tabBar 页面之间的数据传递。

5 特色和创新点

5.1 后端特色和创新点

5.1.1 原生支持多服务器架构

本项目使用 Nginx 进行负载均衡，只需进行少许的配置，就能在未来用户增多时配置多服务器。以 API 服务的转发为例，目前的 Nginx(对此) 的配置如下

```
location /api {
proxy_pass http://apiservice:1234;
}
```

在使用多服务器时，我们可以将其改为:

```
upstream apiservergroup {
      server apiserver1Addr;
      server apiserver2Addr;
      ...
}
server {
    # 此处进行其他配置
    location /api {
    proxy_pass apiservergroup;
    }
}
```

因为架构中各个组件互相独立，因此只需简单修改几行代码便能实现服务的扩展。

5.1.2 使用 Redis 作为缓存

因为 PostgreSQL 的速度较慢，且在我们的架构中，其作为网络数据库受到了网络传输能力的制约。因此，我们引入了 Redis 来进行缓存加速。Redis 作为运行在内存中的数据库，性能远高于 PostgreSQL。具体的 redis 实现已经在系统实现部分详述，本处不再赘述。

5.1.3 高安全性

因为该项目将在公网上线，因此必须要有足够的安全性，我们完成了以下三点，以提高安全性。

容器化 容器化使得不同的服务之间有原生的隔离，攻击者很难离开容器，攻击宿主机或其他容器，增加了服务的鲁棒性

HTTPS 传输 本项目优先使用 HTTPS 传输，同时也支持 HTTP 用于给老旧的浏览器提供服务，我们配置了 SSL 证书

无密码验证 本项目使用了无密码验证，通过类似 Oauth2.0 的签发模式，完成了用户验证

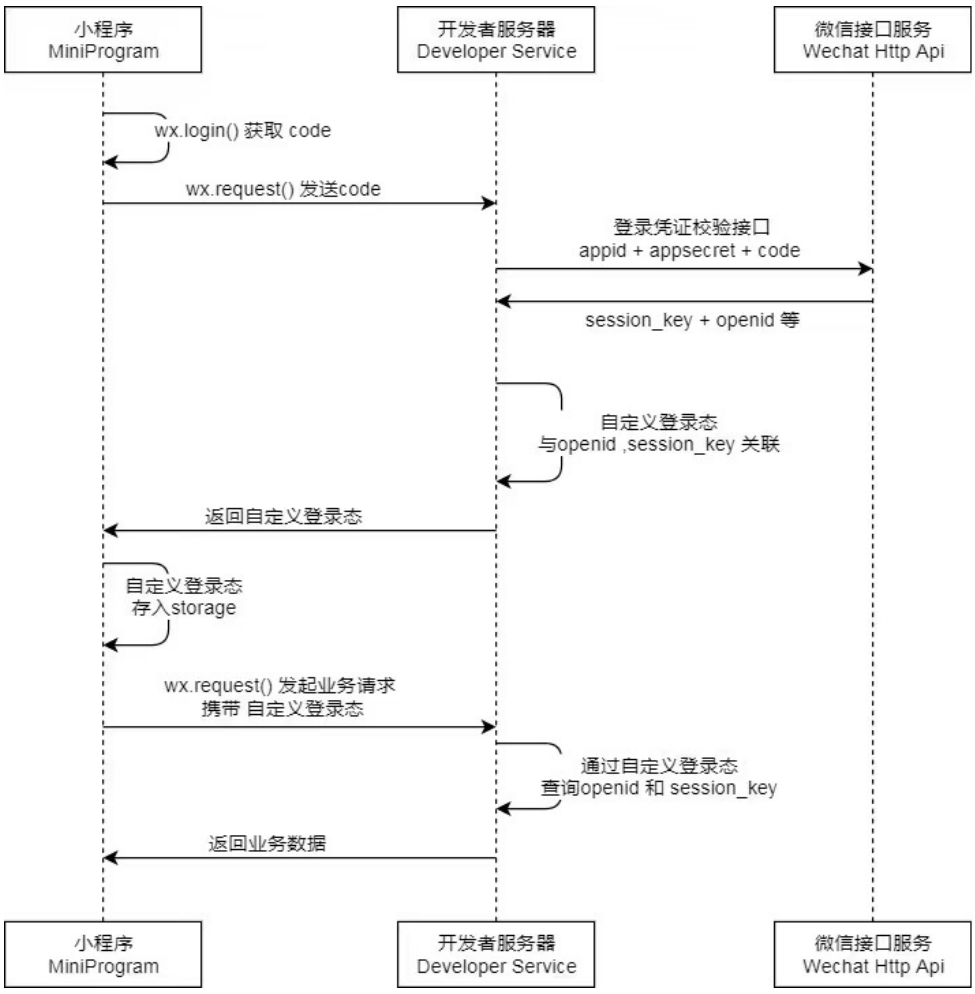

图 17: 无密码验证流程

5.1.4 易于部署

本项目完全使用 docker，因此在服务器上不需要运维，只需使用 docker-compose 命令就能一键构建映像并运行。

5.2 前端特色和创新点

5.2.1 智能教室推荐系统

智能的教室推荐系统，根据距离和个人喜好进行用户级教室推荐。

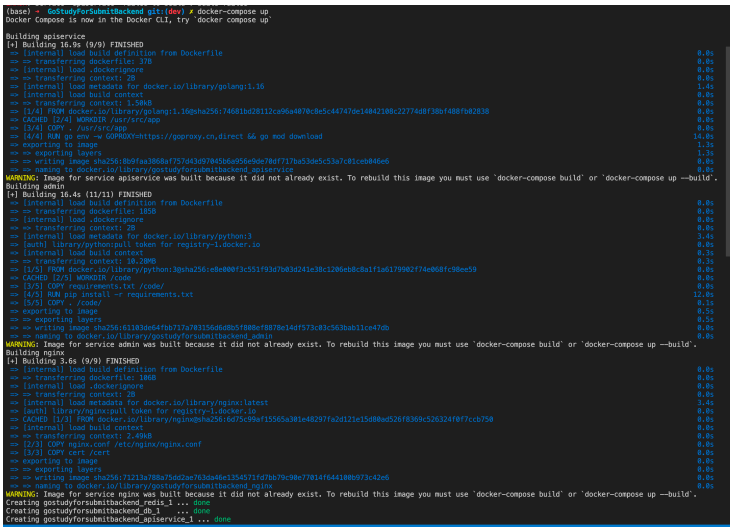

图 18: docker-compose 部署截图

5.2.2 自习时长管理系统

自习时长管理系统，促进用户进入教室自习的同时，为我们提供用户偏好的教室数据，与智能教室推荐系统实现生态闭环。

5.2.3 自习称号系统

不同的自习时长对应不同的自习称号，能够激发同学们的学习兴趣与学习热情。

5.2.4 课表管理系统

灵活强大的课表管理系统，让学生不再为课程和事务的管理发愁。

5.2.5 教室课程查询

直接查看指定教室一天内的课程表，让用户更好的了解课程信息，降低了用户的时间成本。

6 实验分工

本实验中两人贡献度相同，均参与设计前端、后端与数据库。其中王少文在后端部分贡献比重较大，谢子陬在前端部分贡献比重较大。

7 提交文件说明

7.1 后端提交文件说明

后端的所有文件均位于 backend 文件夹中

7.1.1 各文件/文件夹说明

1. admin: 管理页面的所有文件, 其中 static/admin,template/是引用的 simpleUI 的模板，个人基于该模板进行了修改

2. apiservice: 后端 api 的所有文件，包含爬虫、数据库、web 服务
3. nginx: 启动 nginx 服务的证书文件夹、Dockerfile 和 nginx.conf
4. script: 备份数据库、还原数据库的脚本，添加部分函数、视图、触发器的 SQL 脚本，自动化测试 API 服务的 Python 脚本
5. docker-compose.yml: 用于启动 docker 的 docker-compose 文件

7.1.2 运行方法

在配置正常的情况下，可以在 backend 文件夹中使用 docker compose up 启动所有服务。
但因为原先的配置文件中存在小程序秘钥、服务器证书、服务器秘钥等敏感信息，不便于上传，所以在本次实验提交的文件中，已将该类型字段改为无效值。因此若要运行，请先在 backend/nginx 中添加 https 证书，并在 docker-compose.yml 文件中，将各值修改为您即将部署的服务器/小程序的值，再尝试进行运行。

7.2 前端提交文件说明

前端的所有文件均位于 frontend 文件夹中，页面在 pages 中，src 中储存各类前端素材。

7.2.1 页面层级组织

```
pages
|-- classdb
|-- curriculum
|-- index
|   |-- check_classroom
|   |-- empty_classroom
|   |-- recommend
|-- user
```

7.2.2 运行方法

导入微信开发者工具，可以使用测试号 appid，之后会自动生成 project.config.json 在 app.js 中填入基础 API (baseUrl)、鉴权 API (authUrl)、带 token 验证 API (pUrl) 并添加到微信小程序 URL 白名单中，编译运行程序即可。

8 实验总结

本次数据库 PJ 我们从前后端完整搭建了小程序应用，完成了应用的线上部署，完成完整开发流程和初期测试阶段。熟练使用 PostgreSql 数据库进行业务数据储存管理和设计部署 Redis 应对高并发数据流场景。通过 Nginx 进行反向代理实现具有可扩展性和负载均衡能力的后端模块。同时，我们也认识到开发一款成熟、稳定、可供上线的软件的困难，我们也将会在提交后继续迭代该项目，方便同学们的学习与生活。

9 致谢

感谢卢暾老师一学期的辛勤教导，以及梁依帆、管正青助教的指导，以支持我们完成本次课程设计和展示。再次感谢！