

架构设计文档

酒店客房管理系统

学号	姓名
21307037	金思琪
21307056	钟欣余
21307062	郑越
21307035	邓栩瀛
21307026	何倩盈

架构设计文档

1. 引言

- 1.1 背景和目的
- 1.2 范围

2. 系统概述

- 2.1 系统目标
- 2.2 系统环境
- 2.3 系统功能

3. 架构概述

- 3.1 整体架构图
- 3.2 架构风格
- 3.3 架构原则

4. 详细设计

- 4.1 模块划分
- 4.2 接口设计
- 4.3 数据库设计
- 4.4 安全设计
- 4.5 性能设计

5. 系统开发

- 5.1 管理员模块
 - 5.1.1 管理员注册模块
 - 5.1.2 管理员登录模块
- 5.2 用户模块
 - 5.2.1 用户信息管理模块
- 5.3 添加房间模块
- 5.4 预定房间模块
- 5.5 生成账单模块

6. 关键设计决策

- 6.1 技术选型
- 6.2 设计权衡
- 6.3 风险和假设

7. 非功能性需求

- 7.1 可扩展性需求

7.2 可维护性需求
8. 部署架构
8.1 部署环境
8.2 部署流程
8.3 监控和日志
9. 附录
9.1 数据结构

1. 引言

1.1 背景和目的

本架构设计文档旨在描述酒店客房管理系统的整体架构和关键设计决策，系统通过简化预订、入住、退房流程，提升用户体验和酒店运营效率。

1.2 范围

文档涵盖系统的功能设计、数据库设计、详细模块设计、关键技术选型、非功能性需求、部署架构等内容。

2. 系统概述

2.1 系统目标

- 提供便捷的客房预订体验
- 支持酒店高效处理预订、入住、退房等工作
- 实时更新房间状态信息
- 收集和分析数据，支持酒店决策

2.2 系统环境

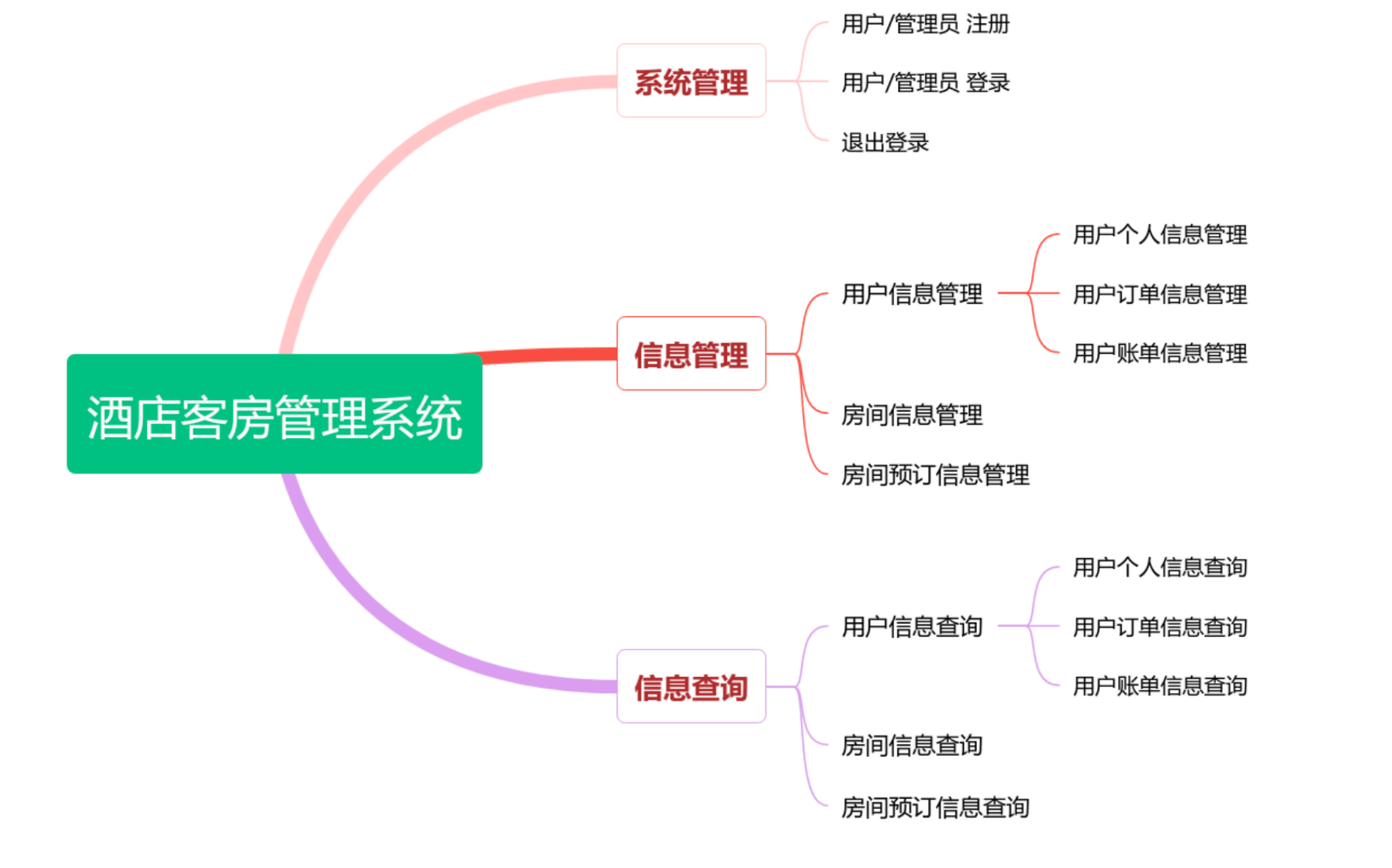
系统基于Web，实现了前后端分离的设计，后端采用Python和MySQL，前端使用HTML/CSS/JavaScript。

2.3 系统功能

- 用户注册和登录
- 客房预订、支付和查询
- 酒店信息展示
- 管理员管理客房和用户信息

3. 架构概述

3.1 整体架构图



3.2 架构风格

采用分层架构，分为展示层、业务逻辑层和数据访问层。

3.3 架构原则

- 高内聚低耦合
- 可扩展性
- 可维护性

4. 详细设计

4.1 模块划分

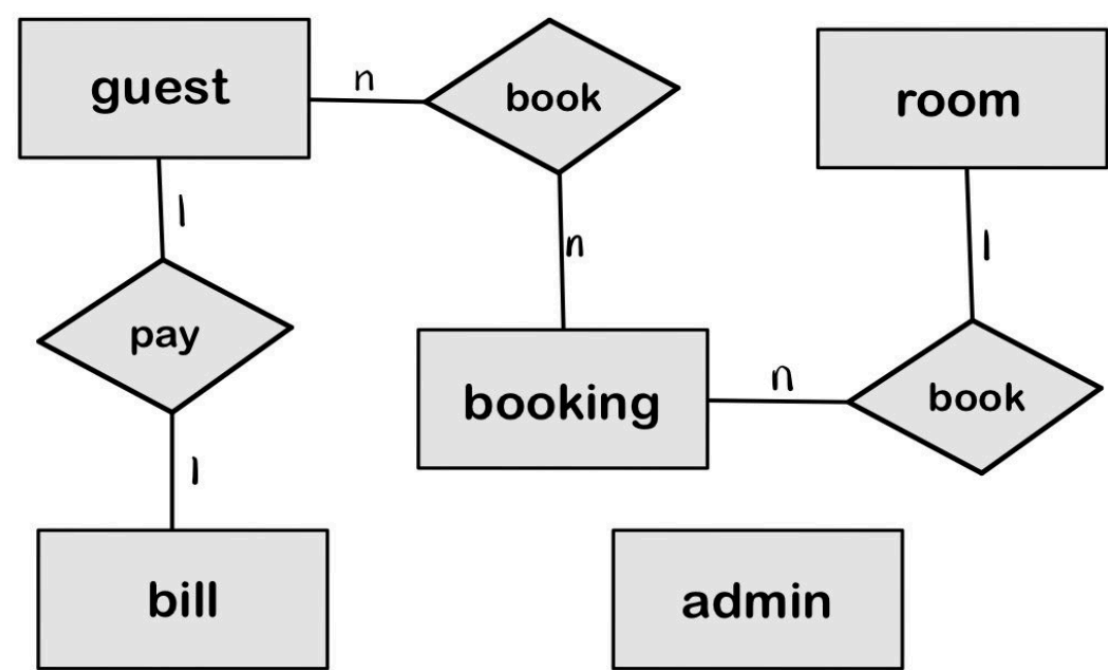
- 用户模块：注册、登录、信息管理
- 管理员模块：房间管理、用户管理
- 预订模块：房间预订、支付、账单生成

4.2 接口设计

各模块之间通过API进行数据交互，使用RESTful风格设计接口。

4.3 数据库设计

▪ ER图：



▪ 关系分析：

- 1. 一对一关系（One-to-One Relationship）：每个客人（guest）都有一个对应的账单（bill），同时每个账单只属于一个客人，构成了一对一关系。
- 2. 一对多关系（One-to-Many Relationship）：每个房间（room）可以对应多个预订（booking），但是每个预订只能对应一个房间。这构成了一对多关系。
- 3. 多对多关系（Many-to-Many Relationship）：每个客人（guest）可以有多个预订（booking），同时每个预订也可以对应多个客人，这构成了多对多关系。

▪ 数据库创建：

创建账单表

```
CREATE TABLE bill(
  bill_id INTEGER PRIMARY KEY,
  g_id INTEGER REFERENCES guest(g_id),
  from_ DATE,
  to_ DATE,
  total_money FLOAT
);
```

创建房间表

```
CREATE TABLE room(
  r_id INTEGER PRIMARY KEY,
  r_type VARCHAR(45),
  price FLOAT,
  num_window INTEGER,
  allow_smoke VARCHAR(5),
  num_bed INTEGER,
  bathtub VARCHAR(45)
);
```

创建客人表

```
CREATE TABLE guest(  
  g_id INTEGER PRIMARY KEY,  
  name VARCHAR(45),  
  password VARCHAR(45) NOT NULL,  
  phone VARCHAR(45) NOT NULL,  
  e_mail VARCHAR(45),  
  country VARCHAR(40)  
);
```

创建管理员表

```
CREATE TABLE admin(  
  a_id INTEGER PRIMARY KEY,  
  name VARCHAR(45) NOT NULL,  
  password VARCHAR(45) NOT NULL,  
  phone VARCHAR(45) NOT NULL,  
  e_mail VARCHAR(45)  
);
```

创建预订表

```
CREATE TABLE booking(  
  booking_id INTEGER PRIMARY KEY,  
  r_id INTEGER REFERENCES room(r_id),  
  g_id INTEGER REFERENCES guest(g_id),  
  from_ DATE,  
  to_ DATE,  
  meal VARCHAR(45),  
  num_adult INTEGER,  
  num_child INTEGER,  
  is_paid VARCHAR(45)  
);
```

4.4 安全设计

- 使用加密技术存储用户密码
- 实现身份验证和授权
- 数据传输使用SSL加密

4.5 性能设计

- 数据库索引优化
- 缓存常用查询结果
- 分布式部署

5.系统开发

5.1 管理员模块

5.1.1 管理员注册模块

由于管理员的权限和系统的安全问题，管理员仅能在终端注册，无法在网页注册。

这段代码的功能就是创建一个新的管理员账户，要求输入用户名、电话号码、邮箱和密码，并将这些信息插入到名为admin的数据库表中。

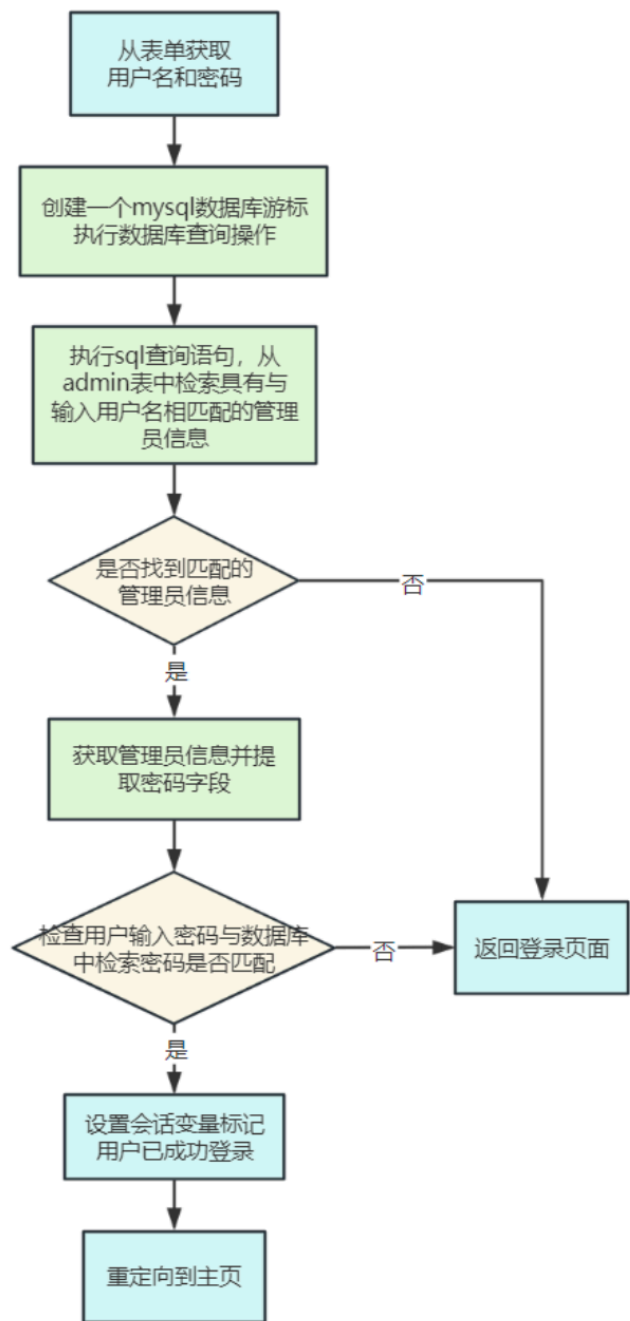
同时，代码还对用户名和密码的合法性进行了基本的验证，确保用户名不重复，并且确认密码输入的一致性。

```
connection = pymysql.connect(host='localhost', port=3306, user='root', password='123456',
database='hms')
cur = connection.cursor()
while True:
    name = input('User Name: ')
    result = cur.execute('select * from admin where name="{0}"'.format(name))
    if result > 0:
        print('User Name Occupied!')
    else:
        break
phone = input('Phone: ')
e_mail = input('E-mail: ')
while True:
    password = getpass.getpass('Password: ')
    confirm = getpass.getpass('Confirm: ')
    if confirm != password:
        print('Password does not match, input again')
    else:
        break
admin_id = cur.execute('select * from admin') + 1
cur.execute('insert into admin (a_id,name,password,phone,e_mail)values ("{}", "{}", "{}", "{}", "{}")'.format(admin_id,name, password, phone,e_mail))
connection.commit()
cur.close()
connection.close()
```

5.1.2 管理员登录模块

该模块用于处理管理员登录请求。

流程图如下



系统定义了一个路由，当用户访问 `/admin_login` 时，将执行该函数。支持GET和POST请求。当用户提交登录表单时执行以下代码操作。

从表单中获取名为username和password的输入字段，即管理员的用户名和输入的密码。创建一个MySQL数据库游标，用于执行数据库查询操作。执行SQL查询语句，从名为admin的表中检索具有与输入的用户名相匹配的管理员信息。检查是否找到匹配的管理员信息。如果找到匹配的管理员信息，则获取数据库中匹配用户名的管理员信息并提取密码字段。之后检查用户输入的密码与数据库中检索到的密码是否匹配。如果用户名和密码验证成功，设置会话（Session）变量来标识用户已经成功登录。将用户名、用户ID和管理员状态存储在会话变量中。重定向到应用的主页。如果验证失败或未找到匹配的用户名信息，执行`cur.close()`后返回登录页面 `render_template('login.html')`。

```
@app.route('/admin_login', methods=['GET', 'POST'])
def admin_login():
    if request.method == 'POST':
        username = request.form['username']
        password_candidate = request.form['password']
        cur = mysql.connection.cursor()
        result = cur.execute('SELECT * FROM admin WHERE name = %s', [username])
        if result > 0:
            data = cur.fetchone()
            password = data['password']
            # Compare Passwords
            cur.close()
            if password_candidate == password:
                session['logged_in'] = True
                session['username'] = username
                session['user_id'] = data['a_id']
                session['is_admin'] = True
```



```
        return redirect('/')
    cur.close()
    return render_template('login.html')
```

5.2 用户模块

用户可以划分为：用户注册模块、用户登录模块、用户信息管理模块、用户信息查看模块。由于用户注册模块和用户登录模块与上述管理员注册模块和管理员登录模块实现逻辑基本一致，故不多加赘述。本部分将选取用户信息管理模块进行详细说明。

5.2.1 用户信息管理模块

用户信息管理模块具有编辑客人信息、删除客人信息的功能。

下面这段代码用于在管理员权限下编辑特定ID的客人信息。

函数获取的客人的ID—g_id。执行SQL查询语句，检索指定 g_id 的客人信息，并获取数据库中匹配给定g_id的客人信息。

如果找到指定ID的客人信息，则创建一个名为 GuestUpdateForm 的表单对象，用于在 HTML页面中编辑客人信息。当提交的表单数据有效时，更新数据库中的特定表格guest中指定g_id的客人信息，并显示一个成功更新的提示信息。如果未找到指定 ID 的客人信息，则返回 guest not found并返回 404 错误。

```
@app.route('/edit_guest/<int:g_id>', methods=['GET', 'POST'])
@is_admin_logged_in
def edit_guest(g_id):
    cur = mysql.connection.cursor()
    result = cur.execute('select * from guest where g_id="{}".format(g_id)')
    guest = cur.fetchone()
    cur.close()
    if result > 0:
        form = GuestUpdateForm()
        if form.validate_on_submit():
            values = request.form.to_dict()
            if form.password.data != '':
                del values['confirm']
                values['password'] = str(form.password.data)
            else:
                del values['confirm']
                del values['password']
            update_value('guest', values, 'g_id="{}".format(g_id)')
            flash('update successfully')
            return redirect('/guests')
        elif request.method == 'GET':
            form.country.data = guest['country']
            form.phone.data = guest['phone']
            form.e_mail.data = guest['e_mail']
            return render_template('edit_guest.html', form=form)
    else:
        return 'guest not found', 404
```

其中，更新数据库中的特定表格guest中指定g_id的客人信息通过自定义函数update_value()实现。这个函数根据传入的表格名称、字段数值字典以及条件，构建SQL命令，执行更新操作。然后提交对数据库的更改。因为这段代码使用的是传统的SQL字符串拼接方式构建命令，存在SQL注入的风险。


```
def update_value(table, values, condition='true'):
    cmd = 'update `{}` set '.format(table)
    for v in values:
        cmd += '`{}`= "{}",'.format(v, values[v])
    cmd = cmd.strip(',')
    cmd += ' where {}'.format(condition)
    cur = mysql.connection.cursor()
    try:
        cur.execute(cmd)
        mysql.connection.commit()
    except Exception as e:
        print('update error: {}'.format(e))
    cur.close()
```

下面这段代码用于在管理员权限下删除特定ID的客人信息。

`g_id`是作为参数传入的客人ID。使用数据库游标执行SQL查询，删除数据库中特定ID的客人记录。在执行删除操作之前，确保管理员已登录，并在删除完成后重定向到相应页面。

```
@app.route('/delete_guest/<int:g_id>', methods=['POST'])
@is_admin_logged_in
def delete_guest(g_id):
    cur = mysql.connection.cursor()
    cur.execute('delete from guest where g_id=%s', [g_id])
    mysql.connection.commit()
    cur.close()
    return redirect('/guests')
```

5.3 添加房间模块

这段代码实现了添加房间信息到MySQL数据库的功能。首先定义了一个路由，该路由可以处理GET和POST请求，只有管理员登录后才能访问这个路由。使用AddRoomForm类初始化一个表单对象，该对象用于处理添加房间的表单。当表单通过验证并提交时，执行以下操作：从表单中提取数据，将其转换为字典，并存储在`values`变量中；从`values`字典中获取房间数量`num`。循环调用`add`函数，构建SQL插入语句，并执行插入操作，将房间信息插入到数据库的`room`表中。

```
@app.route('/add_room', methods=['GET', 'POST'])
@is_admin_logged_in
def add_room():
    form = AddRoomForm()

    if form.validate_on_submit(): # 包含了判断request的methods
        values = request.form.to_dict()
        num = int(values['num_rooms'])
        del values['num_rooms']
        cur = mysql.connection.cursor()
        for i in range(num):
            values['r_id'] = cur.execute('SELECT * from room') + 1
            add('room', values)
        flash('add successfully')
        return redirect('/rooms')

    return render_template('add_room.html', form=form)
```

5.4 预定房间模块

这段代码实现了预订房间的功能。首先定义一个路由，该路由可以处理GET和POST请求，只有已登录的用户才能访问这个路由。使用BookForm类初始化一个表单对象，该对象用于处理预订房间的表单。当表单通过验证并提交时，执行以下操作：检查日期合法性，如果退房日期早于或等于入住日期，或者入住日期早于今天，则返回表单页面；否则执行SQL查询，找出在指定日期范围内尚未被预订的房间，用户可以选择并预订房间。

```
@app.route('/book', methods=['GET', 'POST'])
@is_logged_in
def book():
    form = BookForm()
    if form.validate_on_submit():
        if session.get('is_admin'):
            if form.g_id.data is None:
                flash('Input guest id')
                return render_template('book.html', form=form)
            else:
                session['g_id'] = form.g_id.data
        else:
            session['g_id'] = session.get('user_id')
    today = datetime.date.today()
    check_in = form.check_in.data
    check_out = form.check_out.data
    if check_out <= check_in or check_in < today:
        flash('Illegal date input')
        return render_template('book.html', form=form)
    cur = mysql.connection.cursor()
    # 从所有房间中找出还没被预定的房间
    check_in = check_in.strftime('%Y-%m-%d')
    check_out = check_out.strftime('%Y-%m-%d')
    cur.execute(
        'select * from room where r_id not in (select r_id from booking where `from_`<=%s
and `to_`>=%s)',
        [check_out, check_in])
    rooms = cur.fetchall()
    session['check_in'] = check_in
    session['check_out'] = check_out
    session['meal'] = form.meal.data
    session['num_adult'] = form.num_adult.data
    session['num_child'] = form.num_child.data
    return render_template('select_rooms.html', rooms=rooms)
return render_template('book.html', form=form)
```

5.5 生成账单模块

这段代码实现了生成未付款账单的功能。首先定义一个路由，该路由可以处理GET和POST请求，只有已登录的用户才能访问这个路由。通过SELECT语句查询未付款的预订信息，构建账单数据。遍历每个预订信息，查询相应房间的价格，并计算预订的天数乘以每天的价格，得到每个预订的总费用。如果请求方法是POST，表示用户提交了付款请求，更新数据库中相应预订记录的支付状态为已支付。

```
@app.route('/bills', methods=['GET', 'POST'])
```

```
@is_logged_in
def bills_unpaid():
    cur = mysql.connection.cursor()
    cur.execute(
        'select * from booking where `g_id`="{}" and
`is_paid`="no"'.format(session.get('user_id'))
    )
    bookings = cur.fetchall()
    # print(bookings)
    num = len(bookings)
    total = 0
    for i in range(num):
        cur.execute('select price from room where r_id="{}"'.format(bookings[i]['r_id']))
        price = cur.fetchone()['price']
        intervals = bookings[i]['to_'] - bookings[i]['from_']
        bookings[i]['cost'] = price * intervals.days
        bookings[i]['price'] = price
        bookings[i]['days'] = intervals.days
        total += bookings[i]['cost']
    if request.method == 'POST':
        for i in range(num):
            cur.execute('update booking set is_paid="yes" where booking_id="
{}"'.format(bookings[i]['booking_id']))
            mysql.connection.commit()
            cur.close()
        return render_template('QR_code.html')
    cur.close()
    return render_template('bills.html', g_name=session.get('username'), bookings=bookings,
num=num, total=total)
```

6. 关键设计决策

6.1 技术选型

- 编程语言：Python
- Web框架：Flask
- 数据库：MYSQL
- 前端：HTML/CSS/JavaScript

6.2 设计权衡

- 选择MYSQL作为数据库语言，权衡了其性能和易用性
- 采用分层架构，权衡了系统的复杂性和可维护性

6.3 风险和假设

- 风险：用户并发访问导致的性能问题
 - 假设：用户数量和数据量在可控范围内
-

7. 非功能性需求

7.1 可扩展性需求

- 支持增加新的房间类型和预订功能

7.2 可维护性需求

- 提供详细的开发文档
- 代码遵循命名规范和编码规范

8. 部署架构

8.1 部署环境

- 服务器：Linux
- 数据库服务器：MYSQL
- Web服务器：Nginx

8.2 部署流程

- 部署前端代码到Web服务器
- 部署后端代码到应用服务器
- 部署数据库到数据库服务器

8.3 监控和日志

- 监控服务器性能和数据库性能
- 记录系统日志和错误日志
- 使用Python中的 `prometheus_client` 和 `grafana-api` 监控系统性能和应用状态


9. 附录

9.1 数据结构

admin	
	a_id: int
	name: varchar(45)
	password: varchar(45)
	phone: varchar(45)
	e_mail: varchar(45)

guest	
	g_id: int
	name: varchar(45)
	password: varchar(45)
	phone: varchar(45)
	e_mail: varchar(45)
	country: varchar(40)

booking	
	booking_id: int
	r_id: int
	g_id: int
	from_: date
	to_: date
	meal: varchar(45)
	num_adult: int
	num_child: int
	is_paid: varchar(45)

room	
	r_id: int
	r_type: varchar(45)
	price: float
	num_window: int
	allow_smoke: varchar(5)
	num_bed: int
	bathtub: varchar(45)

bill	
	bill_id: int
	g_id: int
	from_: date
	to_: date
	total_money: float