《数据库系统原理》大作业

系统实现报告

题目名称：游戏分发平台数据库管理

学号及姓名：18373080 杨孜名_____

18373584 甘天淳_____

18373599_崔建彬_____

2020 年 10 月 2 日

目录

系统实现报告

一、 实验概述

1. 实验内容概述：

本次数据库实验主要完成的内容为 SDEAM 游戏商城。

游戏商城包含商家上架游戏、下架游戏、更改游戏属性，用户查看游戏、添加游戏到订单、确认订单并购买游戏、创建收藏夹、添加游戏到收藏夹、添加游戏评论、给游戏打分，管理员删除评论、删除违约用户、删除违约商家等功能。

上述内容共包含七个实体、九张数据库表以及对数据库的增、删、查、改等操作。同时，还使用了大量触发器来完善数据库使用、提升数据库效率。

实现层面后端采用了 python 语言进行编写，应用 Flask 作为 web 框架、使用 SqlAlchemy 作为 ORM 来处理数据库 (Object-Relational Mapping)。前端则使用 Vue 框架和 Ant-Design-Vue 组件库进行具体页面的设计与实现，使用 axios 进行对后端接口的请求。

2. 开发环境说明：

1） 操作系统：

Windows10

2） 开发环境:

Python:3.8

MySQL:8.0.2

Flask:1.1.2

Microsoft Edge:87.0.664.60

Vue:2.5.2

Ant-design-vue:1.6.5

Webpack:3.6.0

3） 系统：

RAM: 16.00GB

系统类别: 64 位操作系统

处理器: Intel(R)Corei7-8750H CPU @ 2.60GHz

3. 后端框架：

Flask:

Flask 是一个微型的 Python 开发的 Web 框架,基于 Werkzeug WSGI 工具箱和 Jinja2 模板引擎。Flask 使用 BSD 授权。 Flask 也被称为“microframework”，因为它使用简单的核心，用 extension 增加其他功能。Flask 没有默认使用的数据库、窗体验证工具。然而，Flask 保留了扩增的弹性，可以用 Flask-extension 加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

特点:

1. 开发相对简单，适合敏捷开发。
2. 拓展性好，可以使用 SqlAlchemy 等 ORM 辅助开发。
3. 基于 python 语言，实现起来较为迅速。

4. 数据库框架

SQLAlchemy:

SQLALchemy 是一个基于 Python 的 ORM 操作数据库的框架。

ORM 就是 Object-Relational Mapping，把关系数据库的表结构映射到对象上。而 SQLAlchemy 对象关系映射器就提供了一种方法,用于将用户定义的 Python 类与数据库表相关联。它包括一个透明的同步对象及相关行之间的所有变化的系统，称为工作单元，以及根据用户定义的类及其定义的彼此的关系表达数据库查询的系统。

5. 前端框架与组件库

Vue：

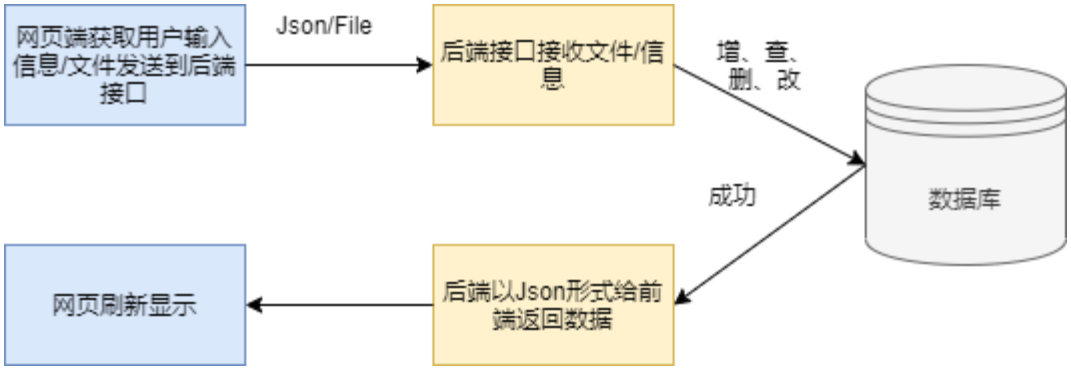
Vue 是一套用于构架用户界面的渐进式框架。与其他大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

二、 系统结构设计

1. 体系结构：

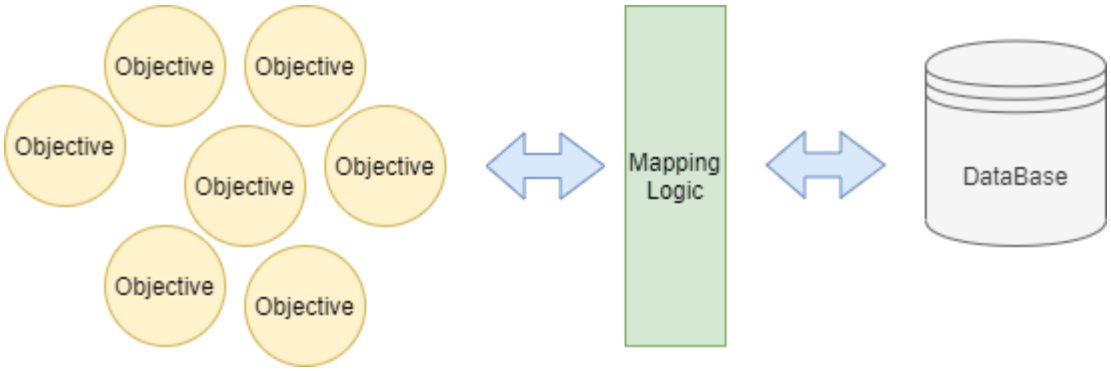
本次实验采用的是前后端分离的 web 开发。后端采用基于 python 语言的 Flask 框架，前端采用 Vue 框架和 Ant-Design-Vue 组件库

整体流程图如图所示:



前后端的交互以 json/file 的形式来传递信息文件。

后端与数据库的操作采用 ORM 的形式对数据库进行访问，其结构示意图如下:



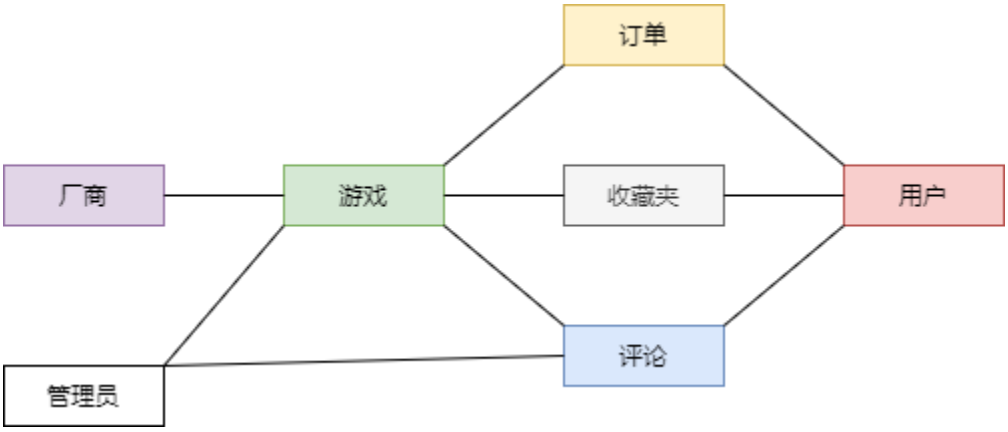
2. 功能结构：

• 简介：

本次开发的内容为电子游戏商城。故主要涉及的实体有：用户、厂商、管理员、游戏、收藏夹、评论、订单等。

• 关系图：

功能结构关系图如下图所示。



其中对每个实体涉及到的功能描述如下

• 游戏：

对于游戏，涉及到的功能有：上架游戏、下架游戏、购买游戏、查看游戏详情、推荐游戏、更改游戏信息。

• 用户：

对于用户，涉及到的功能有：注册用户、用户登录、创建收藏夹、添加收藏夹、移出收藏夹、修改用户信息、添加评论、删除评论、添加游戏订单、确认游戏订单、查看游戏界面、查看收藏夹、查看评论、查看订单等功能。

• 厂商：

对于厂商，涉及到的功能有：厂商注册、厂商登录、厂商更改个人信息、厂商上架游戏、厂商下架游戏、厂商更改游戏信息。

• 管理员：

对于管理员，涉及到的主要功能有：管理员删除游戏、管理员删除违规用户、管理员删除评论、管理员删除违规厂商。

• 收藏夹：

对于收藏夹，涉及到的主要功能有：创建收藏夹、添加收藏夹、从收藏夹中移出、删除收藏夹、查看收藏夹。

• 评论：

对于评论，涉及到的主要功能有：添加评论、游戏打分、查看评论、删除评论。

• 订单：

对于订单，涉及到的主要功能有：添加订单、确认订单、删除订单。

三、 数据库基本表的定义

根据数据库系统设计的 3NF 原则，本次数据库系统共包含 8 个基本表，其分别为:订单表(Buy),评论表(Comment),厂商表(Developer),收藏夹表(Favorite),收藏夹-游戏表(Favorite2Game),游戏表(Game),用户表(Uesr),用户拥有游戏表(User2Game)。下面分别叙述每个表的详细内容。

1. 订单表(Buy)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	buyID	Int	10	是	是	订单 ID
2	buyTime	Varchar	30	是	否	购买时间
3	userID	Int	10	是	否	购买用户 ID
5	gameID	Int	10	是	否	购买游戏 ID
6	status	Enum	2	是	否	订单状态

2. 评论表(Comment)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	commentID	Int	10	是	是	评论 ID
2	commentContents	Varchar	260	是	否	评论内容
3	commentTime	Varchar	30	是	否	评论时间
5	userID	Int	10	是	否	评论用户 ID
6	gameID	Int	10	是	否	评论游戏 ID
7	grade	Enum	5	是	否	用户打分

3. 厂商表(Developer)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	developerID	Int	10	是	是	厂商 ID
2	developerName	Varchar	50	是	否	厂商姓名
3	developerPassword	Varchar	50	是	否	厂商登录密码

5	developerTel	Varchar	20	否	否	厂商电话
6	developerGraph	Varchar	30	否	否	厂商头像

4. 收藏夹表(Favorite)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	favoriteID	Int	10	是	是	收藏夹 ID
2	favoriteName	Varchar	50	是	否	收藏夹命名
3	userID	Int	10	是	否	收藏夹用户 ID

5. 游戏表(Game)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	gameID	Int	10	是	是	游戏 ID
2	gameName	Varchar	50	是	否	游戏姓名
3	gamePrice	Int	10	是	否	游戏价格
5	introduction	Varchar	50	否	否	游戏介绍
6	gradeAvg	Float	20	否	否	平均分数
7	gameType	Enum	6	是	否	游戏类别
8	developerID	Int	10	是	否	厂商 ID
9	gameGraph	Varchar	300	否	否	游戏缩略图
10	gameDetailGraph	Varchar	300	否	否	游戏详情图

6. 用户表 (User)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	userID	Int	10	是	是	用户 ID
2	userName	Varchar	50	是	否	用户姓名
3	userPassword	Varchar	50	是	否	用户密码
5	email	Varchar	30	否	否	用户邮箱
6	userTel	Varchar	20	否	否	用户电话

7	uesrGraph	Varchar	300	否	否	用户头像

7. 收藏夹-游戏表(Favorite2Game)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	favoriteID	Int	10	是	是	收藏夹 ID
2	gameID	Int	10	是	是	收藏夹包含游戏 ID

8. 用户-游戏表(User2Game)

序号	名称	数据类型	大小	是否必填	是否主键	功能
1	userID	Int	10	是	是	用户 ID
2	gameID	Int	10	是	是	用户拥有游戏 ID

四、 系统重要功能及实现方法

在本次实验中，我们主要实现了对各个表的增、查、删、改功能。除此之外，我们还引进了游戏推荐系统，并利用触发器，更好地完善上述功能。因涉及的功能较多，本部分只对部分功能进行描述。

A. 对表的增、删、查、改功能。

1. 用户注册过程：

- 涉及基本表：用户表(User)
- 实现功能:完成用户的注册，将用户信息插入用户表中。
- 代码实现：

```python
@app.route('/userRegister', methods=['POST', 'GET'])  # 用户注册 done3
def UserRegister():
    if request.method == 'POST':
        postForm = json.loads(request.get_data(as_text=True))
        userName = postForm['userName']
        password = postForm['password']
        email = postForm['email'] if 'email' in postForm else None
        userTel = postForm['userTel'] if 'userTel' in postForm else None
        if not session.query(User).filter(User.userName == userName).all():  # 未注册
            user = User(userName=userName, userPassword=password)
            if email:
                user.email = email
            if userTel:
                user.userTel = userTel
            user.save()
            userID = str(user.userID)
            return json.dumps({  # 返回json
                "userID": int(userID)
            }, ensure_ascii=False)
        else:
            print("name repeat")
            return "RepeatName", 400
```

- 流程解析：先从前端以 json 的形式获取用户注册信息，先查询表中是否已有该用户，若不存在，则保存用户非空信息。

注：厂商注册过程同理。

2. 用户登录过程：

- 涉及基本表：用户表(User)
- 实现功能:查询用户表中内容，完成用户的登录，返回给前端 token。
- 代码实现：

```python
@app.route('/userLogin', methods=['POST', 'GET'])  # 查询用户表的某个用户 done3
def UserLogin():
    if request.method == 'POST':
        postForm = json.loads(request.get_data(as_text=True))
        userName = postForm['userName']
        userPassword = postForm['userPassword']
        if not session.query(User).filter(User.userName == userName).all():  # 未注册
            print("NOT REGISTER")
            return "NotRegister", 400  # 还未注册
        else:
            ans = session.query(User.userPassword).filter(User.userName == userName).first()
            if ans[0] == userPassword:
                userID = getUserID(userName)
                if userID not in com.User2Map:
                    com.Game2Time.clear()
                    com.User2Map[userID] = com.Game2Time
                else:
                    com.Game2Time = com.User2Map[userID]
                # 设置返回
                return json.dumps({
                    'userID': userID
                }, ensure_ascii=False)
            else:
                print("WRONG PASSWORD")
                return "WrongPassWord", 400
```

• 流程解析：先从前端以 json 的形式获取用户登录信息，查询用户表中该用户的信息，若不存在，则返回未注册的信息，若查询到存在，则返回用户 ID 给前端作为 token，方便其后续验证。

注：厂商登录过程同理。

3. 修改用户信息：

• 涉及基本表：用户表(User)

• 实现功能:修改用户个人信息，完成上传图片等操作。

• 代码实现：

```python
@app.route('/changeUserInfor', methods=['POST', 'GET'])  # 用户信息更改 done3
def ChangeUserInfor():
    if request.method == 'POST':
        # 获取信息
        postForm = json.loads(request.get_data(as_text=True))
        userID = postForm['userID'] if 'userID' in postForm else None
        if not userID:
            "WITHOUT COOKIES", 400
        userID = int(userID)
        userGraph = postForm['userGraph'] if 'userGraph' in postForm else None
        userTel = postForm['userTel'] if 'userTel' in postForm else None
        userPassword = postForm['userPassword'] if 'userPassword' in postForm else None
        email = postForm['email'] if 'email' in postForm else None
        userName = postForm['userName'] if 'userName' in postForm else None
        # 图片
        user = session.query(User).filter(User.userID == userID).first()
        if userGraph:
            user.userGraph = userGraph
        if userTel:
            user.userTel = userTel
        if userPassword:
            user.userPassword = userPassword
        if email:
            user.email = email
        if userName:
            user.userName = userName
        user.save()
        return "SUCCESS"
```

• 流程解析：先从前端以 json 的形式获取用户 token 和其余要修改的信息。Token 用来验证用户身份，保证数据库的安全问题。其余信息分别被验证是否为空，若不为空，则对数据库进行修改、保存。

注：厂商修改个人信息、游戏修改信息过程同理。

4. 上传用户/游戏/厂商图片：

• 实现功能:对前端上传图片的响应，将图片根据相对地址保存到静态文件夹下 static,并将图片地址返回给前端。若出现不同图片使用同一名字，则将图片文件名进行修改，再给予返回。

• 代码实现：

```python
@app.route('/uploadGameGraph', methods=['POST', 'GET'])  # 上传缩略图片 done3
def UploadGameGraph():
    if request.method == 'POST':
        img = request.files.get('gameGraph')
        if img:
            if not os.path.exists(gameGraphPath):
                os.makedirs(gameGraphPath)
            uploadPath = os.path.join(basepath, gameGraphPath, (img.filename))
            if os.path.exists(uploadPath):
                tmp = (img.filename).split('.')
                ticks = time.time()
                img.filename = tmp[0] + str(int(ticks)) + '.' + tmp[1]
                uploadPath = os.path.join(basepath, gameGraphPath, (img.filename))
            img.save(uploadPath)
            uploadPath = '.' + uploadPath[8:]
            return json.dumps({
                'gameGraphPath': uploadPath
            }, ensure_ascii=False)
        else:
            return "NO GRAPH", 400
```

• 流程解析：先从前端以 file 的形式接受图片，判断是否存在图片所需要的文件夹路径，若不存在则创建。再判断图片名称是否重复，若重复，则在其结尾加上时间戳，来避免重复命名。将图片保存到本地地址后，返回相对路径给前端。

5. **获得用户信息：**

• 涉及基本表：用户表(User)

• 实现功能:根据前端发送的 token,获取用户个人信息用于展示。涉及对表的查询。

• 代码实现：

```python
@app.route('/getUserTable', methods=['POST', 'GET'])  # 根据ID返回全部用户表内容 done3
def GetUserTable():
    if request.method == 'POST':
        postForm = json.loads(request.get_data(as_text=True))
        userID = postForm['userID'] if 'userID' in postForm else None
        if not userID:
            return 'WITHOUT COOKIES', 400
        userID = int(userID)
        user = session.query(User).filter(User.userID == userID).first()
        ansList = {
            'userName': user.userName,
            'email': user.email,
            'userTel': user.userTel,
            'userGraph': user.userGraph
        }
        return json.dumps(ansList, ensure_ascii=False);
```

• 流程解析：先从前端以 json 的形式获取用户 token。Token 用来验证用户身份，保证数据库的安全问题。再用 userID 来对 User 表进行查询，最后以

json 形式返回其个人信息。

注：厂商信息查询、游戏信息查询过程同理。

6. 添加订单：

• 涉及基本表: 用户表(User)，订单表(buy),游戏表(Game)

• 实现功能:实现用户对一游戏添加订单的操作，方便后续确认订单实现购买。 包含对用户表、游戏表的查询，和对订单表的插入。

• 代码实现：

```
@app.route('/buyGame', methods=['POST', 'GET'])  # 添加订单 done3
def BuyGame():
    if request.method == 'POST':
        postForm = json.loads(request.get_data(as_text=True))
        userID = postForm['userID'] if 'userID' in postForm else None
        if not userID:
            return 'WITHOUT COOKIES', 400
        userID = int(userID)
        gameName = postForm['gameName']
        if not session.query(Game).filter(Game.gameName == gameName).first():  # 游戏不存在
            return "NOT EXIST", 400
        gameID = getGameID(gameName)
        buyTime = str(time.asctime(time.localtime(time.time())))
        status = "unconfirmed"
        if session.query(Buy).filter(and_(Buy.gameID == gameID, Buy.userID == userID)).first():  # 已经拥有了
            return "ALREADY HAVE", 400
        buy = Buy(userID=userID, gameID=gameID, buyTime=buyTime, status=status)
        buy.save()
        return json.dumps({
            "buyID": int(buy.buyID),
            "buyTime": buyTime
        }, ensure_ascii=False)
```

• 流程解析：先从前端以 json 的形式获取用户 token。Token 用来验证用户身份，再获取需要购买的游戏名称，查询游戏表获取游戏 ID，插入订单表，并使该订单的状态为未确认(unconfirmed),最终以 json 包形式返回给前端订单序号和下订单时间。

7. 确认订单：

• 涉及基本表:订单表(buy)，用户游戏表(User2Game)

• 实现功能:实现用户对订单的确认购买流程，涉及到对订单表的查询、修改和用户游戏表的插入。

• 代码实现：

```python
@app.route('/confirmGame', methods=['POST', 'GET'])  # 确认游戏 done3
def ConfirmGame():
    if request.method == 'POST':
        postForm = json.loads(request.get_data(as_text=True))
        userID = postForm['userID'] if 'userID' in postForm else None
        if not userID:
            return 'WITHOUT COOKIES', 400
        userID = int(userID)
        buyID = postForm['buyID']
        buy = session.query(Buy).filter(Buy.buyID == buyID).first()  # 获取对象
        gameID = buy.gameID
        if not buy:
            return "BUY NOT EXIST", 400
        buy.status = 'confirmed'
        buy.update()
        com.deleteGame(gameID)  # 从推荐列表中移除该游戏
        return "SUCCESS", 200
```

• 流程解析：先从前端以 json 的形式获取用户 token。Token 用来验证用户身份，再获取需要确认订单的订单号。先再订单表中查询该订单，再完成对该订单状态的修改和保存。通过触发器(见后续)完成对用户游戏表的修改。

8. **删除游戏：**

• 涉及基本表:游戏表(buy)

• 实现功能:厂商/管理员下架游戏。涉及到对表的删除操作。

• 代码实现：

```python
@app.route('/deleteGame', methods=['POST', 'GET'])  # 下架游戏 done3
def DeleteGame():
    if request.method == 'POST':
        postForm = json.loads(request.get_data(as_text=True))
        developerID = postForm['developerID'] if 'developerID' in postForm else None
        admin = postForm['adminID'] if 'admin' in postForm else None  # 管理员也可以下架
        if not developerID and not admin:
            return "WITHOUT COOKIES", 400
        developerID = int(developerID)
        gameName = postForm['gameName']
        game = session.query(Game).filter(Game.gameName == gameName).first()
        if not game:
            return "GAME NOT EXISTS", 400
        com.deleteGame(game.gameID)
        game.delete()
        return "SUCCESS", 200
```

• 流程解析:先从前端以 json 的形式获取厂商/管理员 token 来验证身份，再根据传入的待删除游戏名称在游戏表中查找游戏并删除游戏。若游戏不村咋则单独返回报错，若游戏存在则删除。

以上部分节选功能，分别体现了对基本表的增、删、查、改功能。是数据库

的重要基础功能。

B. ORM:

我们可以通过使用 SQLAlchemhy 实现用 ORM 操作数据库. 即把数据库的表与数据库中类相连。

1. 类的划分:

我们使用类来代表数据库中基本表,则再创建基本表时,我们需要创建类来实现。如下图所示

```
class Game(Father):
    __tablename__ = "Game"
    gameID = Column(Integer, primary_key=True, autoincrement=True)  # 默认不可为空
    gameName = Column(VARCHAR(50))
    gamePrice = Column(Integer)
    introduction = Column(VARCHAR(50), nullable=True)
    gradeAvg = Column(Float, nullable=True) #总评分
    gameType = Column(Enum('action', 'adventure', 'cosplay', 'simulation', 'relaxation', 'else'))
    developerID = Column(Integer)
    gameGraph = Column(VARCHAR(300),nullable=True)#地址
    gameDetailGraph = Column(VARCHAR(300),nullable=True) #详细地址
```

我们创建了 Game 类，继承自 Father 类。即所有表的共有父类。该类中定义了各属性的特征，如是否可以为空，数据类型，是否为主键等。

Father 类：

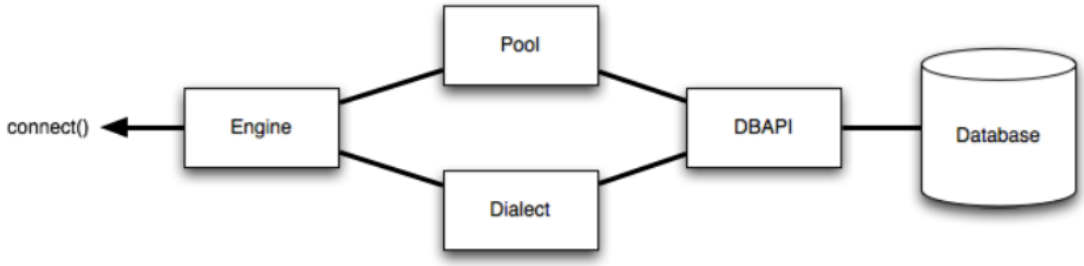
```
from Mysql import Base, session


class Father(Base):
    __abstract__ = True

    def save(self):
        # 保存对象
        session.add(self)
        # 提交事务
        session.commit()

    def update(self):
        session.commit()

    def delete(self):
        # 删除对象
        session.delete(self)
        # 提交事务
        session.commit()


Base = declarative_base()  # Base
```

上图为 Father 类的实现与其继承父类的定义。我们从 SqlAlchemy 中引用 declarative_base()方法创建基类 Base，再 Father 类中继承 Base，从而实现创建类时，完成对数据库创建基本表的操作。

除此之外，在 Father 类中，我们还定义了对表的一些基本操作，如删除、保存等。

2. Engine(引擎):

Engine 即为引擎的意思,SqlAlchemy 就通过其 Engine 来驱动。通过 Engine，我们实现了与数据库的连接。在每次发送 sql 查询的时候都需要先建立链接。Engine 包括了 Pool(连接池)与 Dialect(方言),具体连接方式如下图所示。



在代码中 Engine 实现方式如下图所示:

```
DB_URI = "mysql+pymysql://{username}:{password}@{host}:{port}/{db}?charset=utf8". \
    format(username=USERNAME, password=PASSWORD, host=HOSTNAME, port=PORT, db=DATABASE) #Connect

engine = create_engine(DB_URI, echo=False)  # 引擎
```

3. Session(会话):

Session 用于创建程序和数据库之间的会话。我们对数据库操作的提交都将通过 session 实现。所有对象的载入和保存也需要通过 session 对象。

需要注意的是，session 不是线程安全的，在多线程情况下，多个线程共享 session，就需要调用 scoped_session 方法来保证线程安全。

Session 的创建如下图所示:

```
Session = scoped_session(sessionmaker(bind=engine)  )# 会话
```

我们采用单例模式，完成对与数据库的交互。其代码如下:

```
session = Session()  # 实例化
```

Session 可以实现我们所需的对数据库基本表的增、删、查、改等功能。在每次实现功能后，调用 session.commit()即可提交数据库。具体功能可如下图所示:

增:

```
buy = Buy(userID=userID, gameID=gameID, buyTime=buyTime, status=status)
buy.save()
```

```
def save(self):
    # 保存对象
    session.add(self)
    # 提交事务
    session.commit()
```

我们创建新的类，并通过 Father 类中函数，继承 session.add()、session.commit()功能完成对数据库的增添。

删:

```
def delete(self):
    # 删除对象
    session.delete(self)
    # 提交事务
    session.commit()
```

同样的，我们采用 session.delete(),session.commit()实现对数据库基本表的删除操作。

查:

```
session.query(Buy).filter(and_(Buy.gameID == gameID, Buy.userID == userID)).first():
```

我们通过 Session.query()确定要查询的对象/基本表，.filter()确定查询条件，.first()则返回第一个符合条件的类/类中属性。

C. 触发器：

触发器（trigger）是 SQL server 提供给程序员和数据分析员来保证数据完整性的一种方法，它是与表中事件相关的特殊的存储过程，它的执行不是由程序调用，也不是手工启动，而是由事件来触发。使用触发器，可以简化我们的程序、提升运行效率。在本次实验中我们也同样大量用到了触发器，具体如下:

1. afterDeleteBuy(删除订单后的触发器):

- 涉及基本表:订单表(buy)，用户-游戏表(User2Game)
- 实现功能:删除订单时，一同删除 User2Game 中游戏与玩家的关系。
- 代码实现：

```
def afterDeleteBuy():
    session.execute("DROP TRIGGER IF EXISTS afterDeleteBuy")
    session.commit()
    afterDeleteBuy = "CREATE TRIGGER afterDeleteBuy before DELETE " \
                     "ON Buy FOR EACH ROW " \
                     "BEGIN " \
                     "DELETE FROM User2Game WHERE gameID = old.gameID and userID = old.userID; " \
                     "END;"

    session.execute(afterDeleteBuy)
    session.commit()
```

2. afterDeleteGame(删除游戏后的触发器):

•涉及基本表:游戏表 (Game),用户-游戏表(User2Game),评论表(comment),收藏夹-游戏表(Favorite2Game)

• 实现功能:删除游戏时,一同删除所有表中与该游戏有关的行。

• 代码实现:

```
def afterDeleteGame():
    session.execute("DROP TRIGGER IF EXISTS afterDeleteGame")
    session.commit()
    afterDeleteGame = "CREATE TRIGGER afterDeleteGame before DELETE " \
                      "ON GAME FOR EACH ROW " \
                      "BEGIN " \
                      "DELETE FROM Buy WHERE gameID = old.gameID ; " \
                      "DELETE FROM Comment WHERE gameID = old.gameID ; " \
                      "DELETE FROM Favorite2Game WHERE gameID = old.gameID ; " \
                      "DELETE FROM User2Game WHERE gameID = old.gameID ; " \
                      "END;"

    session.execute(afterDeleteGame)
    session.commit()
```

3. afterDeleteUser(删除游戏后的触发器):

•涉及基本表:用户表 (Game), 订单表(Buy),用户-游戏表(User2Game), 评论表(comment),收藏夹-游戏表(Favorite2Game)

• 实现功能:删除用户时,一同删除所有表中与该用户有关的行。

• 代码实现:

```
def afterDeleteUser():
    session.execute("DROP TRIGGER IF EXISTS afterDeleteUser")
    session.commit()
    afterDeleteUser = "CREATE TRIGGER afterDeleteUser before DELETE " \
                      "ON User FOR EACH ROW " \
                      "BEGIN " \
                      "DELETE FROM Buy WHERE userID = old.userID ; " \
                      "DELETE FROM Comment WHERE userID = old.userID ; " \
                      "DELETE FROM Favorite WHERE userID = old.userID ; " \
                      "DELETE FROM User2Game WHERE userID = old.userID ; " \
                      "END;"

    session.execute(afterDeleteUser)
    session.commit()
```

4. afterDeleteDeveloper(删除厂商后的触发器):

• 涉及基本表:用户表 (Game)

• 实现功能:删除厂商时,一同删除游戏表中与该厂商有关的行。

• 代码实现：

```
def afterDeleteDeveloper():
    session.execute("DROP TRIGGER IF EXISTS afterDeleteDeveloper")
    session.commit()
    afterDeleteDeveloper = "CREATE TRIGGER afterDeleteDeveloper before DELETE " \
                           "ON Developer FOR EACH ROW " \
                           "BEGIN " \
                           "DELETE FROM Game WHERE developerID = old.developerID ; " \
                           "END;"

    session.execute(afterDeleteDeveloper)
    session.commit()
```

5. confirmBuy(确认订单后，给用户添加游戏)：

• 涉及基本表:用户游戏表(User2Game)

• 实现功能:用户确认订单后，为 User2Game 添加用户与游戏的关系。具体代码如下:

```
def confirmBuy():  # 再confirmGame之后增加关系到user2Game中:
    session.execute("DROP TRIGGER IF EXISTS confirmBuy")
    session.commit()
    confirmBuy = "CREATE TRIGGER confirmBuy after UPDATE " \
                 "ON buy FOR EACH ROW " \
                 "BEGIN " \
                 "IF new.status = 'confirmed' AND NOT EXISTS(SELECT * FROM user2game" \
                 " WHERE userID = new.userID AND gameID = new.gameID) THEN " \
                 "INSERT INTO user2game(userID,gameID) " \
                 "VALUES (new.userID,new.gameID) ;" \
                 "end if; " \
                 "END;"

    session.execute(confirmBuy)
    session.commit()
```

• 程序解析： 在本次触发器中，我们选择触发器状态为表的状态更改后(after UPDATE)， 先判断其更改状态是否为确认状态(Confirmed),在确定其在用户-游戏表中不存在，则在用户-游戏表中增加该用户与游戏的关系。

D. 推荐内容：

1. 介绍：

作为一款游戏商城，推荐内容自然少不了。我们对每个用户创建一个 Map，记录其对每款游戏的访问权值。最终对于不同用户，我们给予不同的推荐内容，来增加用户购买游戏的机率。

2. 访问权重：

我们通过查看用户访问的情况(即调用对应后端查询接口的次数)来基于一个权重，判单用户对某个游戏的倾向程度。

当用户访问游戏界面时，其权重加一。

```
if gameID in com.Game2Time:
    com.Game2Time[gameID] = com.Game2Time[gameID] + 1  # 浏览了一次
else:
    com.Game2Time[gameID] = 1
```

当用户对某游戏添加评论时，其权重加一。

当用户将某游戏添加至收藏夹时，其权重加二。

当用户已经购买某游戏/从收藏夹中删除该游戏/删除该游戏购买订单后，我们认为该用户对此游戏没有兴趣，清空该游戏的权重。

3. **获取结果：**

最终，我们根据用户对不同游戏的访问权重，由大到小给游戏排序，并返回结构。靠前的游戏将被放置在推荐栏第一页、并依次类推。代码展示如下图所示:

```
def getCommentList(self):
    print(self.Game2Time)
    self.commentList = self.sort_by_value(self.Game2Time)
    return self.commentList

def sort_by_value(self, d):
    items = d.items()
    backitems = [[v[1], v[0]] for v in items]
    backitems.sort(reverse=True)#从大到小排序
    return [backitems[i][1] for i in range(0, len(backitems))]  # 返回key
```

E. Token 生成与验证:

为了验证登陆用户、厂商的身份，我们通过 token 来进行验证。Token 是服务端生成的一串字符串，以作客户端进行请求的一个令牌，当第一次登录后，服务器生成一个 Token 便将此 Token 返回给客户端，以后客户端只需带上这个 Token 前来请求数据即可，无需再次带上用户名和密码。

1. **Token 生成：**

我们利用 TimedJSONWebSignatureSerializer 函数来进行加密。将用户 ID 结合密钥(secret_key)和当前时间，生成一个有效期为 60min 的密钥。生成字符串形式并返回。具体代码如下图所示:

```python
def create_token(api_user):
    # 第一个参数是内部的私钥，这里写在共用的配置信息里了，如果只是测试可以写死
    # 第二个参数是有效期(秒)
    s = Serializer(secret_key, expires_in=3600)
    # 接收用户id转换与编码
    api_user = int(api_user)
    token = s.dumps({"id": api_user}).decode("utf-8")
    return token
```

2. Token 验证：

为了破解上述生成的密钥，我们需要一个验证函数，输入密钥 secret_key 将用户 ID 以字典形式取出，并返回。从而获取用户 ID 代码如图所示

```python
def verify_token(token):
    # 参数为私有秘钥，跟上面方法的秘钥保持一致
    s = Serializer(secret_key)
    try:
        # 转换为字典
        data = s.loads(token)
    except Exception:
        return None
    # 拿到转换后的数据，根据模型类去数据库查询用户信息
    return int(data['id'])
```

五、系统实现结果

本部分将通过截图的形式展示各页面布局以及主要功能的实现结果。

一、静态页面展示

1. 登录及注册页面

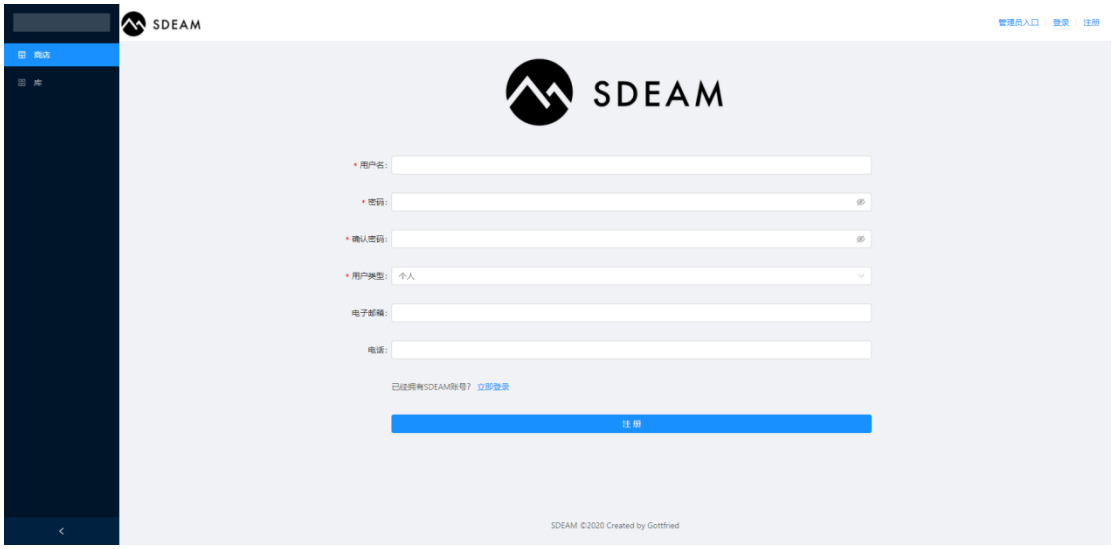


图 1 注册界面

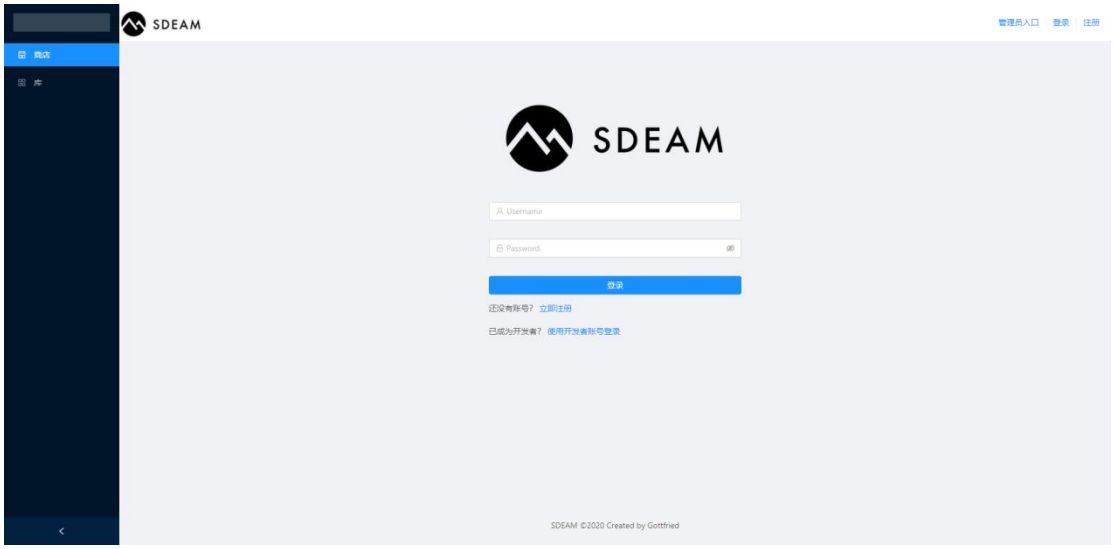


图 2 登录页面

2. 商店首页

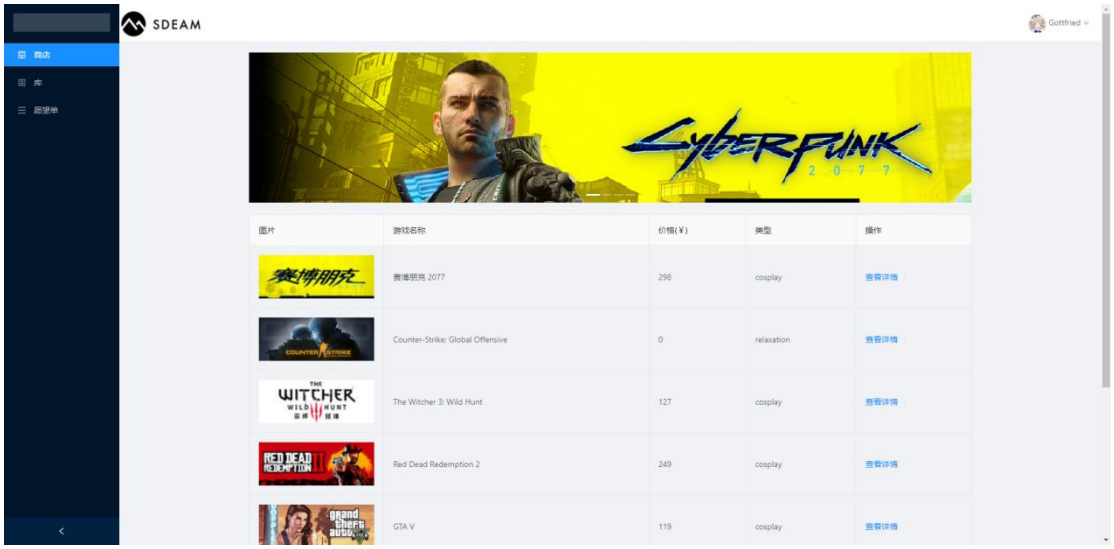


图 3 商店首页

3. 用户库存页面

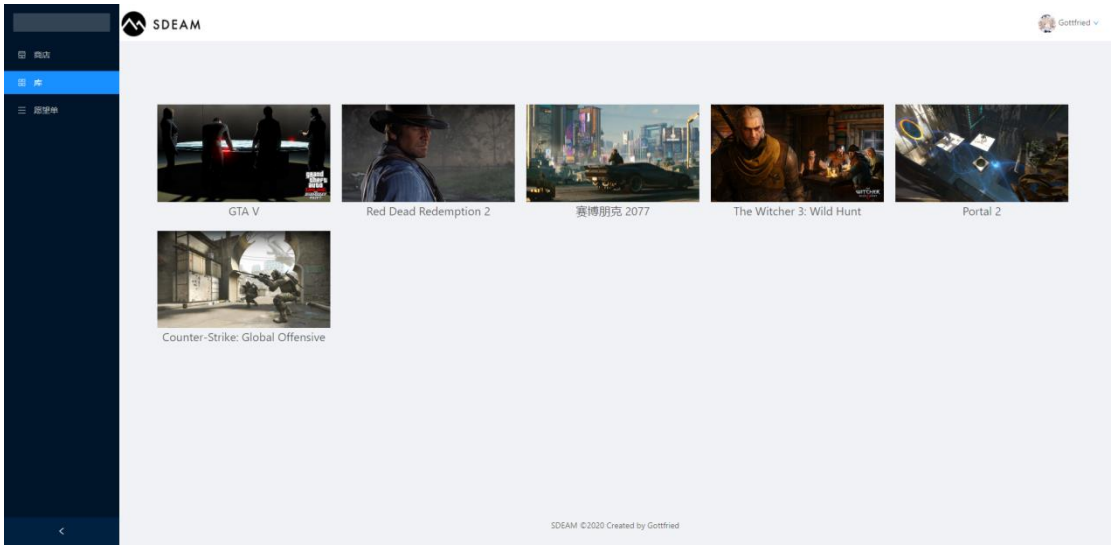


图 4 用户库存页面

4. 收藏夹首页与收藏夹内容页面

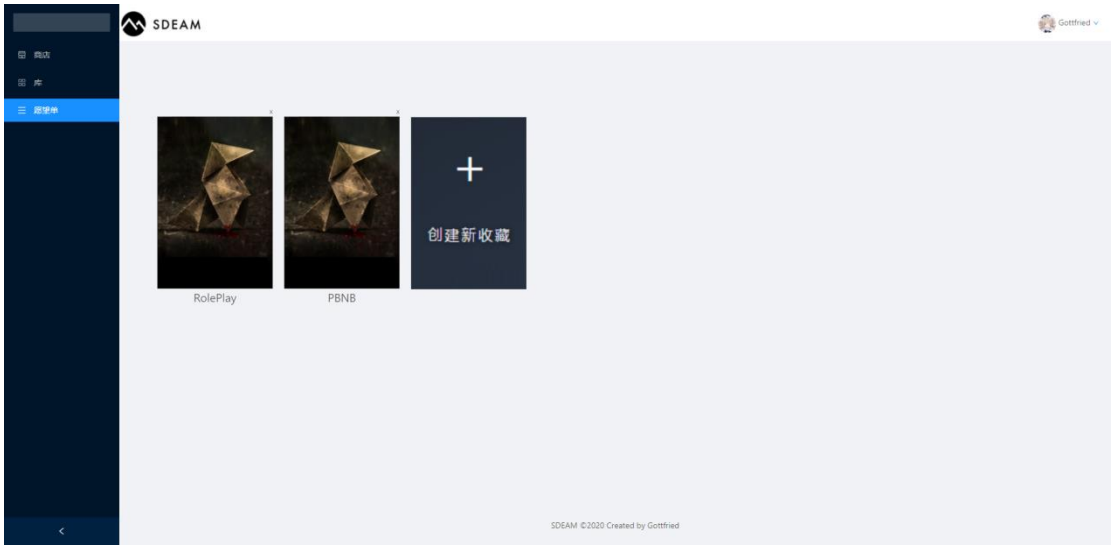


图 5 收藏夹首页

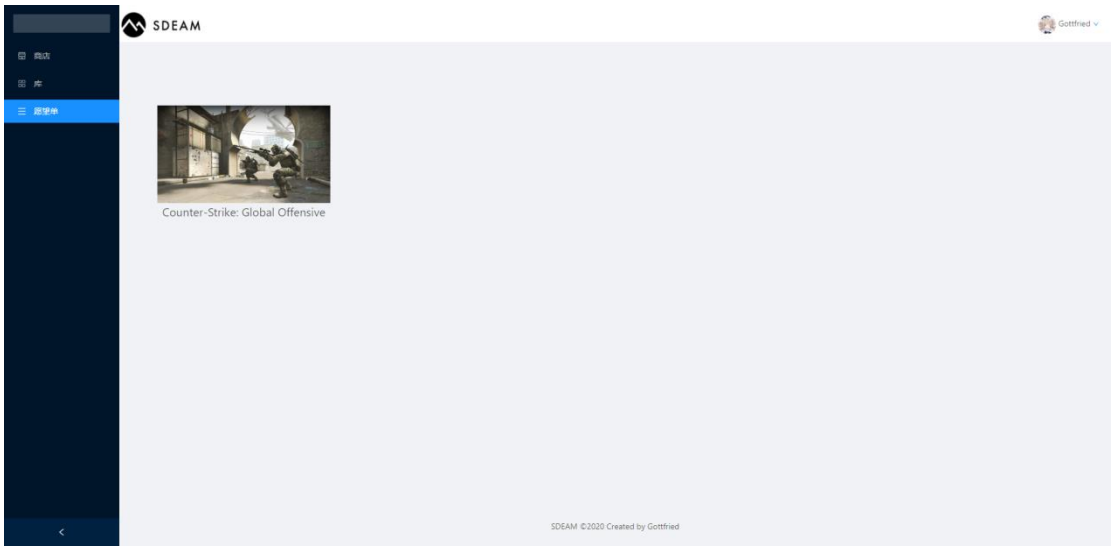


图 6 收藏夹内容页面

5. 用户信息页面

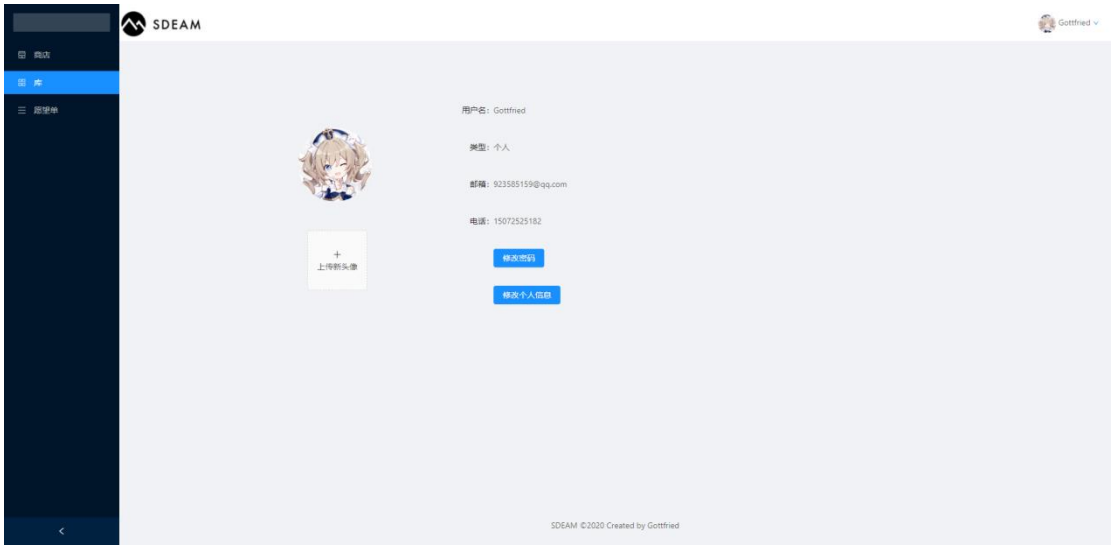


图 7 用户信息页面

6. 游戏详情页面

图 8 游戏详情页面（上）

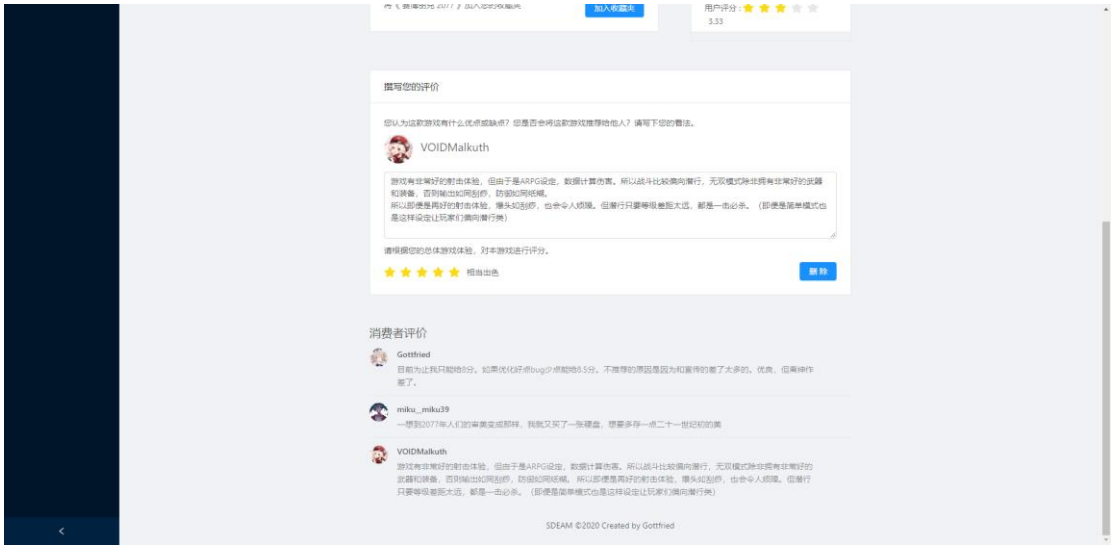


图 9 游戏详情页面（下）

二、功能实现展示

1. 用户修改信息与上传头像

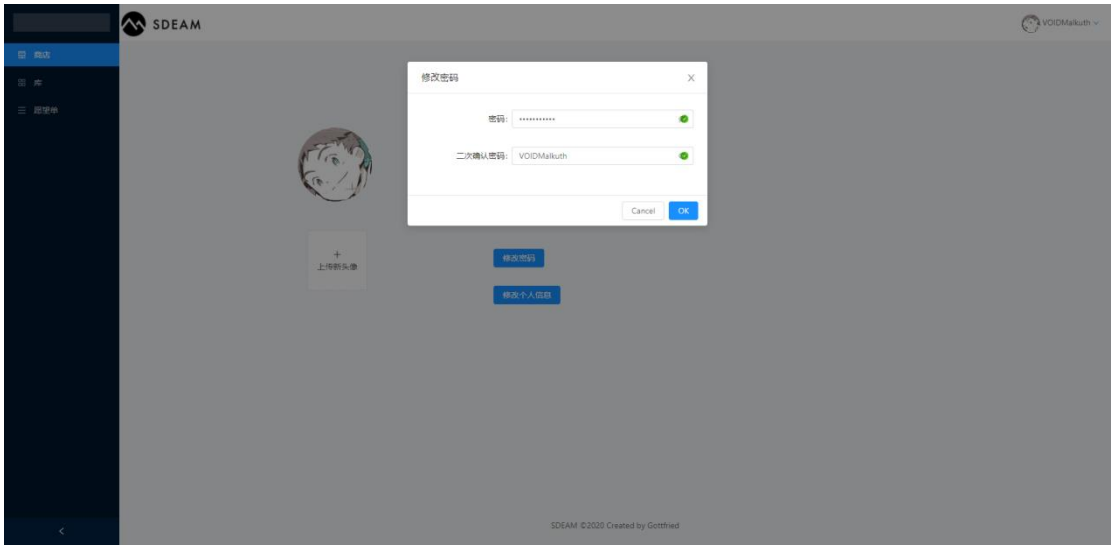


图 10 用户修改信息

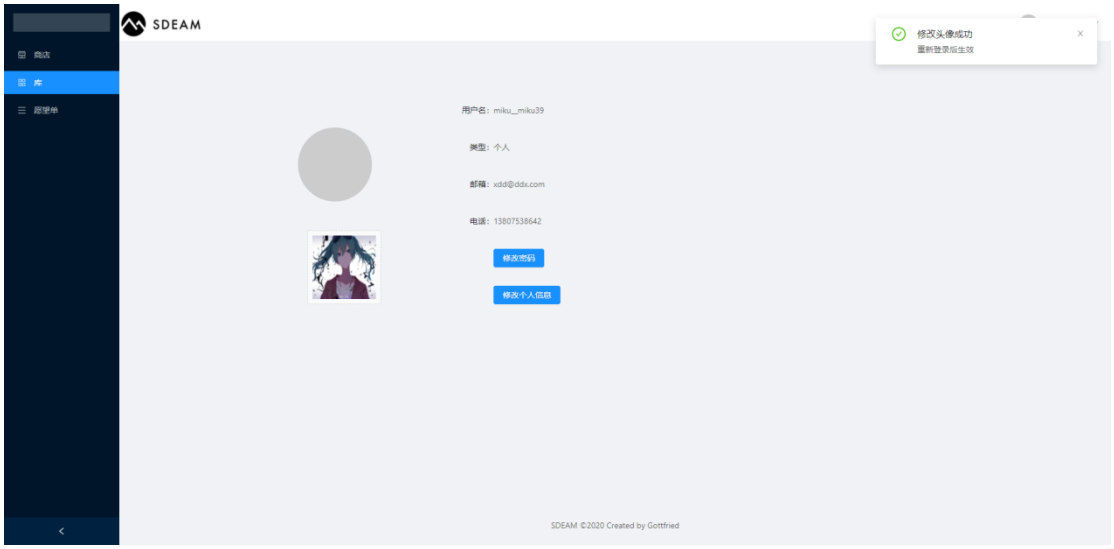


图 11 用户上传头像

2. 购买游戏与确认订单

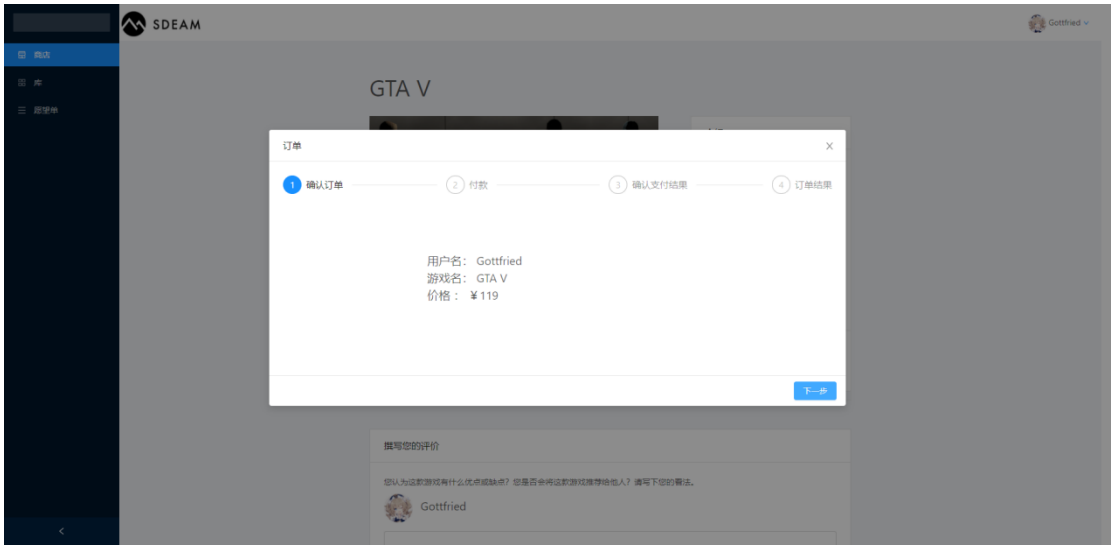


图 12 提交订单

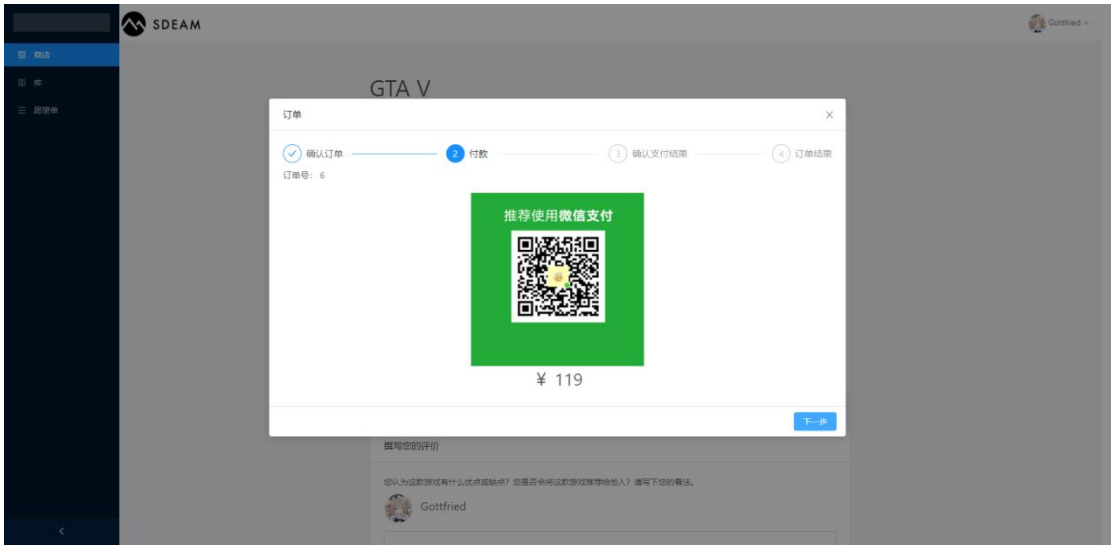


图 13 付款页面

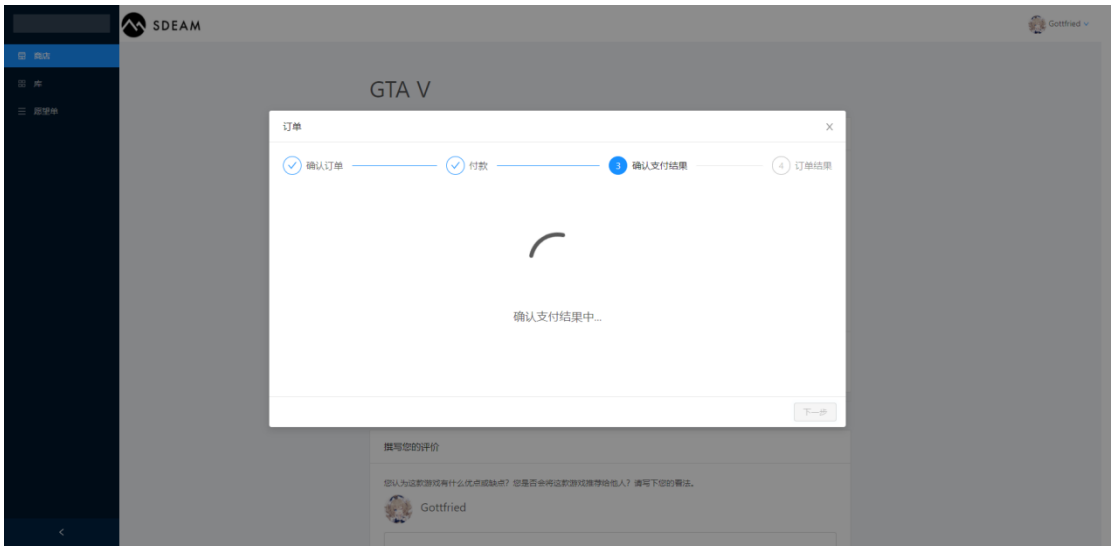


图 14 确认付款结果页面

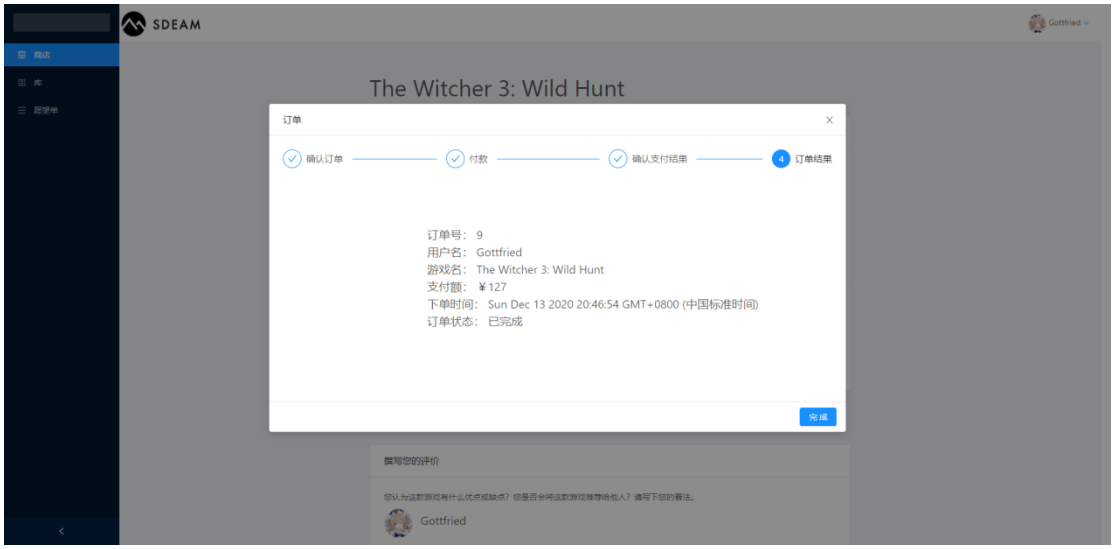


图 15 订单结果页面

3. 撰写评论及评分

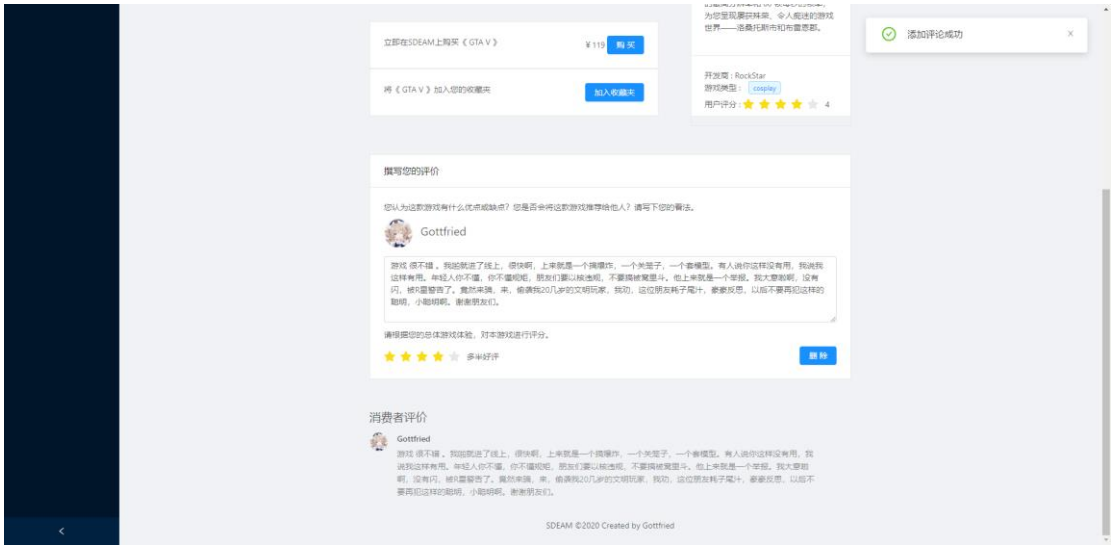


图 16 用户撰写评论与评分

4. 添加收藏夹

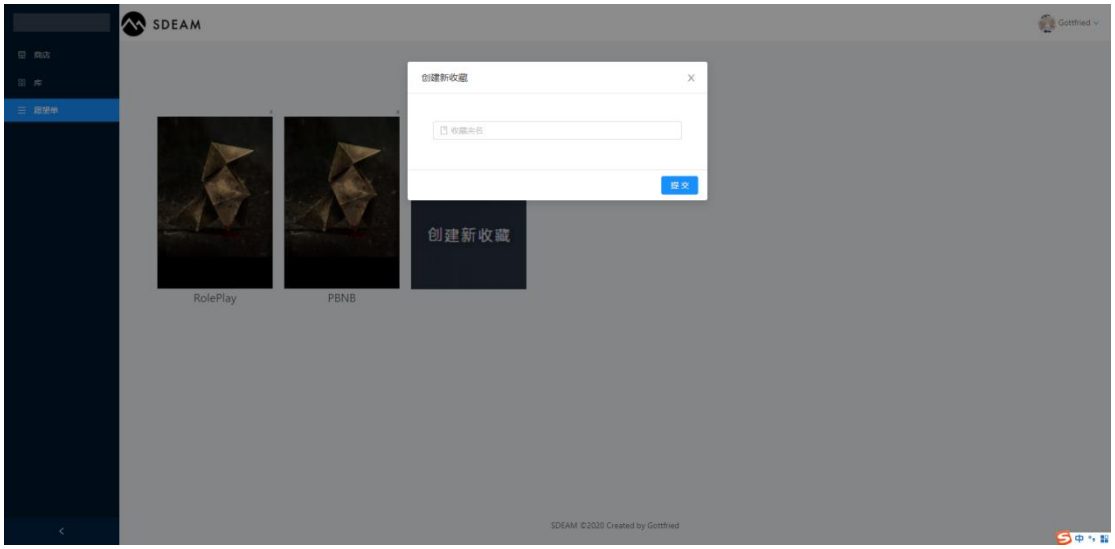


图 17 用户新建收藏夹

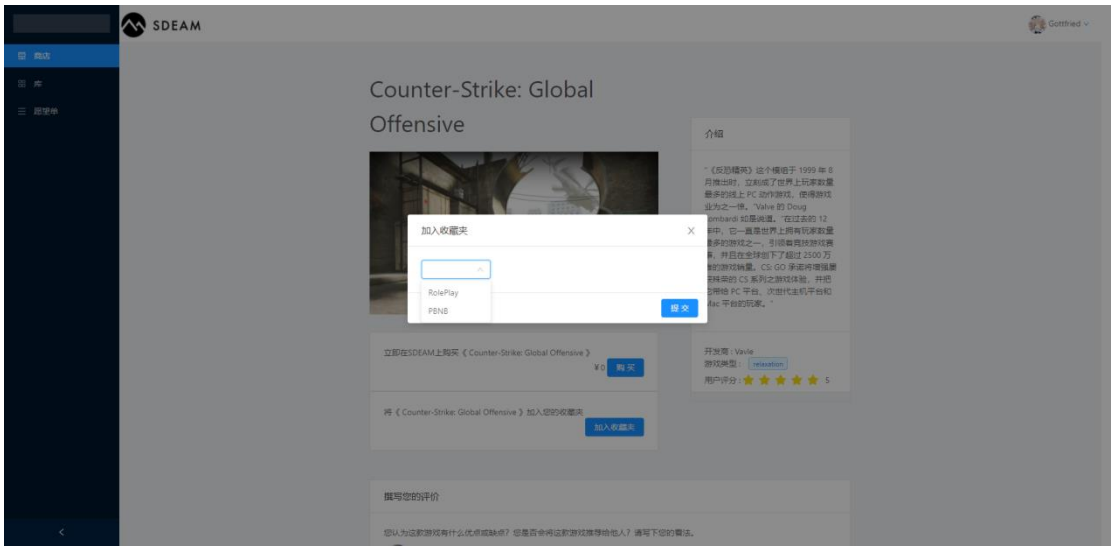


图 18 将游戏加入收藏夹

5. 开发商上传游戏

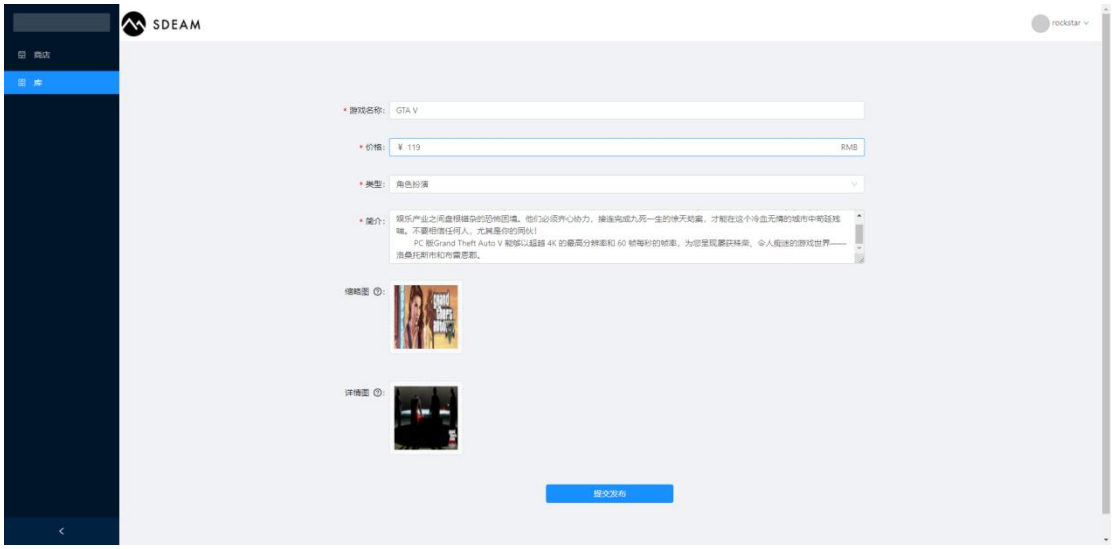


图 19 开发商上传新游戏

6. 管理员权限相关

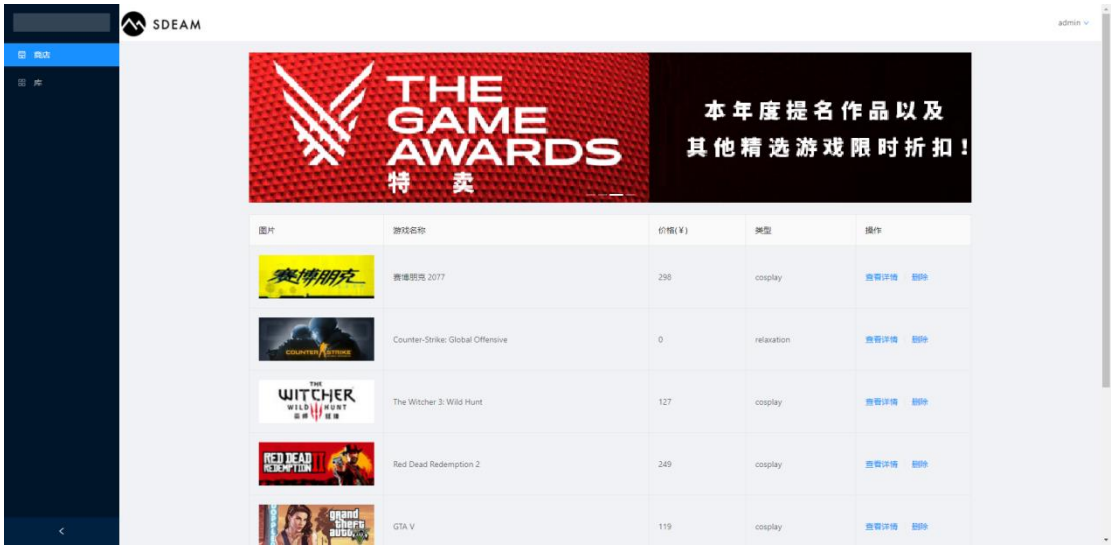


图 20 管理员视角的商店首页（可删除游戏）

图 21 管理员视角的详情页面（可删除评论）

值得注意的是，由于截图形式与篇幅的限制，本报告中并没有对大量存在的动画或提示等信息做出充分且合理的展示。

六、总结：

本次数据库课程设计作业对我们来说是一次机遇、也是一次挑战。这是我们组的同学们第一次开发一个网页，也是第一次实现 web 前端、后端、数据库系统设计。在完成作业期间，我们遇到了不少的困难，也学到了很多。

在网页开发阶段，我们就遇到了许多困难。第一次了解什么叫 web 框架，第一次本地部署前端、后端框架、第一次实现前后端依赖于 Json 的通信，第一次实现前后端文件(图片)的传输。在这期间，因为对 Web 开发的陌生、我们遇到了许多的阻碍，也尝试了许多的方法，这其中还包括不少本地调试时遇到的跨域问题。但是通过不断地查找解决方法，我们最终逐一解决了这些问题，为前后端分离开发积累了宝贵的经验。

在后端操控数据库阶段，我们也同样陌生。面对 ORM 是什么的一无所知、编写对数据库操控指令时的疯狂 debug、为了增添新的功能而不断的调试、对后端数据库的测试都是我们遇到的难题。最终，我们通过本次实验，更加深入地理解了对数据库的操作，对触发器的使用以及其效果。也通过第一次作业的手写 Sql 语句，增加了对 mysql 语句的掌握，第二次作业的 ORM，让我们认识到了全新的操控数据库的方式。

除此之外，我们通过本次实验，还学习到了团队合作的能力。一名同学负责前端编写、一名同学负责后端编写、一名同学负责设计。在这个小团队中，同学们如何合力完成目标无疑是不小的挑战。尤其是在面对前后端同学的对接、交流时候遇到的代码同步问题。为了解决这个问题，我们使用 github 来进行合作。同时我们对每个功能提前设置好接口，并规范好前后端对接的数据传输格式，最终高效、高质量的完成课设内容。

总的来说，本次实验使我们积累了宝贵的数据库设计、web 开发、团队合作的能力，除此之外还通过实践更加清晰地理解和掌握了数据库理论知识包括范式规范、sql 指令、为什么要规范范式等。虽然在实现的过程中，我们遇到了许多阻力与难题，但是在一一克服这些困难，完成整个网站开发后，我们有着巨大的喜悦与成就感。所以，可以说本次实验经历是十分重要和难得可贵的。