ФГБОУ ВО Уральский государственный горный университет

Инженерно-экономический факультет

Кафедра информатики

**Курсовой проект**

по дисциплине «Управление данными»

на тему «Торговая площадка сервиса видеоигр»

|  |  |
| --- | --- |
|  | Выполнил:  студент гр. АУБП-22-2  Писковский И. В. |
|  | Проверил:  ст. преп. каф. информатики  Рыжков Д. С. |

Екатеринбург, 2024 г.

**СОДЕРЖАНИЕ**

1. Постановка задачи………………………………………………………………………3
   1. Краткое описание задачи…………………………………………………………3
2. Проектирование системы……………………………………………………………...4
   1. Проектирование БД………………………………………………………………4
   2. Проектирование API для доступа к данным……………………………………5
3. Разработка системы…………………………………………………………………….7
   1. Выбор средств реализации………………………………………………………7
   2. Структура проекта………………………………………………………………..8
   3. Реализация………………………………………………………………………...10
4. **ПОСТАНОВКА ЗАДАЧИ**
   1. **Краткое техническое задание**

Разработать базу данных с использованием Python, ORM-фреймворков и Web API по схеме:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 1. Схема базы данных

1. **ПРОЕКТИРОВАНИЕ СИСТЕМЫ**
   1. **Проектирование БД**

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 2. Структура базы данных

* 1. **Проектирование API для доступа к данным**

Реализация Web API с маршрутами для обработки запросов к БД (листинг 1):

@app.route(‘/users’, methods=[‘POST’])

def create\_user():

return UserController.create\_user(request.get\_json())

@app.route(‘/users/<int:user\_id>’, methods=[‘PUT’])

def update\_user(user\_id):

return UserController.update\_user(user\_id, request.get\_json())

@app.route(‘/users/<int:user\_id>’, methods=[‘DELETE’])

def delete\_user(user\_id):

return UserController.delete\_user(user\_id)

@app.route(‘/users’, methods=[‘GET’])

def get\_users():

return UserController.get\_users()

@app.route(‘/transactions’, methods=[‘POST’])

def create\_transaction():

return TransactionController.create\_transaction(request.get\_json())

@app.route(‘/transactions/<int:transaction\_id>’, methods=[‘DELETE’])

def delete\_transaction(transaction\_id):

return TransactionController.delete\_transaction(transaction\_id)

@app.route(‘/transactions’, methods=[‘GET’])

def get\_transactions():

return TransactionController.get\_transactions()

Листинг 1. Реализация маршрутов Web API

1. **РАЗРАБОТКА СИСТЕМЫ**
   1. **Выбор средств реализации**

В качестве языка программирования для решения этой задачи был выбран высокоуровневый язык Python, так как в нем много встроенных полезных функций. Также имеются библиотеки, которые облегчают работу с программой. Как альтернативу Python можно было использовать C# или Java, с помощью которых тоже возможно решить поставленную задачу.

Python — это высокоуровневый язык программирования общего назначения. Он поддерживает несколько парадигм программирования, включая структурное, объектно-ориентированное и функциональное программирование. Код Python часто называют псевдокодом, поскольку он позволяет вам выражать очень мощные идеи в очень небольшом количестве кода и его синтаксис очень читаемый. Python интерпретируемый язык, что означает, что код исполняется построчно, что позволяет быстро прототипировать и тестировать идеи без затрат времени на компиляцию. Python также имеет динамическую типизацию, что позволяет работать с различными типами данных без необходимости их предварительного объявления.

Для разработки был использован Visual Studio Code (далее VS Code). VS Code — это бесплатная среда разработки от Microsoft, которая поддерживает множество языков программирования и имеет огромное количество расширений. Она поддерживает рефакторинг, профилирование, выделение синтаксических конструкций цветом, автодополнение набираемых конструкций на лету, множество предопределённых шаблонов кода и др. VS Code является бесплатным, расширяемым, простым в использовании, а также обладает понятным интерфейсом.

ORM — Object-Relational Mapping или в переводе на русский объектно-реляционное отображение. Это технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования. Она нашла воплощение в виде SQLAlchemy — это библиотека SQL для Python, которая обеспечивает полный набор высокоэффективных и высокоуровневых SQL Abstraction.

Flask — это микрофреймворк для Python, на котором пишут веб-приложения и микросервисы. Flask представляет собой набор утилит, автоматизирующих настройки фреймворка. Он легкий и простой в использовании, идеально подходит для маленьких проектов или простых API. Он также имеет большое сообщество, которое создает множество расширений, облегчающих разработку.

* 1. **Структура проекта**

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 3. Структура проекта

Изображение выглядит как текст, снимок экрана, Параллельный

Автоматически созданное описание

Рис. 4. Дерево классов

* 1. **Реализация**

Реализация создания таблиц описана в листинге 2, реализация функционала для таблицы пользователей описана в листинге 3, для таблицы транзакций в листинге 4, реализация маршрутов для обработки описана в листинге 5.

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///database.db'

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(50), nullable=False)

email = db.Column(db.String(50), nullable=False)

password = db.Column(db.String(50), nullable=False)

balance = db.Column(db.Float, nullable=False)

transactions = db.relationship('Transaction', backref='user')

def to\_dict(self):

return {c.name: getattr(self, c.name) for c in self.\_\_table\_\_.columns}

class Transaction(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

seller\_id = db.Column(db.Integer, nullable=False)

item\_name = db.Column(db.String(100), nullable=False)

item\_price = db.Column(db.Float, nullable=False)

transaction\_type = db.Column(db.String(50), nullable=False)

transaction\_date = db.Column(db.DateTime, default=datetime.utcnow)

def to\_dict(self):

return {c.name: getattr(self, c.name) for c in self.\_\_table\_\_.columns}

Листинг 2. Инициализация БД и создание моделей

class UserController:

@staticmethod

def create\_user(data):

new\_user = User(username=data['username'], email=data['email'], password=data['password'], balance=data['balance'])

db.session.add(new\_user)

db.session.commit()

return {'id': new\_user.id}, 201

@staticmethod

def update\_user(user\_id, data):

user = User.query.get\_or\_404(user\_id)

user.username = data.get('username', user.username)

user.email = data.get('email', user.email)

user.password = data.get('password', user.password)

user.balance = data.get('balance', user.balance)

db.session.commit()

return {'message': 'User updated successfully'}

@staticmethod

def delete\_user(user\_id):

user = User.query.get\_or\_404(user\_id)

db.session.delete(user)

db.session.commit()

return {'message': 'User deleted successfully'}

@staticmethod

def get\_users():

users = User.query.all()

return jsonify([user.to\_dict() for user in users])

Листинг 3. Реализация контроллера модели пользователей

class TransactionController:

@staticmethod

def create\_transaction(data):

buyer = User.query.get\_or\_404(data['user\_id'])

seller = User.query.get\_or\_404(data['seller\_id'])

if buyer.balance < data['item\_price']:

return {'message': 'Insufficient balance'}, 400

new\_transaction = Transaction(user\_id=data['user\_id'], seller\_id=data['seller\_id'], item\_name=data['item\_name'], item\_price=data['item\_price'], transaction\_type=data['transaction\_type'])

buyer.balance -= data['item\_price']

seller.balance += data['item\_price']

db.session.add(new\_transaction)

db.session.commit()

return {'id': new\_transaction.id}, 201

@staticmethod

def delete\_transaction(transaction\_id):

transaction = Transaction.query.get\_or\_404(transaction\_id)

db.session.delete(transaction)

db.session.commit()

return {'message': 'Transaction deleted successfully'}

@staticmethod

def get\_transactions():

transactions = Transaction.query.all()

return jsonify([transaction.to\_dict() for transaction in transactions])

Листинг 4. Реализация контроллера модели транзакций

@app.route('/users', methods=['GET'])

def get\_users():

return UserController.get\_users()

@app.route('/users', methods=['POST'])

def create\_user():

return UserController.create\_user(request.get\_json())

@app.route('/users/<int:user\_id>', methods=['PUT'])

def update\_user(user\_id):

return UserController.update\_user(user\_id, request.get\_json())

@app.route('/users/<int:user\_id>', methods=['DELETE'])

def delete\_user(user\_id):

return UserController.delete\_user(user\_id)

@app.route('/transactions', methods=['GET'])

def get\_transactions():

return TransactionController.get\_transactions()

@app.route('/transactions', methods=['POST'])

def create\_transaction():

return TransactionController.create\_transaction(request.get\_json())

@app.route('/transactions/<int:transaction\_id>', methods=['DELETE'])

def delete\_transaction(transaction\_id):

return TransactionController.delete\_transaction(transaction\_id)

Листинг 5. Реализация маршрутов