**Национальный исследовательский ядерный университет «МИФИ»**

|  |  |
| --- | --- |
|  | **Институт интеллектуальных и кибернетических систем**  **Кафедра «Компьютерные системы и технологии»** |

Пояснительная записка

к проекту по курсу «Программирование

сетевых приложений»

на тему «Разработка веб-приложения для учета финансов»

Студент гр. Б20-513 Ханбекова Ю. Н. /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Руководитель Овчаренко Е. С. / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Москва, 2023г.

**АННОТАЦИЯ**

Пояснительная записка 27 с., 11 рис., 2 табл., 8 ист.

Объектом разработки является приложение для учета финансов.

Целью работы является разработка серверной и клиентской части веб – приложения «Finspect», которое предназначено для отслеживания личных финансов, включая учет доходов и расходов. Данная разработка направлена на упрощение и автоматизацию управления финансами пользователя.

Серверная часть приложения реализована на языке программирования Python с использованием фреймворка Flask. Для создания клиентской части веб-приложения использовались три основные веб-технологии: HTML, CSS и JavaScript. Для работы с базой данных была выбрана СУБД PostgreSQL.

**Содержание**

[**Введение** 4](#_Toc154793669)

[**1.** **Теоретическая часть** 5](#_Toc154793670)

[**1.1.** **Постановка задачи** 5](#_Toc154793671)

[**1.2.** **Схема приложения** 5](#_Toc154793672)

[**1.3.** **Формализация данных** 5](#_Toc154793673)

[**2.** **Практическая часть** 7](#_Toc154793674)

[**2.1.** **Используемые технологии** 7](#_Toc154793675)

[**2.2.** **Описание таблиц базы данных** 7](#_Toc154793676)

[**2.3.** **Уровень доступа к данным** 8](#_Toc154793677)

[**2.4.** **Аутентификация пользователей** 9](#_Toc154793678)

[**3.** **Интерфейс** 10](#_Toc154793679)

[**3.1.** **Авторизация и регистрация** 10](#_Toc154793680)

[**3.2.** **Боковая панель** 11](#_Toc154793681)

[**3.3.** **Счет** 12](#_Toc154793682)

[**3.4.** **Статистика** 13](#_Toc154793683)

[**3.5.** **История операций** 13](#_Toc154793684)

[**3.6.** **Календарь** 14](#_Toc154793685)

[**3.7.** **Копилка** 15](#_Toc154793686)

[**Заключение** 16](#_Toc154793687)

[**Список используемых источников** 17](#_Toc154793688)

[**Приложение А «Листинги основных контроллеров»** 18](#_Toc154793689)

# **Введение**

Проблема отслеживания финансов и эффективного контроля за расходами актуальна на сегодняшний день. Неизменно растущая сложность финансовых транзакций и индивидуальных бюджетов требует инновационных решений для обеспечения пользовательского комфорта. Для некоторых людей использование учета финансов нужно для разных целей: уменьшить расходы и понять, где их можно сократить, выплатить кредиты или спланировать даты необходимых платежей или поступлений Правильный менеджмент денег позволит улучшить качество жизни и увеличить накопления.

В процессе работы по курсу «Программирование сетевых приложений» была поставлена задача выполнения проекта, целью которого является реализация веб - приложения для учета финансов пользователя.

Данное приложение будет использоваться открытым кругом лиц для упрощения планирования их бюджета, достижения финансовых целей, расчета дохода и расхода, а также для наглядного анализа расходов с учетом их категоризации.

# **Теоретическая часть**

* 1. **Постановка задачи**

Данное веб – приложение будет иметь следующий функционал:

* Регистрация и аутентификация пользователя
* Ввод или удаление финансовых операций (доходов и расходов) с указанием даты, суммы, категории
* Категоризация трат
* Отображение статистики о финансовом состоянии по периодам и счетам (график, выписка расходов/доходов)
* История операций
* Календарь расходов
* Установка цели и планирование бюджета, накопление средств
* Управление счетами (создание; редактирование названия; удаление счета)
  1. **Схема приложения**

Клиент-серверное приложение состоит из нескольких связанных частей. Схема взаимодействия между частями приложения представлена на рисунке 1.

«Клиент — сервер» - вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Пользователь через веб - приложение может совершать действия, которые будут обрабатываться сервером и сохраняться в базе данных.

* 1. **Формализация данных**

База данных должна хранить данные о следующих сущностях, представленных в таблице 1.

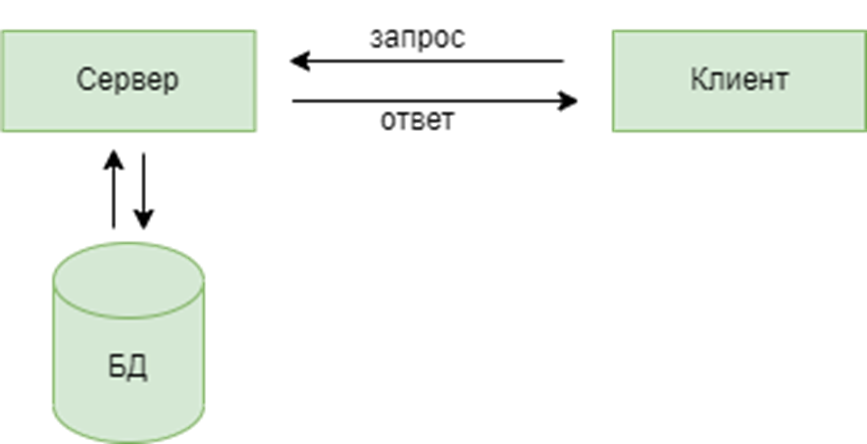


Рисунок 1. Схема клиент-серверной архитектуры

Таблица 1. Сущности базы данных

|  |  |
| --- | --- |
| **Сущности** | **Сведения** |
| Пользователь | Логин, хешированный пароль |
| Счет | Имя, баланс |
| Транзакции | Сумма, дата, доход/расход, категория |
| Категории | Имя |
| Цели | Имя, желаемая сумма, дата окончания, достигнута/не достигнута |
| Накопления | Сумма, дата |

1. **Практическая часть**
   1. **Используемые технологии**

В качестве языка программирования для разработки серверной части приложения был выбран язык программирования Python, поскольку он поддерживает парадигму объектно-ориентированного программирования, а также поддерживает возможность работы с фреймворками, такими как Flask [1].

В качестве базы данных используется PostgreSQL [2], как самая распространённая, надежная и быстрая СУБД с открытым исходным кодом.

Для разработки пользовательского интерфейса было принято решение использовать HTML, CSS, JavaScript, Jinja2 [3]. Jinja2 – удобный шаблонизатор для языка программирования Python [4]. Он предоставляет простой синтаксис и расширенные возможности для генерации динамически обновляемого контента на странице.

* 1. **Описание таблиц базы данных**

Чтобы организовать хранение информации о пользователях, счетах и транзакциях используется база данных PostgreSQL. Структура базы данных представлена ниже.

Сущность User – в ней хранятся id пользователя, логин, хэшированный пароль.

Сущность Account – в ней хранятся id счета, имя, баланс (указывается в рублях) и id пользователя, которому принадлежит счет.

Сущность FinancialTransaction– в ней хранится id транзакции, id пользователя, которому принадлежит транзакция, id счета, на котором происходило выполнение транзакции, id категории, к которой относится данная транзакция, сумма, дата, доход/расход.

Сущность Category – в ней хранится id, имя.

Сущность Goals– в ней хранится id заданной цели, id пользователя, которому принадлежит заданная цель, имя цели, сумма, дата начала, дата окончания, достигнута/не достигнута.

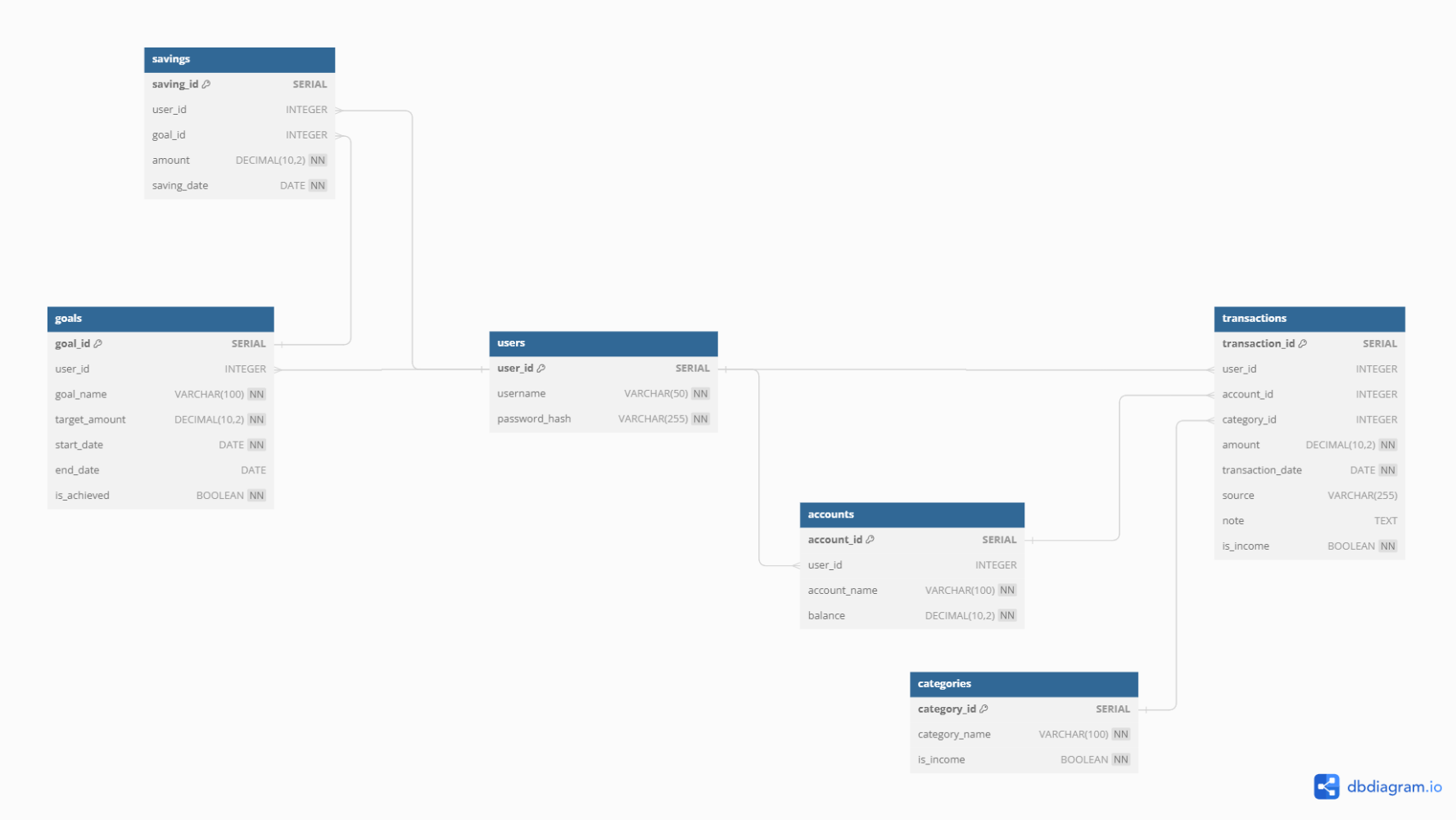
Сущность Savings – в ней хранится id накопления, id пользователя, которому принадлежит данное накопление, id цели, в рамках которой происходит накопление, сумма, дата.

Рисунок 2. Структура базы данных

* 1. **Уровень доступа к данным**

Запросы к базе данных осуществляются через сессии SQLAlchemy. SQLAlchemy это библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM. Служит для синхронизации объектов Python и записей реляционной базы данных. SQLAlchemy позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python без использования SQL.

Flask — это фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков, минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

В Flask-SQLAlchemy [5], уровень доступа к данным осуществляется через использование моделей данных. Модель данных — это класс Python, который отображает структуру таблицы в базе данных. Эти классы определяются как обычные классы Python и используют SQLAlchemy для отображения и взаимодействия с базой данных.

* 1. **Аутентификация пользователей**

Аутентификация пользователя - важнейший аспект веб-разработки, особенно когда речь идет о защите учетных записей пользователей и конфиденциальной информации. Для безопасного хранения пароля пользователя было использовано хэширование пароля с помощью библиотеки языка программирования Python – Bcrypt [6]. Bcrypt [7], специально разработанный для хэширования паролей, является популярным выбором из-за его устойчивости к атакам методом перебора и адаптивного характера. Bcrypt автоматически корректирует свои вычислительные затраты с течением времени, что делает его надежным решением с учетом меняющихся аппаратных возможностей.

# **Интерфейс**

* 1. **Авторизация и регистрация**

На рисунке 3 приведена форма создания профиля(регистрации). При регистрации проверяется уникальность имени пользователя(логина). Пароль должен удовлетворять следующим критериям:

1. Должен содержать больше 5 символов
2. Должен состоять из букв и цифр

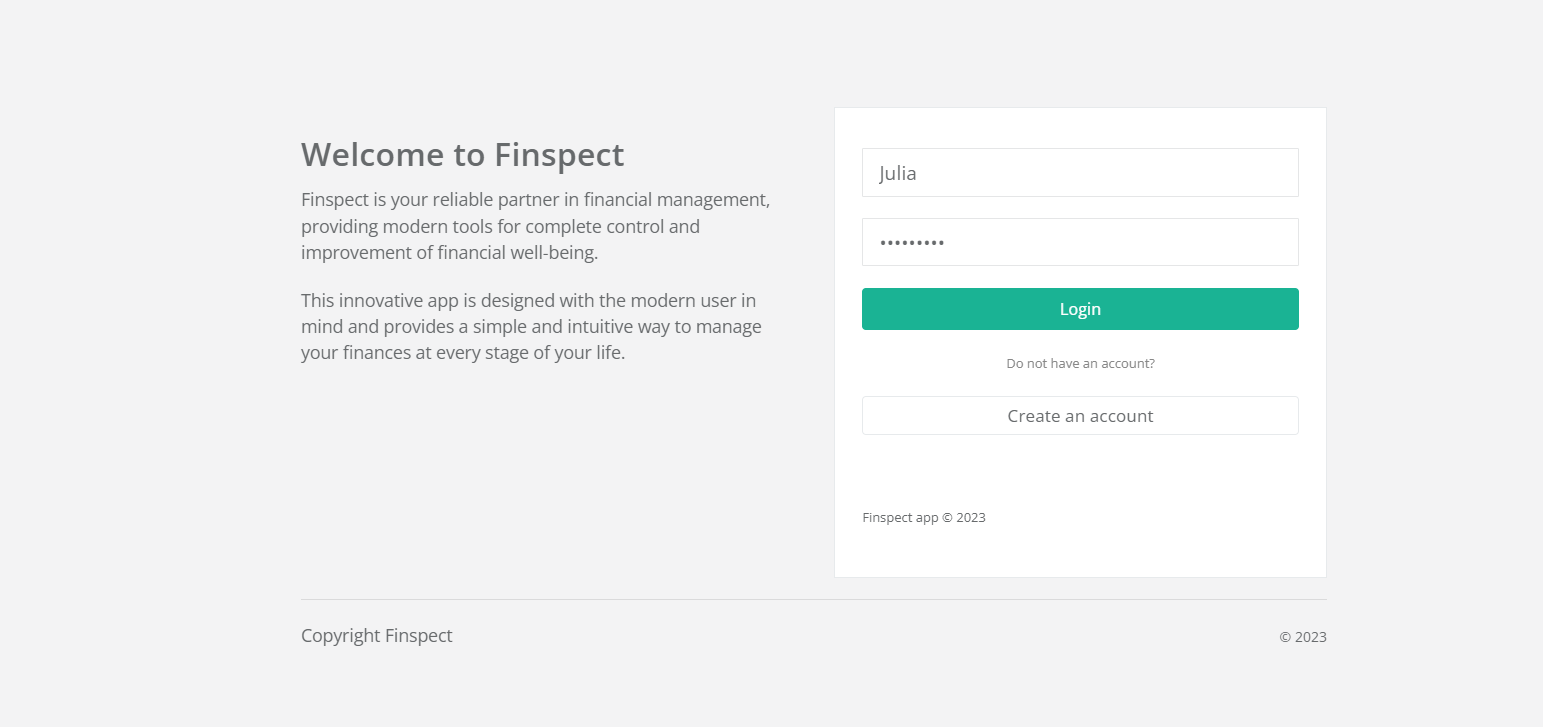
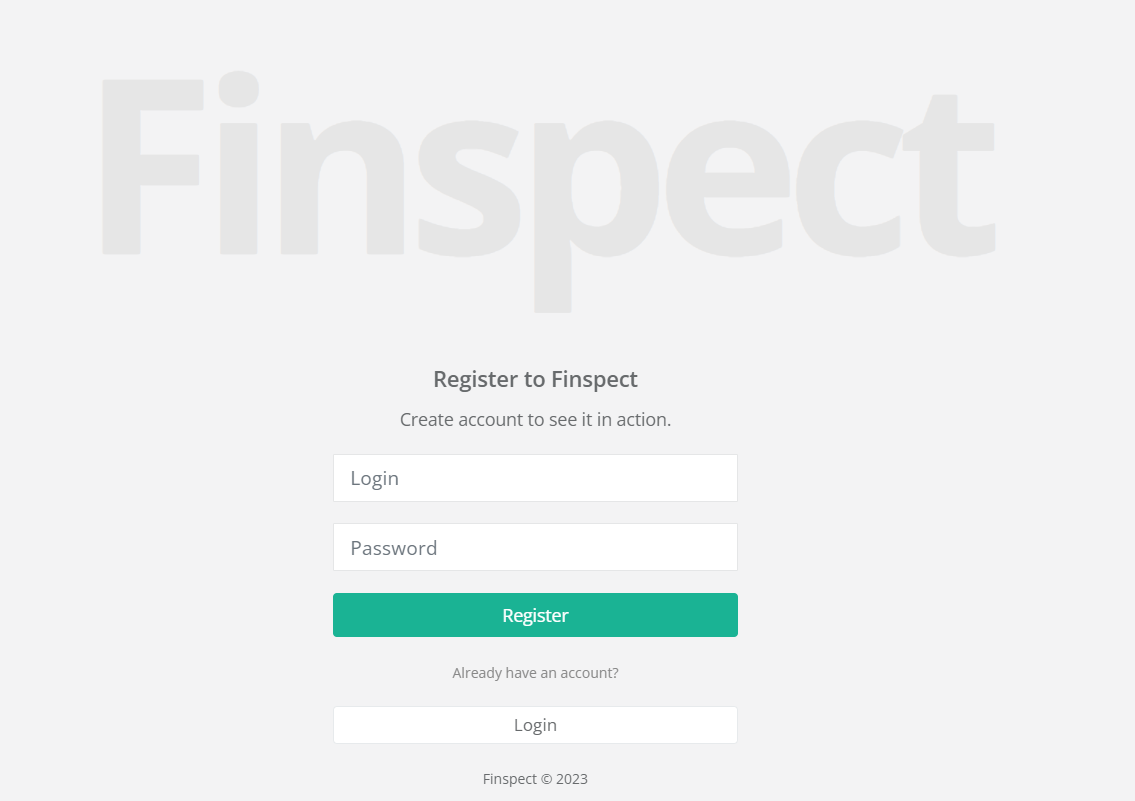
****При несоблюдении правил создания пароля пользователь перенаправляется на страницу регистрации с указанием ошибки “Password should be longer than 5 characters.” или “Password should contain both letters and numbers.”

Рисунок 4. Форма авторизации пользователя

Рисунок 3. Форма регистрации пользователя

При неверно введенных данных авторизации пользователь перенаправляется снова на страницу авторизации с сообщением “Login failed. Check your username and password.”

* 1. **Боковая панель**

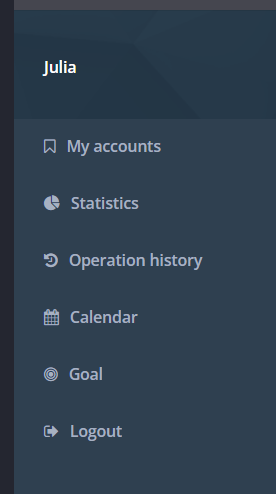
Приложение предусматривает боковую панель с необходимыми вкладками, изображенную на рисунке 5.

Рисунок 5. Боковая панель

Подробнее функционал вкладок описан в таблице 2.

Таблица 2. Элементы боковой панели

|  |  |
| --- | --- |
| **Панель** | **Сведения** |
| My accounts | Счета пользователя. Предусматривается добавление нового счета, удаление счета, редактирование имени счета. |
| Statistics | График трат и диаграмма категорий трат |
| Operation history | История операций |
| Calendar | Календарь трат |
| Goal | Создание, редактирование и удаление цели накопления средств |

* 1. **Счет**

На данной вкладке страницы отображаются все счета пользователя с указанием их суммы и имени. Счет можно удалить, переименовать или создать новый.

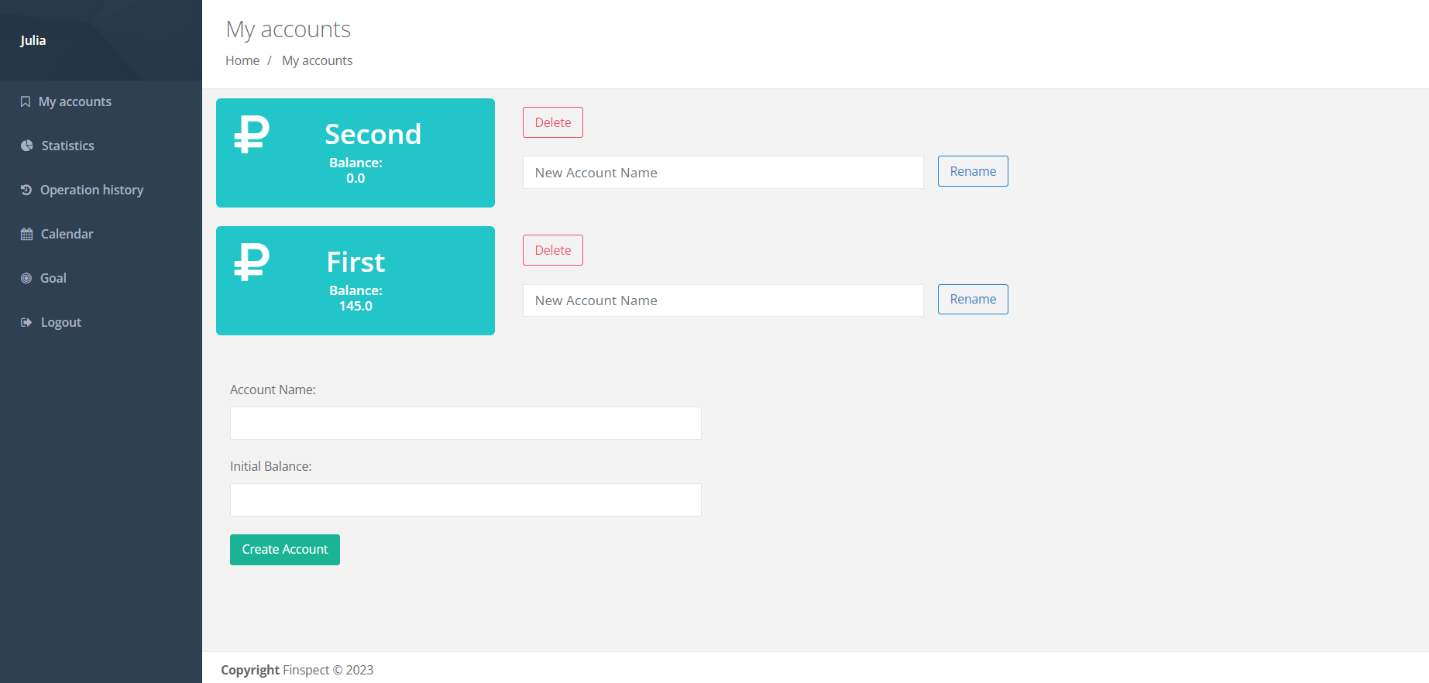
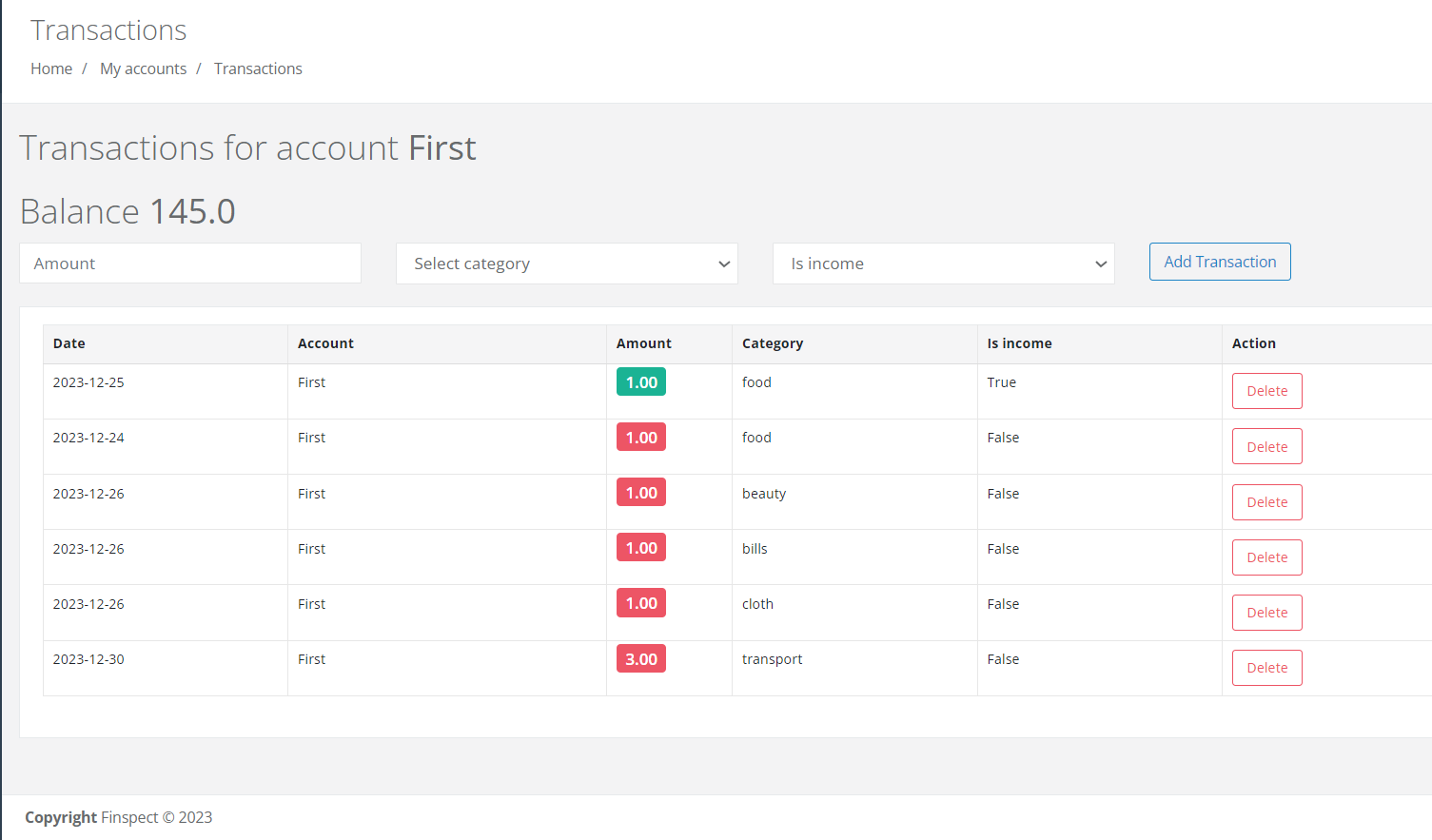
При нажатии на счет пользователь перенаправляется на страницу счета с описанием всех транзакций по этому счету. У каждой транзакции есть сумма, дата, имя счета, категория и доход/расход. Для упрощения анализа таблицы расходы выделены красным цветом, доходы – зеленым.

Рисунок 6. Вкладка счетов пользователя

Рисунок 7. Транзакции счета

* 1. **Статистика**

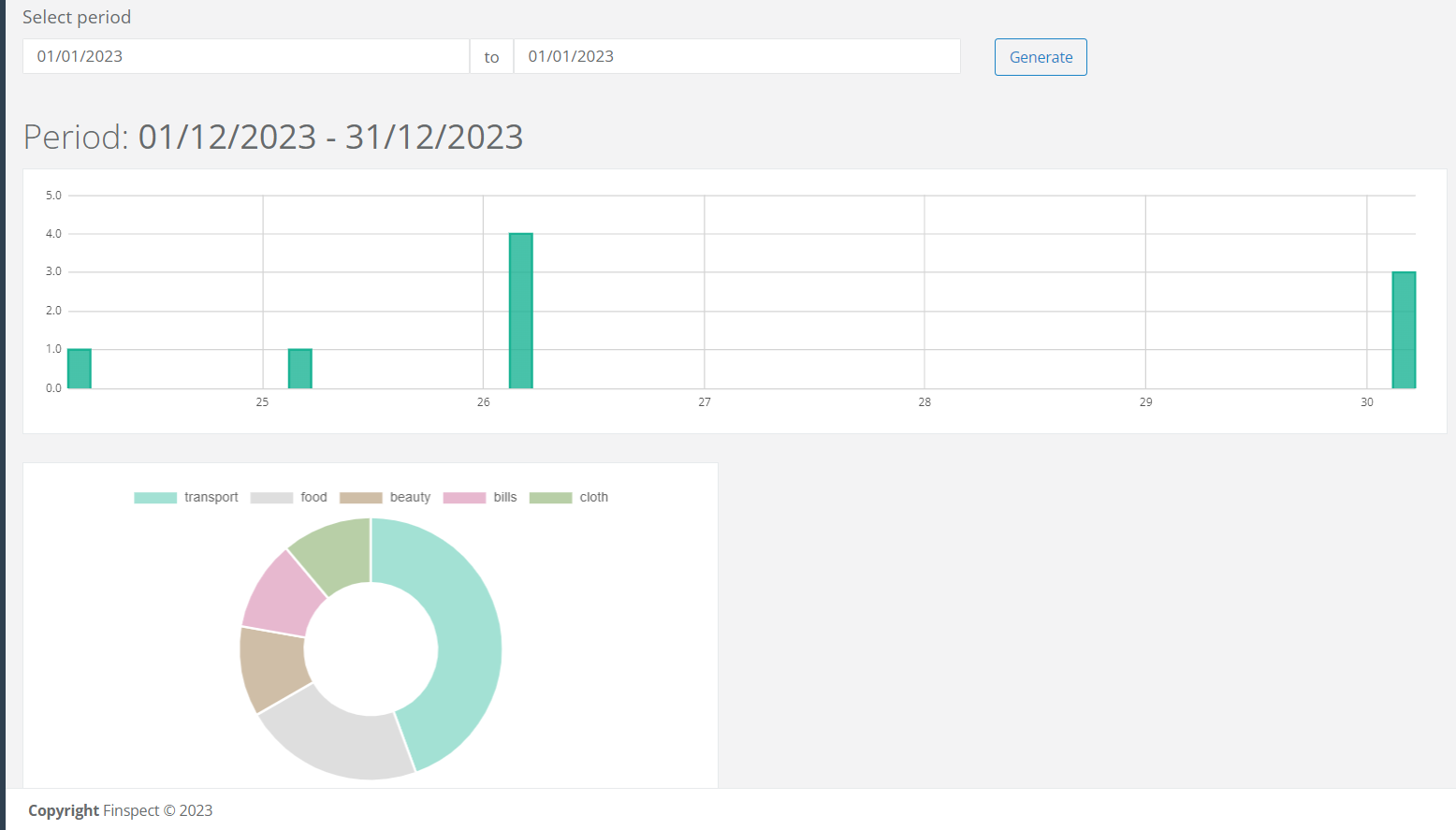
В этой вкладке отображается столбчатая диаграмма расходов и круговая диаграмма категоризации расходов. При загрузке страницы пользователь видит данные за последний месяц среди всех счетов.

Рисунок 8. Вкладка статистики

Функционал страницы предусматривает выбор периода, по которому будут создаваться графики. При неверно введенных данных пользователь перенаправляется на начальную страницу вкладки (отображение графиков за период последнего месяца).

* 1. **История операций**

Здесь происходит отображение истории операций среди всех счетов пользователя. По умолчанию страница отображает 25 последних транзакций, но можно настроить необходимый период с помощью селекторов в верхней части страницы.

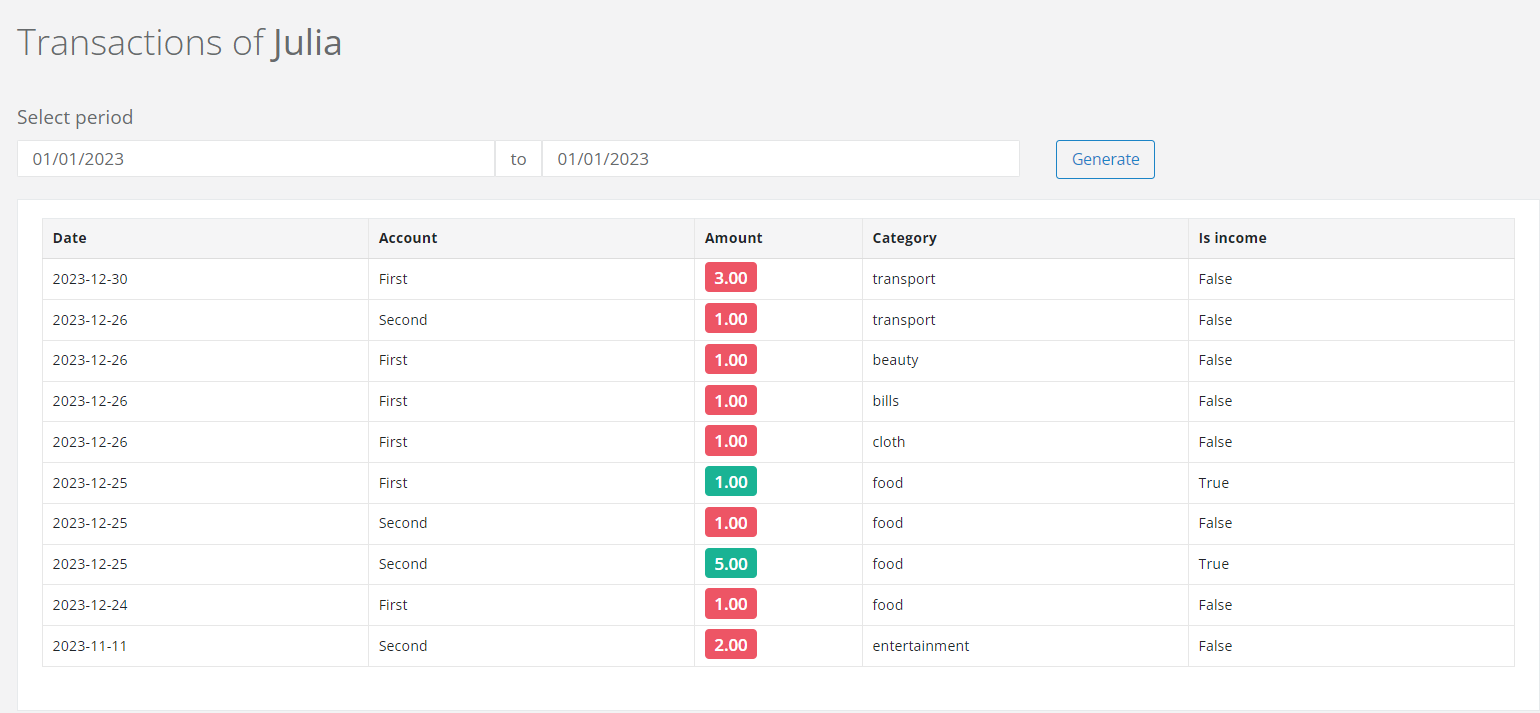


Рисунок 9. История операций пользователя

* 1. **Календарь**

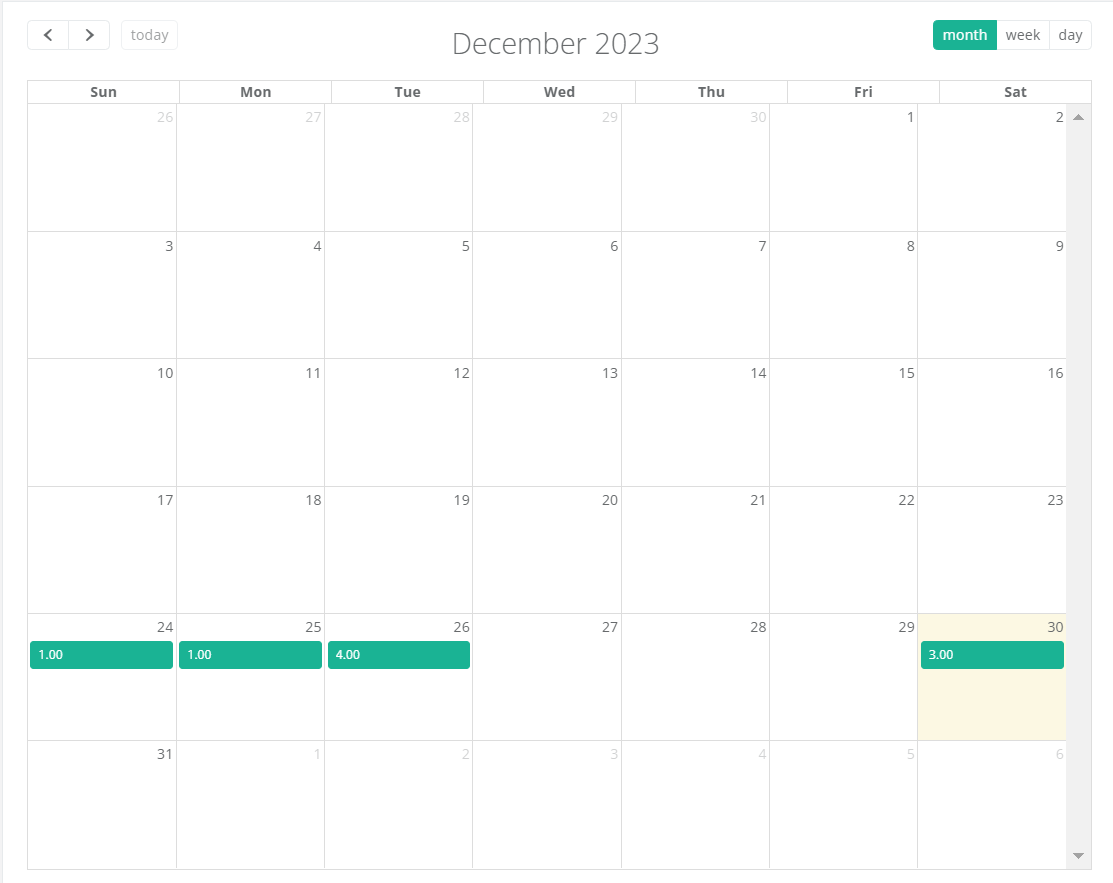
Отображение календаря с указанием суммы траты в ячейке дня.

Рисунок 10. Календарь расходов пользователя

* 1. **Копилка**

Данная вкладка предусматривает функционал “копилки”. Пользователь должен указать необходимую накопленную сумму, дату окончания и свою зарплату. При поступлении зарплаты на счет пользователя некоторый % от суммы зарплаты будет перенаправляться со счета в копилку. Если дата окончания пройдена, но необходимая сумма накопления еще не достигнута копилка продолжает действовать, пока пользователь сам ее не удалит.

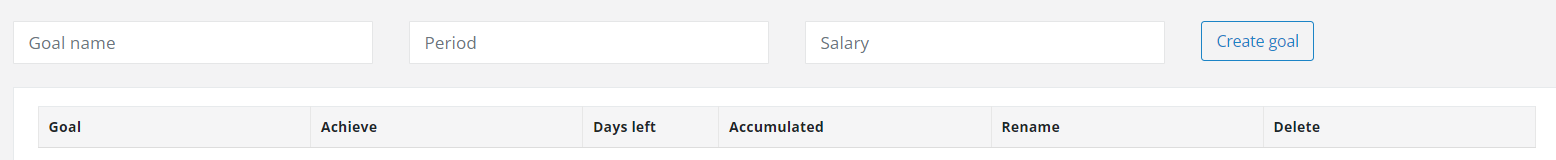


Рисунок 11. Визуализация таблицы на вкладке “Goal”

# **Заключение**

В соответствии с техническим заданием было разработано веб-приложение для контроля финансов пользователя.

В приложении представлены такие функции как:

* регистрация и вход в аккаунт пользователя;
* создание и просмотр счетов пользователя, переименование счета, удаление счета;
* создание транзакций счета, удаление транзакций;
* просмотр истории операций по всем счетам и по каждому счету;
* отображение статистики трат пользователя в виде столбчатой диаграммы по периоду;
* отображение круговой диаграммы категоризации расходов по указанному периоду;
* календарь расходов;
* создание цели накопления.

При разработке использовались следующие технологии: Python, Flask, Jinja2, HTML, CSS, Javascript, PostgreSQL. Приложение основано на клиент-серверной архитектуре. Была спроектирована база данных, которая позволяет обращаться с информацией, которая нужна для функционирования приложения.

Возможное дальнейшее развитие сервиса связано с улучшением интерфейса и добавления новых функций, направленных на оптимизацию процесса контроля финансов.

# **Список используемых источников**

1. Документация Flask. [Электронный ресурс] URL: https://flask.palletsprojects.com/en/latest/ (дата обращения 15.12.2023)
2. The PostgreSQL Global Development Group, PostgreSQL [Электронный ресурс]. 1996-2022 URL: https://www.postgresql.org/ (дата обращения: 17.12.2023)
3. Templating With Jinja2 in Flask [Электронный ресурс] URL: https://www.geeksforgeeks.org/templating-with-jinja2-in-flask/ (дата обращения: 17.12.2023)
4. Документация по Python 3.12.1. [Электронный ресурс] URL: https://docs.python.org/3/index.html (дата обращения 17.12.2023)
5. The Python SQL Toolkit and Object Relational Mapper [Электронный ресурс] URL: https://www.sqlalchemy.org/ (дата обращения 17.12.2023)
6. Bcrypt [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Bcrypt (дата обращения 17.12.2023)
7. bcrypt 4.1.2 [Электронный ресурс] URL: https://pypi.org/project/bcrypt/ (дата обращения 17.12.2023)
8. INSPINIA theme [Электронный ресурс] URL: https://chuibility.github.io/inspinia/ (дата обращения 17.12.2023)

# **Приложение А «Листинги основных контроллеров»**

**А.1 «Контроллеры»**

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

def register():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if len(password) <= 5:

flash('Password should be longer than 5 characters.', 'danger')

return redirect(url\_for('register'))

if not (any(c.isalpha() for c in password) and any(c.isdigit() for c in password)):

flash("Password should contain both letters and numbers.", 'danger')

return redirect(url\_for('register'))

if User.query.filter\_by(username=username).first():

flash("Username is already taken. Please choose a different one.", 'danger')

return redirect(url\_for('register'))

else:

hashed\_password = bcrypt.generate\_password\_hash(password).decode('utf-8')

user = User(username=username, password=hashed\_password)

db.session.add(user)

db.session.commit()

return redirect(url\_for('login'))

return render\_template('register.html')

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = User.query.filter\_by(username=username).first()

if user and bcrypt.check\_password\_hash(user.password, password):

session['user\_id'] = user.id

return redirect(url\_for('dashboard'))

else:

flash('Login failed. Check your username and password.', 'danger')

return redirect(url\_for('login'))

return render\_template('login.html')

@app.route('/logout')

def logout():

return redirect(url\_for('login'))

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

def accounts():

user\_id = session.get('user\_id')

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

name = user.username

form = AccountForm()

if form.validate\_on\_submit():

new\_account = Account(

user\_id=user\_id,

name=form.account\_name.data,

balance=form.initial\_balance.data

)

db.session.add(new\_account)

db.session.commit()

return redirect(url\_for('accounts', user\_id=user.id))

if 'delete\_account' in request.form:

account\_id\_to\_delete = request.form['delete\_account']

FinancialTransaction.query.filter\_by(account\_id=account\_id\_to\_delete).delete()

acc = Account.query.get\_or\_404(account\_id\_to\_delete)

db.session.delete(acc)

db.session.commit()

return redirect(url\_for('accounts', user\_id=user.id))

if 'rename\_account' in request.form:

account\_id\_to\_rename = request.form['rename\_account']

new\_account\_name = request.form['new\_account\_name']

account = Account.query.get\_or\_404(account\_id\_to\_rename)

account.name = new\_account\_name

db.session.commit()

return redirect(url\_for('accounts', user\_id=user.id))

user\_accounts = Account.query.filter\_by(user\_id=user\_id).all()

return render\_template('accounts.html', user\_id=user.id, name=name, form=form, user\_accounts=user\_accounts)

@app.route('/calend')

def calend():

user\_id = session.get('user\_id')

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

name = user.username

start\_date = datetime(2023, 1, 1).date()

end\_date = datetime.now().date()

events\_data = []

current\_date = start\_date

while current\_date <= end\_date:

daily\_expenses = db.session.query(db.func.sum(FinancialTransaction.amount)).\

filter(FinancialTransaction.user\_id == user\_id, FinancialTransaction.is\_income == False, FinancialTransaction.transaction\_date == current\_date).scalar()

if daily\_expenses:

events\_data.append({

'title': str(daily\_expenses),

'start': current\_date.strftime('%Y-%m-%d')

})

current\_date += timedelta(days=1)

return render\_template('calend.html', user=user, name=name, events\_data=events\_data)

@app.route('/goal')

def goal():

user\_id = session.get('user\_id')

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

name = user.username

goal = request.form.get('goal\_name')

start\_date = datetime.now().date()

end\_date = request.form.get('period')

salary = request.form.get('salary')

return render\_template('goal.html', user=user, name=name)

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

def history():

user\_id = session.get('user\_id')

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

account = Account.query.filter\_by(user\_id=user\_id).all()

categories = Category.query.all()

#last = request.args.get('show\_last')

if request.method == 'POST':

start\_date\_str = request.form.get('start')

end\_date\_str = request.form.get('end')

try:

start\_date = datetime.strptime(start\_date\_str, '%d/%m/%Y').date()

end\_date = datetime.strptime(end\_date\_str, '%d/%m/%Y').date()

if end\_date >= start\_date:

transactions = FinancialTransaction.query.filter(

FinancialTransaction.user\_id == user\_id,

FinancialTransaction.transaction\_date.between(start\_date, end\_date)

).order\_by(FinancialTransaction.transaction\_date.desc()).all()

else:

return redirect(url\_for('history'))

except ValueError:

return redirect(url\_for('history'))

else:

transactions = FinancialTransaction.query.filter\_by(user\_id=user\_id).order\_by(FinancialTransaction.transaction\_date.desc()).limit(25).all()

return render\_template('history.html', user=user, account=account, categories=categories, transactions=transactions)

#return render\_template('history.html', user=user, account=account, categories=categories, transactions=transactions)

transactions = FinancialTransaction.query.filter\_by(user\_id=user\_id).order\_by(

FinancialTransaction.transaction\_date.desc()).limit(25).all()

return render\_template('history.html', user=user, account=account, categories=categories, transactions=transactions)

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

def statistics():

user\_id = session.get('user\_id')

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

name = user.username

today = datetime.now()

start\_date1 = today.replace(day=1)

end\_date1 = today.replace(day=calendar.monthrange(today.year, today.month)[1])

input\_start = datetime.strptime(str(start\_date1), "%Y-%m-%d %H:%M:%S.%f")

input\_end = datetime.strptime(str(end\_date1), "%Y-%m-%d %H:%M:%S.%f")

start\_date11 = input\_start.strftime("%d/%m/%Y")

end\_date11 = input\_end.strftime("%d/%m/%Y")

if request.method == 'POST':

start\_date1 = request.form.get('start1')

end\_date1 = request.form.get('end1')

try:

start\_date11 = datetime.strptime(start\_date1, '%d/%m/%Y').date()

end\_date11 = datetime.strptime(end\_date1, '%d/%m/%Y').date()

start\_date11 = start\_date11.strftime("%d/%m/%Y")

end\_date11 = end\_date11.strftime("%d/%m/%Y")

except ValueError:

return redirect(url\_for('statistics'))

expenses\_by\_day = (

db.session.query(

func.extract('day', FinancialTransaction.transaction\_date).label('day'),

func.extract('month', FinancialTransaction.transaction\_date).label('month'),

func.extract('year', FinancialTransaction.transaction\_date).label('year'),

func.sum(FinancialTransaction.amount).label('total\_expenses')

)

.filter(

FinancialTransaction.user\_id == user\_id,

FinancialTransaction.transaction\_date.between(start\_date11, end\_date11),

FinancialTransaction.is\_income == False

)

.group\_by('day', 'month', 'year')

.order\_by('year', 'month', 'day')

.all()

)

result = [

[

float(f"{entry[0]:.0f}.{entry[1]:.0f}"),

float(entry[3])

] for entry in expenses\_by\_day

]

expenses\_by\_category = (

db.session.query(

Category.category\_name.label('category\_name'),

func.sum(FinancialTransaction.amount).label('total\_expenses')

)

.join(FinancialTransaction, FinancialTransaction.category == Category.category\_id)

.filter(

FinancialTransaction.user\_id == user\_id,

FinancialTransaction.transaction\_date.between(start\_date11, end\_date11),

FinancialTransaction.is\_income == False

)

.group\_by(Category.category\_name)

.order\_by(func.sum(FinancialTransaction.amount).desc())

.all()

)

category\_names = [entry.category\_name for entry in expenses\_by\_category]

print(category\_names)

total\_expenses = [float(entry.total\_expenses) for entry in expenses\_by\_category]

print(total\_expenses)

return render\_template('statistics.html', user=user, name=name, result=result, category\_names=category\_names, total\_expenses=total\_expenses, start\_date11=start\_date11, end\_date11=end\_date11)

@app.route('/dashboard')

def dashboard():

user\_id = session.get('user\_id')

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

name = user.username

return render\_template('dashboard.html', user=user, name=name)

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

def transactions():

user\_id = session.get('user\_id')

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

account\_id = request.args.get('account\_id')

account = Account.query.get\_or\_404(account\_id)

categories = Category.query.all()

name = None

if request.method == 'POST':

if 'delete\_transaction' in request.form:

transaction\_id\_to\_delete = request.form['delete\_transaction']

transaction\_to\_delete = FinancialTransaction.query.get\_or\_404(transaction\_id\_to\_delete)

if transaction\_to\_delete.is\_income:

if account.balance >= transaction\_to\_delete.amount:

account.balance = Decimal(str(account.balance)) - Decimal(str(transaction\_to\_delete.amount))

db.session.delete(transaction\_to\_delete)

db.session.commit()

else:

return redirect(url\_for('transactions', user\_id=user\_id, account\_id=account\_id))

else:

account.balance = Decimal(str(account.balance)) + Decimal(str(transaction\_to\_delete.amount))

db.session.delete(transaction\_to\_delete)

db.session.commit()

else:

amount = float(request.form['amount'])

category\_id = request.form['category']

is\_income\_str = request.form['is\_income']

is\_income = is\_income\_str.lower() == 'true'

name = (Category.query.filter\_by(category\_id=category\_id).first()).category\_name

if is\_income == False and account.balance < amount:

return redirect(url\_for('transactions', user\_id=user\_id, account\_id=account\_id))

else:

new\_transaction = FinancialTransaction(

amount=amount,

category=category\_id,

transaction\_date=datetime.now().date(),

account\_id=account\_id,

is\_income=is\_income,

user\_id=user\_id

)

if is\_income:

account.balance += amount

else:

account.balance -= amount

db.session.add(new\_transaction)

db.session.commit()

return redirect(url\_for('transactions', user\_id=user\_id, account\_id=account\_id))

transactions = FinancialTransaction.query.filter\_by(user\_id=user\_id, account\_id=account\_id).all()

return render\_template('transactions.html', user=user, account=account, transactions=transactions, categories=categories, name=name)