Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ ПО ЗАЧЕТНОЙ РАБОТЕ

Предмет Языки интернет программирования

Студент группы ИУ6-31Б	23.12.2023	Семенов А.А.
		(И.О. Фамилия)
Преподаватель	23.12.2023	Маняшев Э.Р.
		(И.О. Фамилия)

2023 г.

Проблема

1. В наше время трудно контролировать свои финансы, которые у большинства людей находятся на разных счетах в разных банках.
2. По статистике финансовая подушка безопасности есть у каждого третьего Россиянина, но большинство просто хранит деньги, не преумножая их. Многим людям трудно следить за своими расходами.

Цель

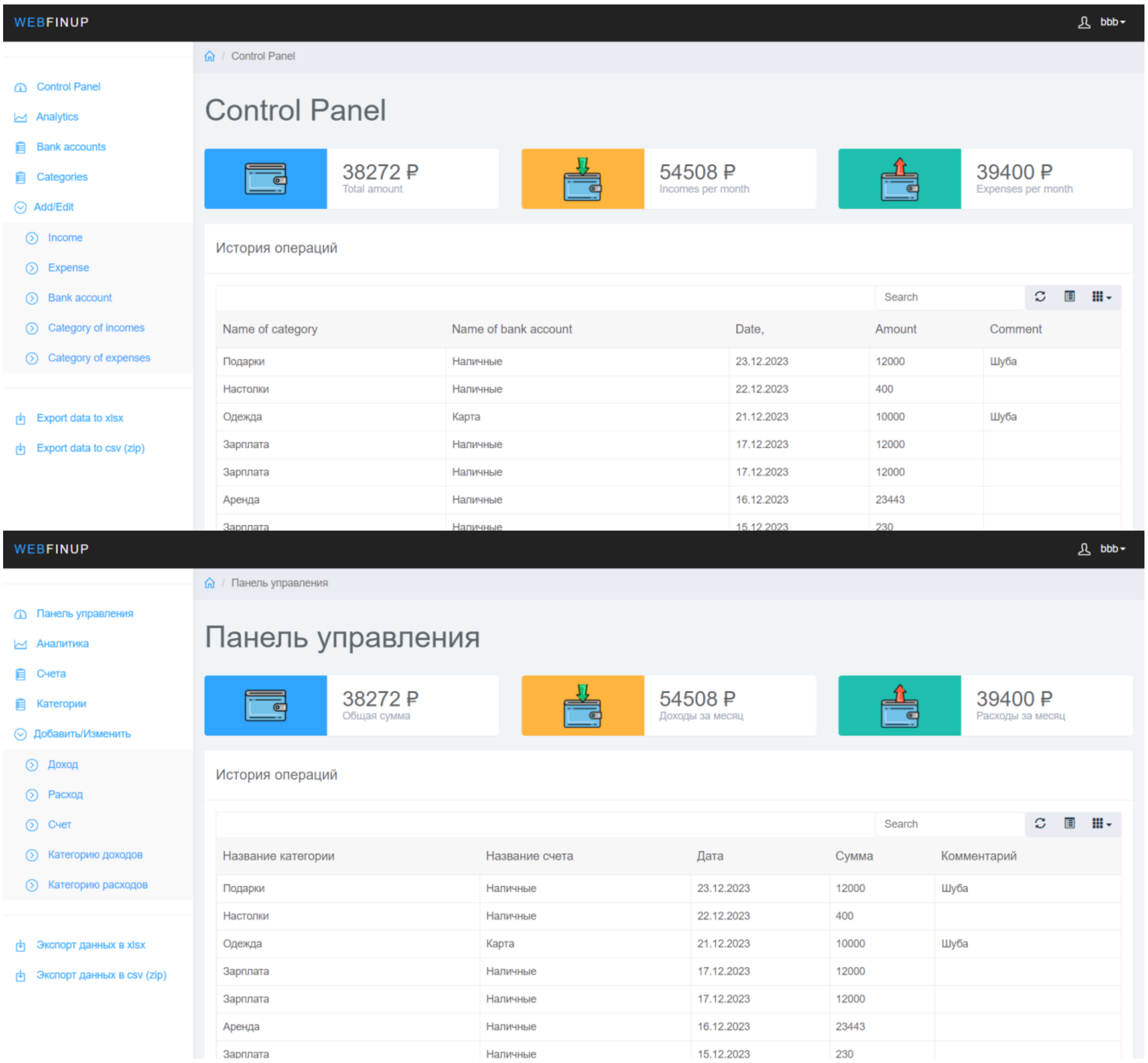
Разработать удобное веб-приложение для контроля финансов.

Задачи (Ход работы)

1. Спроектировать схему базы данных;
2. Разработать интерфейс веб-приложения с использованием шаблонов Bootstrap;
3. Разработать функции для построения графиков на основе переданных данных;
4. Разработать функции для экспорта в xlsx и csv на основе переданных данных;
5. С помощью микрофреймворка Flask привязать html-шаблоны к веб-страницам, наладить взаимодействие с базой данных SQLite, организовать вывод данных на веб-страницы и получение с них других данных от пользователя.

Решение

Код приложения представлен в «Приложении 1», а ниже идут скрины страниц.


WEBFINUP bbb

Control Panel

- Control Panel
- Analytics
- Bank accounts
- Categories
- Add/Edit
  - Income
  - Expense
  - Bank account
  - Category of incomes
  - Category of expenses
- Export data to xlsx
- Export data to csv (zip)

Control Panel

38272 ₽ Total amount | 54508 ₽ Incomes per month | 39400 ₽ Expenses per month

История операций

Search

Name of category	Name of bank account	Date,	Amount	Comment
Подарки	Наличные	23.12.2023	12000	Шуба
Настолки	Наличные	22.12.2023	400	
Одежда	Карта	21.12.2023	10000	Шуба
Зарплата	Наличные	17.12.2023	12000	
Зарплата	Наличные	17.12.2023	12000	
Аренда	Наличные	16.12.2023	23443	
Зарплата	Наличные	15.12.2023	230	

WEBFINUP bbb

Панель управления

- Панель управления
- Аналитика
- Счета
- Категории
- Добавить/Изменить
  - Доход
  - Расход
  - Счет
  - Категорию доходов
  - Категорию расходов
- Экспорт данных в xlsx
- Экспорт данных в csv (zip)

Панель управления

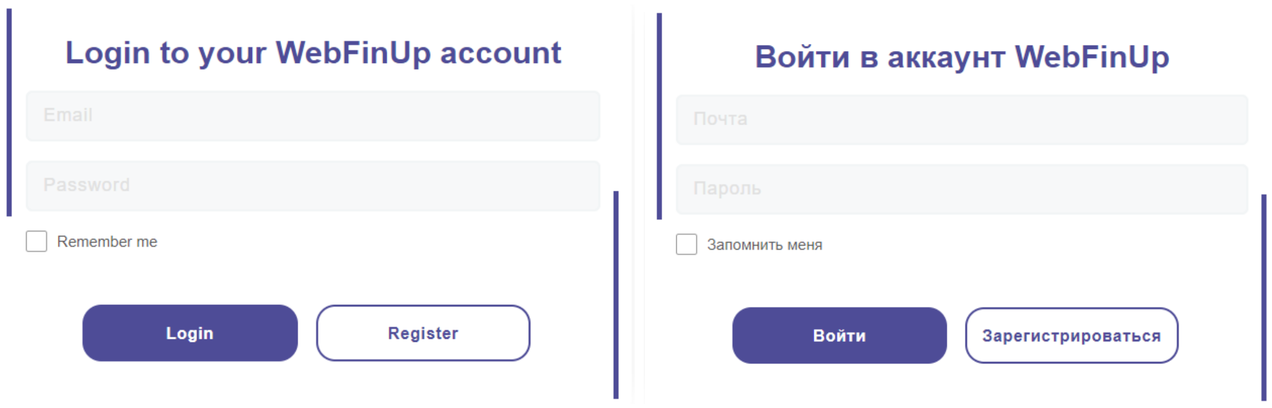
38272 ₽ Общая сумма | 54508 ₽ Доходы за месяц | 39400 ₽ Расходы за месяц

История операций

Search

Название категории	Название счета	Дата	Сумма	Комментарий
Подарки	Наличные	23.12.2023	12000	Шуба
Настолки	Наличные	22.12.2023	400	
Одежда	Карта	21.12.2023	10000	Шуба
Зарплата	Наличные	17.12.2023	12000	
Зарплата	Наличные	17.12.2023	12000	
Аренда	Наличные	16.12.2023	23443	
Зарплата	Наличные	15.12.2023	230	

Рисунок 1 – Главная страница


Login to your WebFinUp account

Email

Password

Remember me

Login | Register

Войти в аккаунт WebFinUp

Почта

Пароль

Запомнить меня

Войти | Зарегистрироваться

Рисунок 2 – Вход

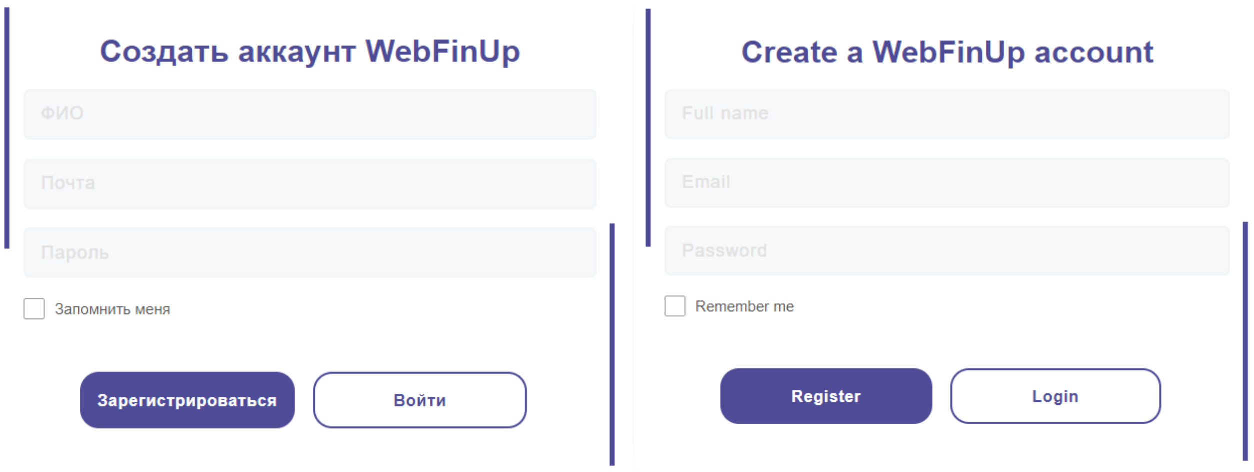
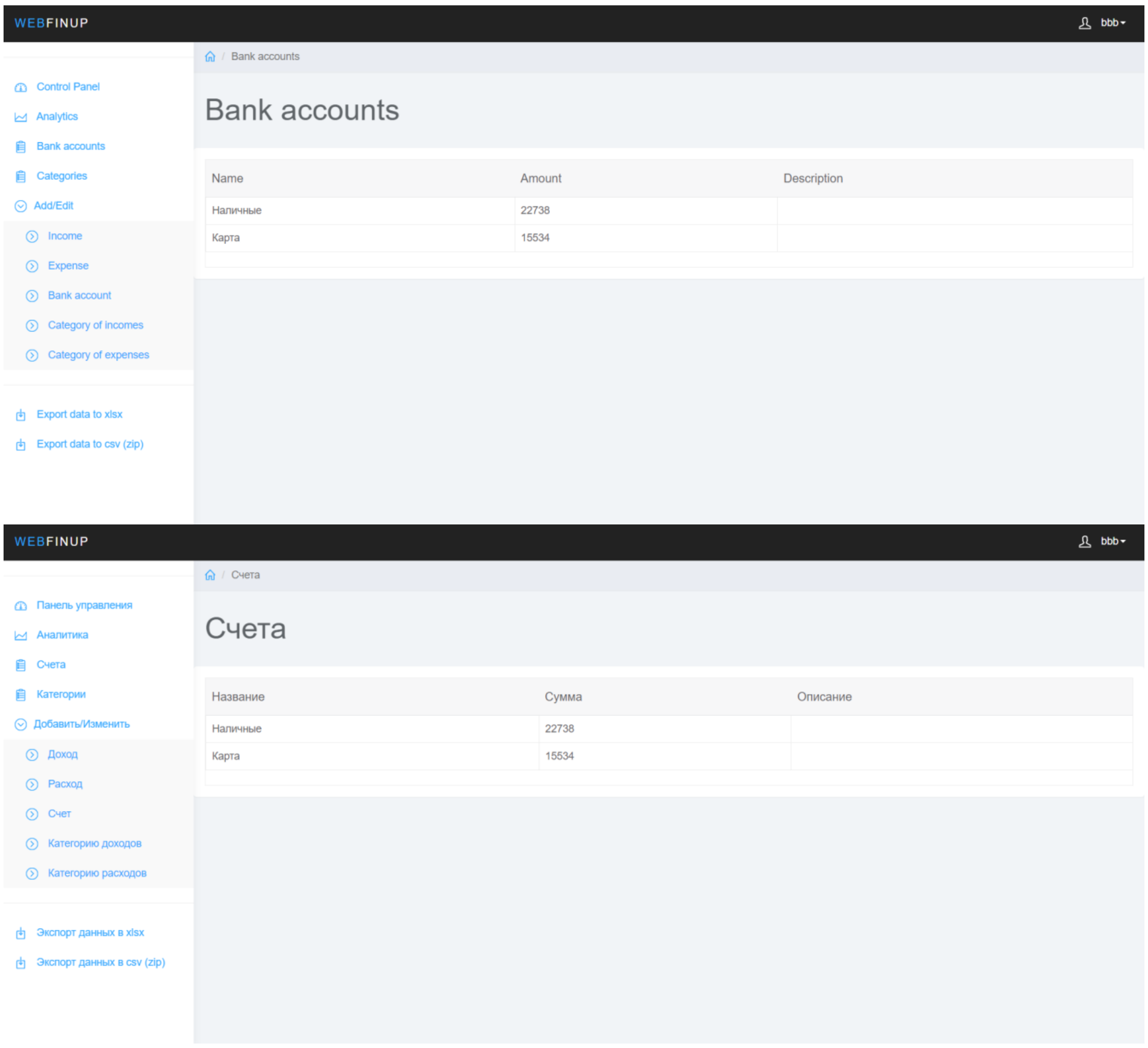


Рисунок 3 – Регистрация



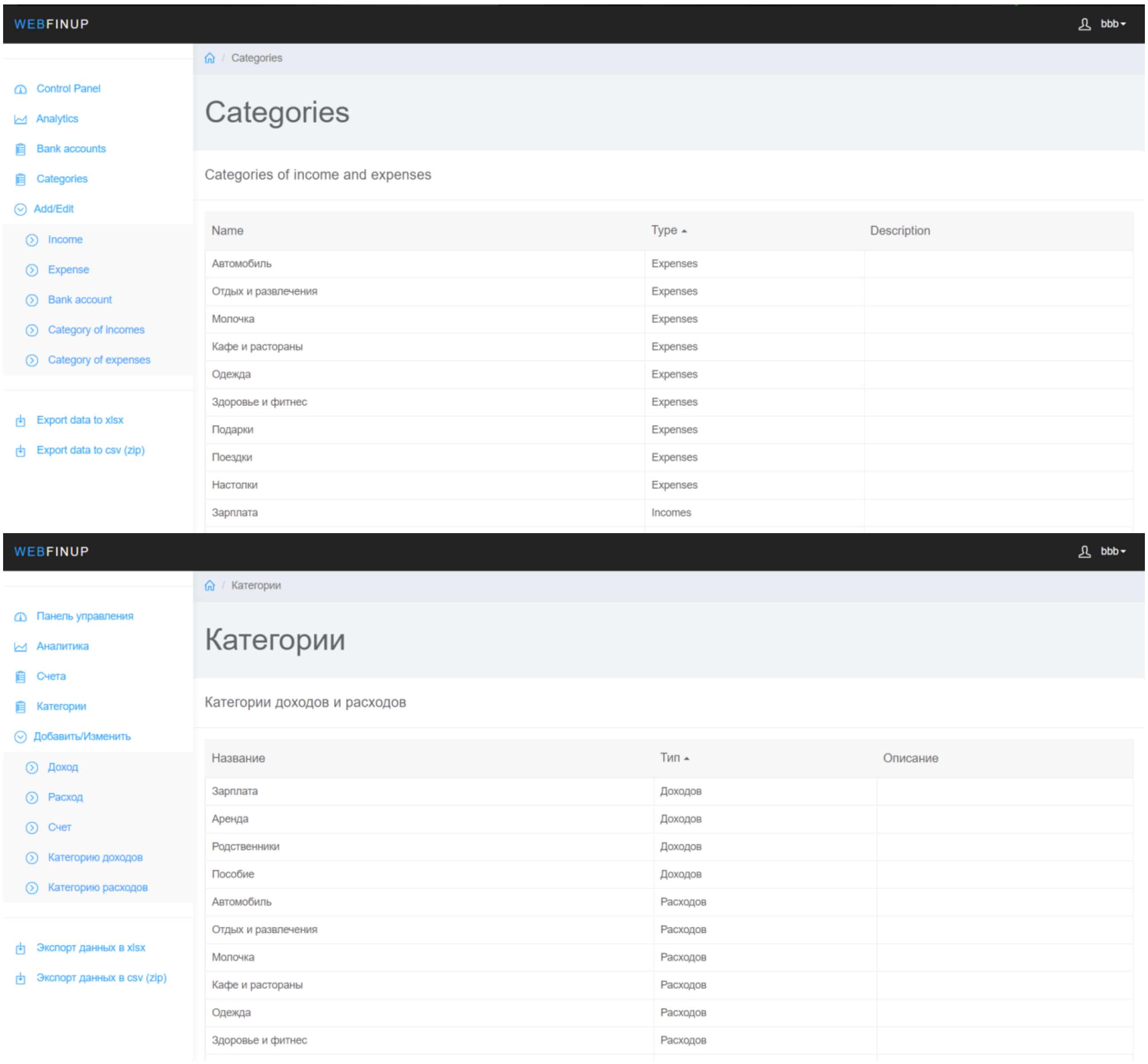
Bank accounts

Name	Amount	Description
Наличные	22738	
Карта	15534	

Счета

Название	Сумма	Описание
Наличные	22738	
Карта	15534	

Рисунок 4 – Просмотр счетов



Categories

Categories of income and expenses

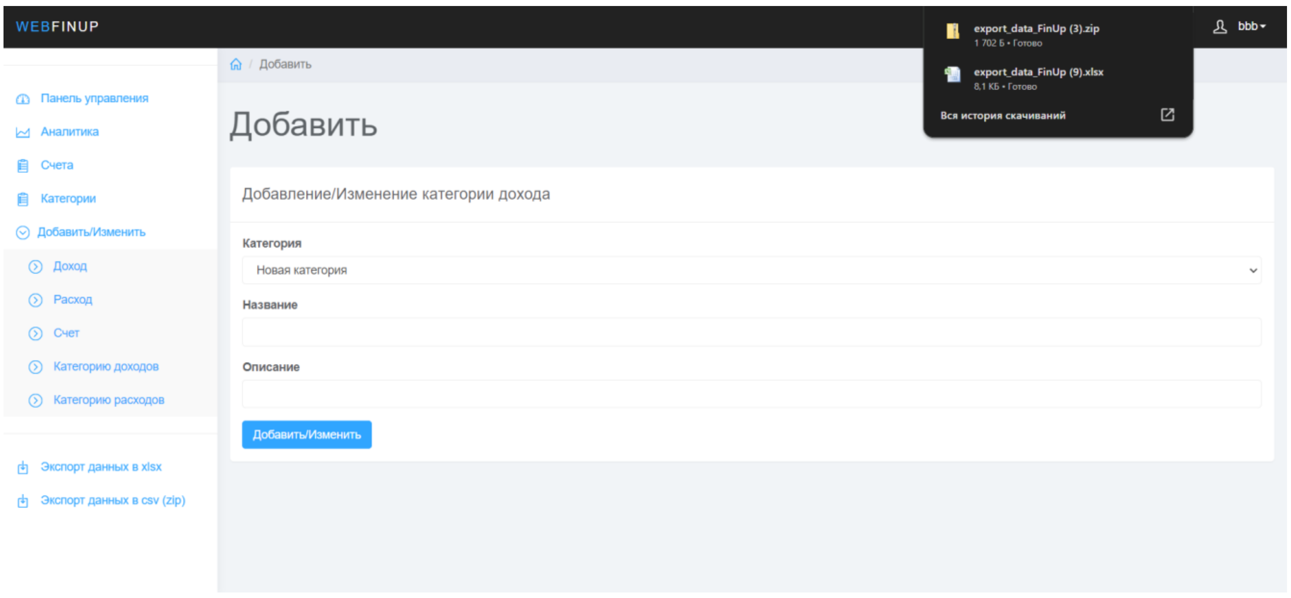
Name	Type	Description
Автомобиль	Expenses	
Отдых и развлечения	Expenses	
Молочка	Expenses	
Кафе и растораны	Expenses	
Одежда	Expenses	
Здоровье и фитнес	Expenses	
Подарки	Expenses	
Поездки	Expenses	
Настолки	Expenses	
Зарплата	Incomes	

Категории

Категории доходов и расходов

Название	Тип	Описание
Зарплата	Доходов	
Аренда	Доходов	
Родственники	Доходов	
Пособие	Доходов	
Автомобиль	Расходов	
Отдых и развлечения	Расходов	
Молочка	Расходов	
Кафе и растораны	Расходов	
Одежда	Расходов	
Здоровье и фитнес	Расходов	

Рисунок 5 – Просмотр категорий доходов и расходов



export_data_FinUp (3).zip
1 702 Б • Готово

export_data_FinUp (9).xlsx
8,1 КБ • Готово

Добавить

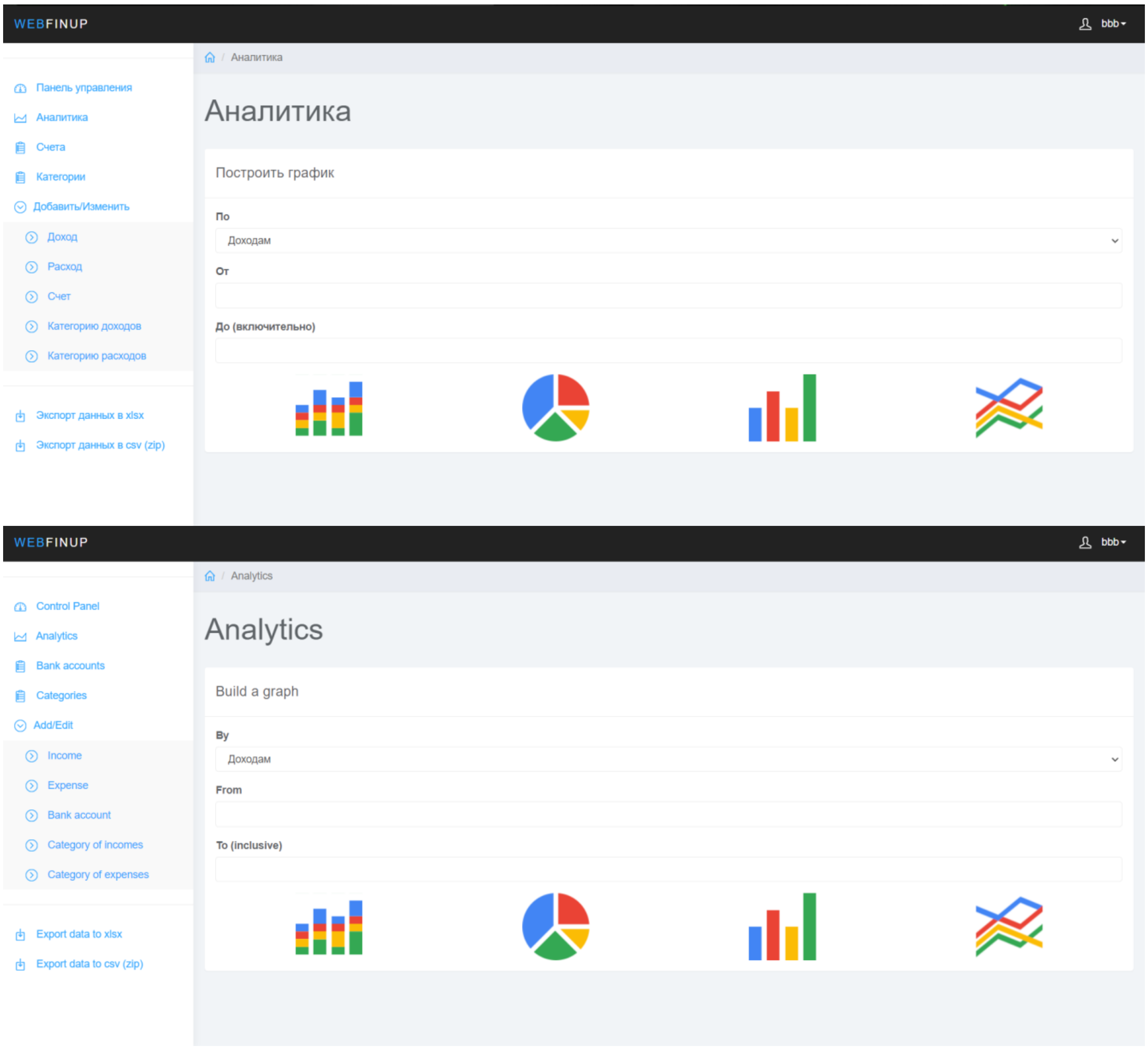
Добавление/Изменение категории дохода

Категория: Новая категория

Название

Описание

Рисунок 6 – Экспорт данных



Аналитика

Построить график

По

Доходам

От

До (включительно)

Analytics

Build a graph

By

Доходам

From

To (inclusive)

Рисунок 7 – Построение графиков

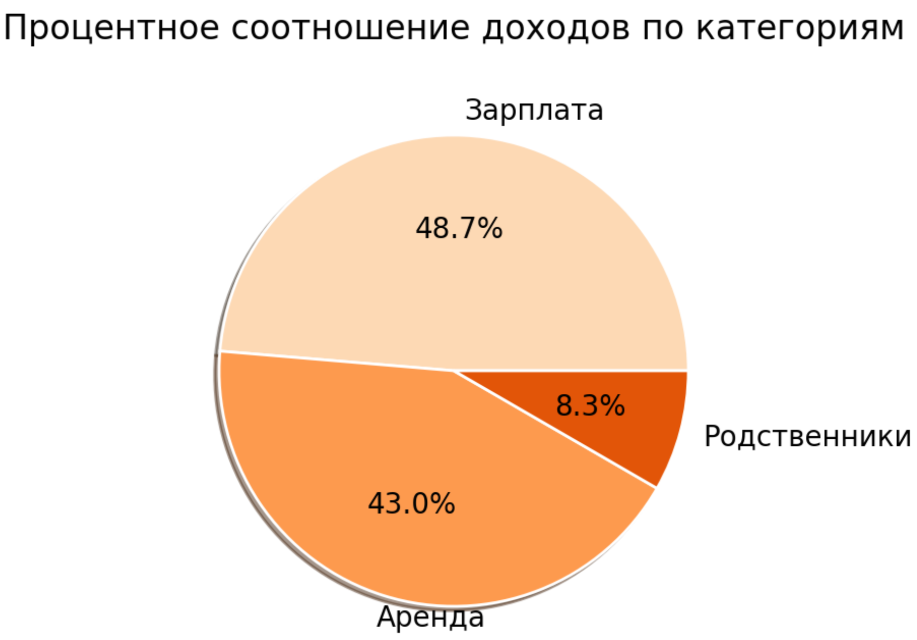
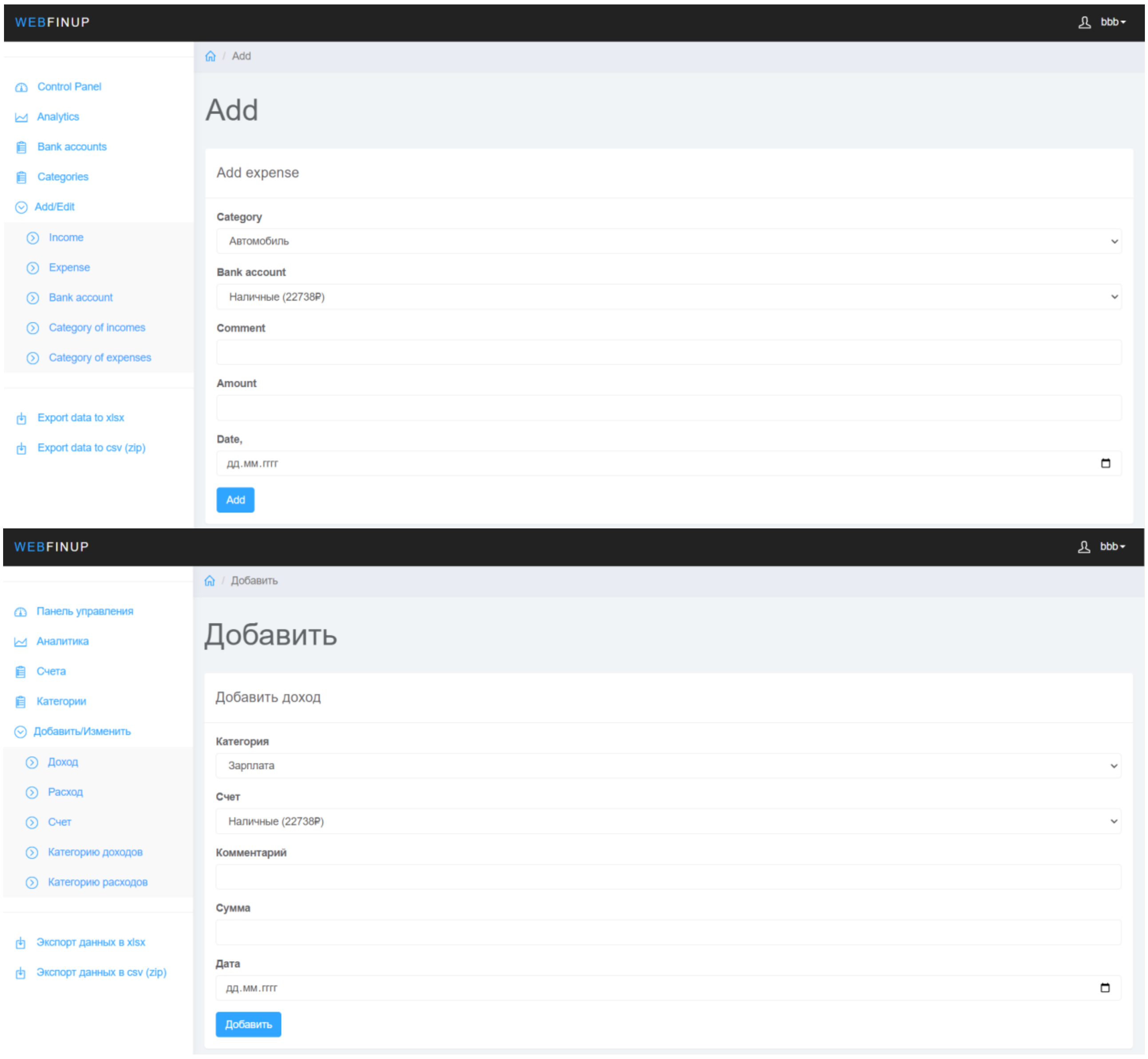


Рисунок 8 – Пример построенного графика


WEBFINUP   bbb▾

Control Panel
Analytics
Bank accounts
Categories
Add/Edit
Income
Expense
Bank account
Category of incomes
Category of expenses
Export data to xlsx
Export data to csv (zip)

Add

Add expense

Category
Автомобиль

Bank account
Наличные (22738₽)

Comment

Amount

Date,
дд.мм.гггг

Add

WEBFINUP   bbb▾

Панель управления
Аналитика
Счета
Категории
Добавить/Изменить
Доход
Расход
Счет
Категорию доходов
Категорию расходов
Экспорт данных в xlsx
Экспорт данных в csv (zip)

Добавить

Добавить доход

Категория
Зарплата

Счет
Наличные (22738₽)

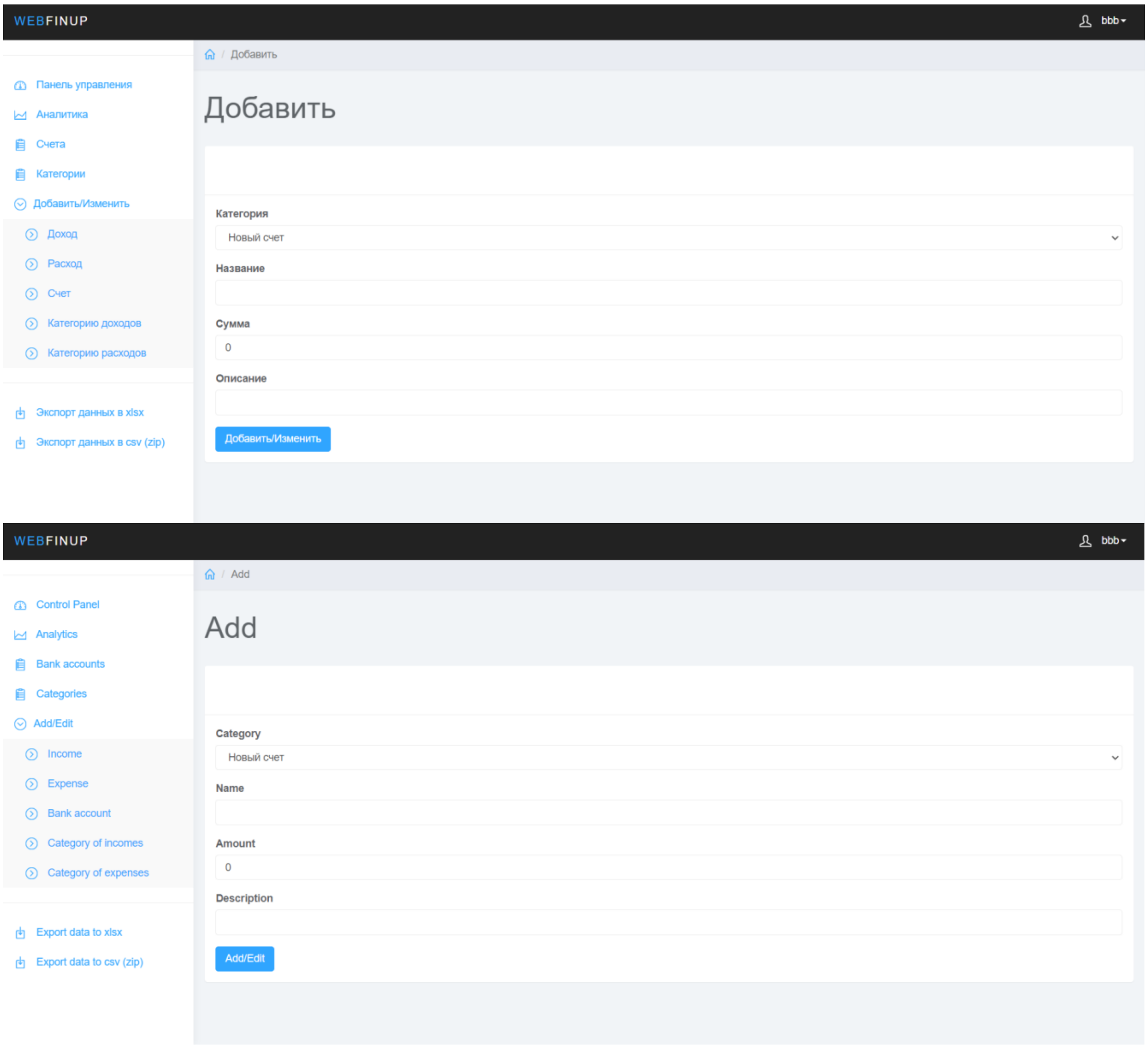
Комментарий

Сумма

Дата
дд.мм.гггг

Добавить

Рисунок 9 – Добавление расхода/дохода



Добавить

Категория: Новый счет

Название:

Сумма: 0

Описание:

Add

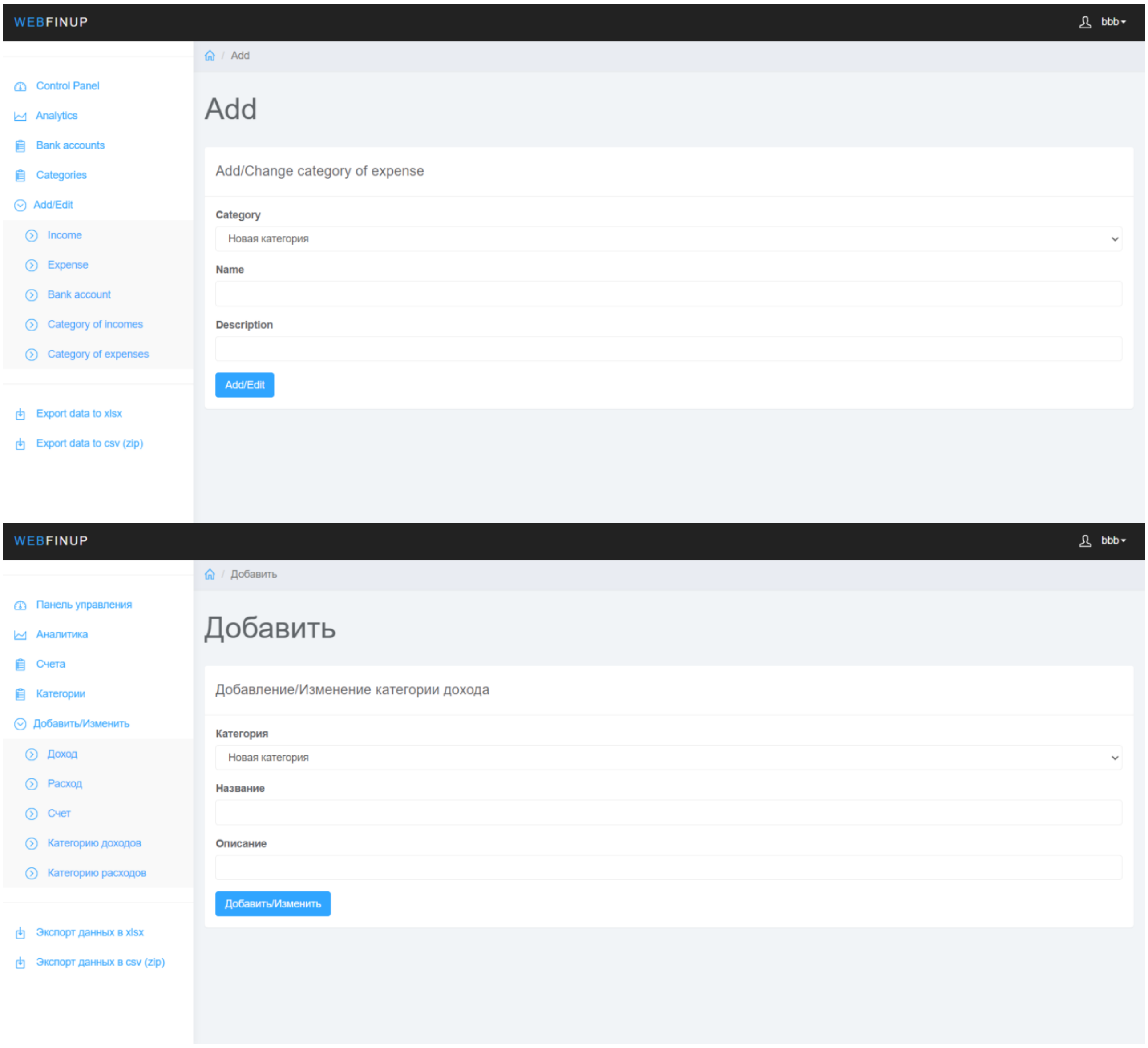
Category: Новый счет

Name:

Amount: 0

Description:

Рисунок 10 – Добавление счёта


WEBFINUP

bbb

Add

Control Panel
Analytics
Bank accounts
Categories
Add/Edit
Income
Expense
Bank account
Category of incomes
Category of expenses
Export data to xlsx
Export data to csv (zip)

Add

Add/Change category of expense

Category: Новая категория

Name:

Description:

Add/Edit

WEBFINUP

bbb

Добавить

Панель управления
Аналитика
Счета
Категории
Добавить/Изменить
Доход
Расход
Счет
Категорию доходов
Категорию расходов
Экспорт данных в xlsx
Экспорт данных в csv (zip)

Добавить

Добавление/Изменение категории дохода

Категория: Новая категория

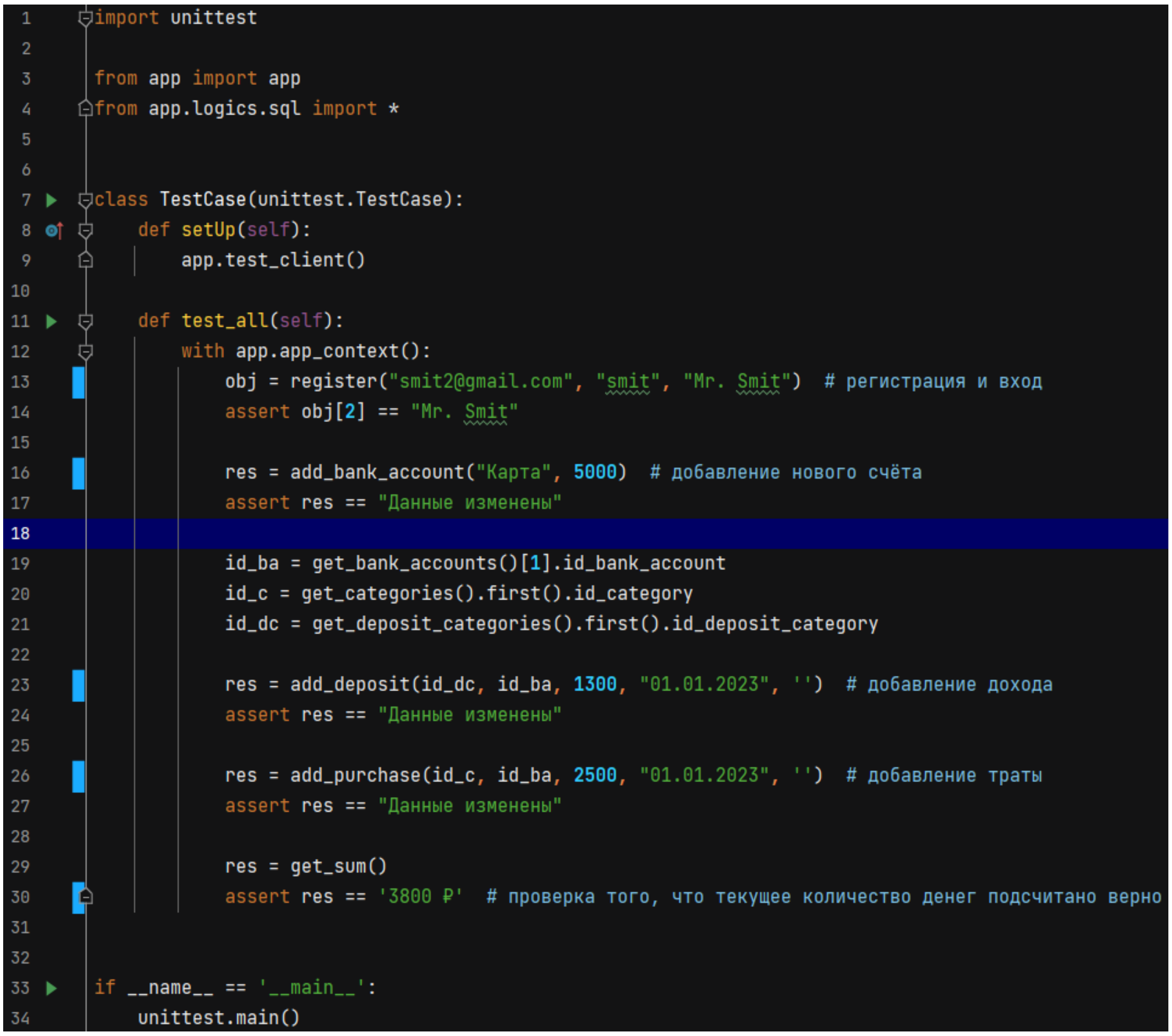
Название:

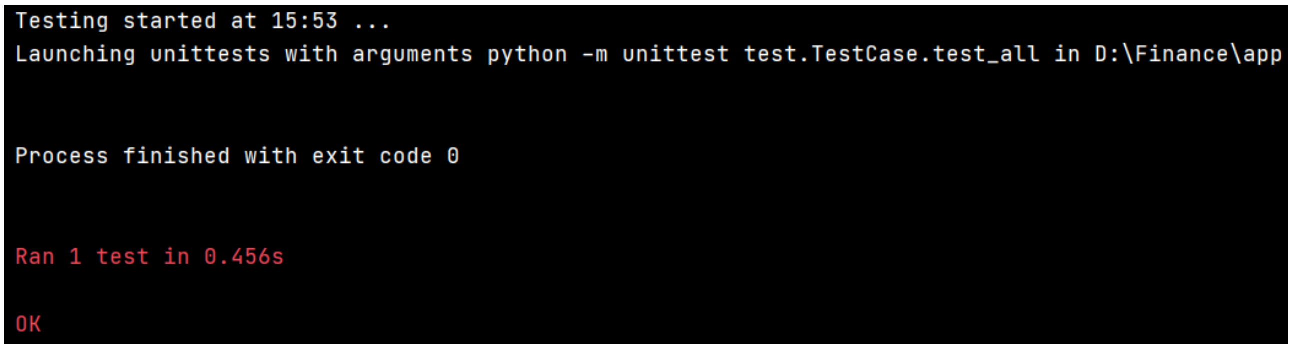
Описание:

Добавить/Изменить

Рисунок 11 – Добавление/изменение категории расходов/доходов

Тестирование


```
import unittest

from app import app
from app.logics.sql import *


class TestCase(unittest.TestCase):
    def setUp(self):
        app.test_client()

    def test_all(self):
        with app.app_context():
            obj = register("smit2@gmail.com", "smit", "Mr. Smit")  # регистрация и вход
            assert obj[2] == "Mr. Smit"

            res = add_bank_account("Карта", 5000)  # добавление нового счёта
            assert res == "Данные изменены"

            id_ba = get_bank_accounts()[1].id_bank_account
            id_c = get_categories().first().id_category
            id_dc = get_deposit_categories().first().id_deposit_category

            res = add_deposit(id_dc, id_ba, 1300, "01.01.2023", '')  # добавление дохода
            assert res == "Данные изменены"

            res = add_purchase(id_c, id_ba, 2500, "01.01.2023", '')  # добавление траты
            assert res == "Данные изменены"

            res = get_sum()
            assert res == '3800 ₽'  # проверка того, что текущее количество денег подсчитано верно


if __name__ == '__main__':
    unittest.main()
```
Рисунок 12 – Код теста


```
Testing started at 15:53 ...
Launching unittests with arguments python -m unittest test.TestCase.test_all in D:\Finance\app


Process finished with exit code 0


Ran 1 test in 0.456s

OK
```
Рисунок 13 – Результат тестирования

Приложение 1

routes.py

```
from flask import render_template, redirect, make_response, request
import json

from app import app

from app.logics.csv_xlsx import *
from app.logics.charts import charts
from app.forms import *
from app.ru_en import ru_en


directory_name = ['app']


class Alert:
    alert = ""

    def get(self):
        s = self.alert
        self.alert = ""
        return s  # вместо s поставить "" для отключения всплывающего окна

    def read(self):
        return self.alert

    def put(self, s):
        self.alert = s


alert = Alert()


def temp_dir():
    lang = request.args.get('lang')
    if lang and lang == 'en':
        return 'en/'
    return ''


def to_lang(word):
    lang = request.args.get('lang')
    if lang and lang == 'en':
        return ru_en[word]
    return word


def am_i_not_login():
    flag = 'id_user' not in request.cookies or request.cookies['id_user'] ==
'null'
    write_request_cookies(request.cookies)
    return flag


def generate_params(title, **kwargs):
    params = {
        "alert": alert.get(),
        "username": get_full_name(),
        "title": title,
        "to_lang": to_lang
    }
    for keys, value in kwargs.items():
        params[keys] = value
    return params


@app.route('/login', methods=['GET', 'POST'])
def login_web():
    form = Login()
    if form.validate_on_submit():
        answer = login(form.email.data, form.password.data)
        if answer == "Неверный логин или пароль":
            alert.put(to_lang(answer))
            return render_template('login.html', form=form, alert=alert.get(),
to_lang=to_lang)
        else:
            alert.put(to_lang("Вы вошли в аккаунт"))
            if form.remember_me.data:
                max_age = 60 * 60 * 24 * 30
            else:
                max_age = None
            resp = make_response(redirect('/' + to_lang('')))
            resp.set_cookie('id_user', str(answer[0]), max_age)
            resp.set_cookie('email', answer[1], max_age)
            resp.set_cookie('full_name', answer[2], max_age)
            return resp
    return render_template('login.html', form=form, alert=alert.get(),
to_lang=to_lang)


@app.route('/register', methods=['GET', 'POST'])
def register_web():
    form = Register()
    if form.validate_on_submit():
        answer = register(form.email.data, form.password.data,
form.username.data)
        if answer == "Аккаунт на эту почту уже зарегистрирован":
            alert.put(to_lang(answer))
            return render_template('register.html', form=form,
alert=alert.get(), to_lang=to_lang)
        else:
            alert.put(to_lang("Поздравляем, Вы создали аккаунт!"))
            if form.remember_me.data:
                max_age = 60 * 60 * 24 * 30
            else:
                max_age = None
            resp = make_response(redirect('/' + to_lang('')))
            resp.set_cookie('id_user', str(answer[0]), max_age)
            resp.set_cookie('email', answer[1], max_age)
            resp.set_cookie('full_name', answer[2], max_age)
            return resp
    return render_template('register.html', form=form, alert=alert.get(),
to_lang=to_lang)


@app.route('/logout')
def logout_web():
    resp = make_response(redirect('/login' + to_lang('')))
    if not (am_i_not_login()):
        resp.set_cookie('id_user', 'null')
        resp.set_cookie('email', 'null')
        resp.set_cookie('full_name', 'null')
        write_request_cookies({'id_user': 'null', 'full_name': 'null', 'email':
```

```
'null'})
        return resp
    return redirect('/login' + to_lang(''))


@app.route('/')
@app.route('/index')
def index():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    data = []
    for i in get_all():
        d = {
            "number": i[5],
            "comment": i[4],
            "name_category": i[0],
            "name_bank_account": i[1],
            "sum": i[2],
            "date": i[3]
        }
        data.append(d)
    with open(directory_name[0] + url_for('static',
filename='tables/data.json'), 'w+') as file:
        json.dump(data, file, sort_keys=True, indent=4)

    summa = get_sum()
    income = get_sum_deposits()
    expense = get_sum_purchases()
    params = generate_params(to_lang("Панель управления"),
                             summa=summa, income=income, expense=expense)
    return render_template('index.html', **params)


@app.route('/income', methods=['GET', 'POST'])
def income():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    form = Income()
    if form.validate_on_submit():
        category = 0
        for i in form.categories_data:
            if i[1] == form.categories.data:
                category = i[0]
        bank_account = 0
        for i in form.bank_accounts_data:
            if i[1] == ' '.join(form.bank_accounts.data.split(' ')[:-1]):
                bank_account = i[0]
        alert.put(
            add_deposit(category, bank_account, form.sum.data,
form.date.data.strftime("%d.%m.%Y"),
                        form.comment.data))
        if alert.read() == "Данные изменены":
            alert.put(to_lang("Данные изменены"))
            return redirect('/' + to_lang(''))
    params = generate_params(to_lang("Добавить"), name=to_lang("Доход"),
form=form)
    return render_template('purchase.html', **params)


@app.route('/expense', methods=['GET', 'POST'])
def expense():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    form = Expense()
    if form.validate_on_submit():
        category = 0
        for i in form.categories_data:
            if i[1] == form.categories.data:
                category = i[0]
        bank_account = 0
        for i in form.bank_accounts_data:
            if i[1] == ' '.join(form.bank_accounts.data.split(' ')[:-1]):
                bank_account = i[0]
        alert.put(add_purchase(category, bank_account, form.sum.data,
                               form.date.data.strftime("%d.%m.%Y"),
form.comment.data))
        if alert.read() == "Данные изменены":
            return redirect('/' + to_lang(''))
    params = generate_params(to_lang("Добавить"), name=to_lang("Расход"),
form=form)
    return render_template('purchase.html', **params)


@app.route('/bank_accounts', methods=['GET', 'POST'])
def bank_accounts():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    data = []
    for i in get_bank_accounts():
        d = {
            "name": i.name,
            "sum": str(i.current_sum),
            "description": i.description
        }
        data.append(d)
    with open(directory_name[0] + url_for('static',
filename='tables/data1.json'), 'w+') as file:
        json.dump(data, file, indent=4)

    params = generate_params(to_lang("Счета"))
    return render_template('bank_accounts.html', **params)


@app.route('/categories', methods=['GET', 'POST'])
def categories():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    data = []
    for i in get_categories():
        d = {
            "name": i.name,
            "of": to_lang("Расходов"),
            "description": i.description
        }
        data.append(d)
    for i in get_deposit_categories():
        d = {
            "name": i.name,
            "of": to_lang("Доходов"),
            "description": i.description
        }
        data.append(d)
    with open(directory_name[0] + url_for('static',
filename='tables/data2.json'), 'w+') as file:
        json.dump(data, file, indent=4)

    params = generate_params(to_lang("Категории"))
    return render_template('categories.html', **params)


@app.route('/income_category', methods=['GET', 'POST'])
def income_category():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    form = IncomeCategory()
    if form.validate_on_submit():
        if form.categories.data == 'Новая категория':
            alert.put(add_deposit_category(form.name.data, form.comment.data))
        else:
            category = 0
            for i in form.categories_data:
                if i[1] == form.categories.data:
                    category = i[0]
            alert.put(edit_deposit_category(category, form.name.data,
form.comment.data))
        if alert.read() == "Данные изменены":
            alert.put(to_lang("Данные изменены"))
            return redirect('/' + to_lang(''))
    params = generate_params(to_lang("Добавить"), name=to_lang("Доход"),
form=form)
    return render_template('category.html', **params)


@app.route('/expenses_category', methods=['GET', 'POST'])
def expenses_category():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    form = ExpensesCategory()
    if form.validate_on_submit():
        if form.categories.data == 'Новая категория':
            alert.put(add_category(form.name.data, form.comment.data))
        else:
            category = 0
            for i in form.categories_data:
                if i[1] == form.categories.data:
                    category = i[0]
            alert.put(edit_category(category, form.name.data,
form.comment.data))
        if alert.read() == "Данные изменены":
            alert.put(to_lang("Данные изменены"))
            return redirect('/' + to_lang(''))
    params = generate_params(to_lang("Добавить"), name=to_lang("Расход"),
form=form)
    return render_template('category.html', **params)


@app.route('/bank_account', methods=['GET', 'POST'])
def bank_account():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    form = BankAccounts()
    if form.validate_on_submit():
        if form.bank_accounts.data == 'Новый счет':
            alert.put(add_bank_account(form.name.data, form.sum.data,
```

```python
form.comment.data))
        else:
            bank_account = 0
            for i in form.bank_accounts_data:
                if i[1] == form.bank_accounts.data:
                    bank_account = i[0]
            alert.put(edit_bank_account(bank_account, form.name.data,
form.sum.data, form.comment.data))
        if alert.read() == "Данные изменены":
            alert.put(to_lang("Данные изменены"))
            return redirect('/' + to_lang(''))
    params = generate_params(to_lang("Добавить"), form=form)
    return render_template('bank_account.html', **params)


@app.route('/analytics', methods=['GET', 'POST'])
def analytics():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))

    form = Analytics()
    if form.validate_on_submit():
        if form.data['submit1']:
            mode = 'bar2'
        elif form.data['submit2']:
            mode = 'pie'
        elif form.data['submit3']:
            mode = 'bar'
        else:
            mode = "plot"
        type = False if form.mode.data == to_lang("Доходам") else True
        charts(type, mode, form.date_start.data, form.date_end.data)
        image = url_for('static', filename='images/saved_figure.png')
        params = generate_params(to_lang("Аналитика"), form=form, image=image)
        return render_template('analytics.html', **params)
    params = generate_params(to_lang("Аналитика"), form=form)
    return render_template('analytics.html', **params)


@app.route('/export_xlsx')
def export_xlsx_web():
    if am_i_not_login():
        return redirect('/login' + to_lang(''))
    export_xlsx()
    return redirect(url_for('static', filename='export/export_data_FinUp.xlsx'))


@app.route('/export_csv')
def export_csv_web():
    if am_i_not_login():
        return redirect('/login + to_lang('')')
    export_csv()
    return redirect(url_for('static', filename='export/export_data_FinUp.zip'))


@app.errorhandler(404)
def web404(error):
    return render_template('404.html', to_lang=to_lang)


@app.errorhandler(500)
def web500(error):
    return render_template('500.html', to_lang=to_lang)
```

models.py

```
from app import db


class Users(db.Model):
    id_user = db.Column(db.Integer, primary_key=True)
    full_name = db.Column(db.String(50))
    username_email = db.Column(db.String(50), unique=True)
    password_hash = db.Column(db.String(150))

    def __init__(self, full_name, username_email, password_hash):
        self.full_name = full_name
        self.username_email = username_email
        self.password_hash = password_hash

    def as_list(self):
        return [getattr(self, c.name) for c in self.__table__.columns]


class DepositCategories(db.Model):
    id_deposit_category = db.Column(db.Integer, primary_key=True)
    id_user = db.Column(db.Integer, db.ForeignKey('users.id_user'))
    name = db.Column(db.String(20))
    description = db.Column(db.String(200))

    def __init__(self, id_user, name, description):
        self.id_user = id_user
        self.name = name
        self.description = description

    def as_list(self):
        return [getattr(self, c.name) for c in self.__table__.columns]


class Categories(db.Model):
    id_category = db.Column(db.Integer, primary_key=True)
    id_user = db.Column(db.Integer, db.ForeignKey('users.id_user'))
    name = db.Column(db.String(20))
    description = db.Column(db.String(200))

    def __init__(self, id_user, name, description):
        self.id_user = id_user
        self.name = name
        self.description = description

    def as_list(self):
        return [getattr(self, c.name) for c in self.__table__.columns]


class BankAccounts(db.Model):
    id_bank_account = db.Column(db.Integer, primary_key=True)
    id_user = db.Column(db.Integer, db.ForeignKey('users.id_user'))
    name = db.Column(db.String(20))
    current_sum = db.Column(db.Integer)
    description = db.Column(db.String(200))

    def __init__(self, id_user, name, current_sum, description):
        self.id_user = id_user
        self.name = name
        self.current_sum = current_sum
        self.description = description
    def as_list(self):
        return [getattr(self, c.name) for c in self.__table__.columns]


class Deposits(db.Model):
    id_deposit = db.Column(db.Integer, primary_key=True)
    id_deposit_category = db.Column(db.Integer,

db.ForeignKey('deposit_categories.id_deposit_category'))
    id_bank_account = db.Column(db.Integer,
db.ForeignKey('bank_accounts.id_bank_account'))
    sum = db.Column(db.Integer)
    date = db.Column(db.String(20))
    comment = db.Column(db.String(200))
    date_time_add = db.Column(db.String(20))

    def __init__(self, id_deposit_category, id_bank_account, sum, date, comment,
date_time_add):
        self.id_deposit_category = id_deposit_category
        self.id_bank_account = id_bank_account
        self.sum = sum
        self.date = date
        self.comment = comment
        self.date_time_add = date_time_add

    def as_list(self):
        return [getattr(self, c.name) for c in self.__table__.columns]


class Purchases(db.Model):
    id_purchase = db.Column(db.Integer, primary_key=True)
    id_category = db.Column(db.Integer, db.ForeignKey('categories.id_category'))
    id_bank_account = db.Column(db.Integer,
db.ForeignKey('bank_accounts.id_bank_account'))
    sum = db.Column(db.Integer)
    date = db.Column(db.String(20))
    comment = db.Column(db.String(200))
    date_time_add = db.Column(db.String(20))

    def __init__(self, id_category, id_bank_account, sum, date, comment,
date_time_add):
        self.id_category = id_category
        self.id_bank_account = id_bank_account
        self.sum = sum
        self.date = date
        self.comment = comment
        self.date_time_add = date_time_add

    def as_list(self):
        return [getattr(self, c.name) for c in self.__table__.columns]
```

forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField,
EmailField, SelectField, \
    DateField
from wtforms.validators import DataRequired

from app.logics.sql import *
```

```
class Login(FlaskForm):
    email = EmailField('Почта', validators=[DataRequired()])
    password = PasswordField('Пароль', validators=[DataRequired()])
    remember_me = BooleanField('Запомнить меня')
    submit = SubmitField('Войти')


class Register(FlaskForm):
    username = StringField('ФИО', validators=[DataRequired()])
    email = EmailField('Почта', validators=[DataRequired()])
    password = PasswordField('Пароль', validators=[DataRequired()])
    remember_me = BooleanField('Запомнить меня')
    submit = SubmitField('Зарегистрироваться')


class Income(FlaskForm):
    categories = SelectField('Категория', validators=[DataRequired()])
    bank_accounts = SelectField('Счет', validators=[DataRequired()])
    comment = StringField('Комментарий')
    sum = StringField('Сумма', validators=[DataRequired()])
    date = DateField('Дата', validators=[DataRequired()])
    submit = SubmitField('Добавить')

    def __init__(self):
        super().__init__()
        self.categories_data = [(str(i.id_deposit_category), i.name) for i in
                                get_deposit_categories()]
        self.categories.choices = [i[1] for i in self.categories_data]
        self.bank_accounts_data = [(str(i.id_bank_account), i.name,
str(i.current_sum))
                                   for i in get_bank_accounts()]
        self.bank_accounts.choices = [i[1] + ' (' + i[2] + '₽)' for i in
self.bank_accounts_data]


class Expense(FlaskForm):
    categories = SelectField('Категория', validators=[DataRequired()])
    bank_accounts = SelectField('Счет', validators=[DataRequired()])
    comment = StringField('Комментарий')
    sum = StringField('Сумма', validators=[DataRequired()])
    date = DateField('Дата', validators=[DataRequired()])
    submit = SubmitField('Добавить')

    def __init__(self):
        super().__init__()
        self.categories_data = [(str(i.id_category), i.name) for i in
get_categories()]
        self.categories.choices = [i[1] for i in self.categories_data]
        self.bank_accounts_data = [(str(i.id_bank_account), i.name,
str(i.current_sum))
                                   for i in get_bank_accounts()]
        self.bank_accounts.choices = [i[1] + ' (' + i[2] + '₽)' for i in
self.bank_accounts_data]


class IncomeCategory(FlaskForm):
    categories = SelectField('Категория', validators=[DataRequired()])
    name = StringField('Название', validators=[DataRequired()])
    comment = StringField('Описание')
    submit = SubmitField('Добавить/Изменить')

    def __init__(self):
        super().__init__()
        self.categories_data = [(str(i.id_deposit_category), i.name)
                                for i in get_deposit_categories()]
        self.categories_data.insert(0, ('-1', 'Новая категория'))
        self.categories.choices = [i[1] for i in self.categories_data]


class ExpensesCategory(FlaskForm):
    categories = SelectField('Категория', validators=[DataRequired()])
    name = StringField('Название', validators=[DataRequired()])
    comment = StringField('Описание')
    submit = SubmitField('Добавить/Изменить')

    def __init__(self):
        super().__init__()
        self.categories_data = [(str(i.id_category), i.name) for i in
get_categories()]
        self.categories_data.insert(0, ('-1', 'Новая категория'))
        self.categories.choices = [i[1] for i in self.categories_data]


class BankAccounts(FlaskForm):
    bank_accounts = SelectField('Счет', validators=[DataRequired()])
    name = StringField('Название', validators=[DataRequired()])
    sum = StringField('Сумма', validators=[DataRequired()], default='0')
    comment = StringField('Описание')
    submit = SubmitField('Добавить/Изменить')

    def __init__(self):
        super().__init__()
        self.bank_accounts_data = [(str(i.id_bank_account), i.name) for i in
get_bank_accounts()]
        self.bank_accounts_data.insert(0, ('-1', 'Новый счет'))
        self.bank_accounts.choices = [i[1] for i in self.bank_accounts_data]


class Analytics(FlaskForm):
    mode = SelectField('Режим', validators=[DataRequired()], choices=['Доходам',
'Расходам'])
    date_start = StringField('От', validators=[DataRequired()])
    date_end = StringField('До', validators=[DataRequired()])
    submit1 = SubmitField('', id='first')
    submit2 = SubmitField('', id='second')
    submit3 = SubmitField('', id='third')
    submit4 = SubmitField('', id='forth')
```

ru_en.py

```
ru_en = {
    "Неверный логин или пароль": "Wrong login or password",
    "Вы вошли в аккаунт": "You are signed in",
    "Аккаунт на эту почту уже зарегистрирован":
        "An account for this email has already been registered",
    "Поздравляем, Вы создали аккаунт!": "Congratulations, you have created an
account!",
    "Добавить": "Add",
    "Доход": "Income",
    "Расход": "Expense",
    "Расходов": "Expenses",
    "Доходов": "Incomes",
    "Доходам": "Incomes",
    "Расходам": "Expenses",
    "Категории": "Categories",
    "Панель управления": "Control Panel",
    "Данные изменены": "Data changed",
    "Счета": "Bank accounts",
    'Новая категория': 'New category',
    'Новый счет': 'New bank account',
    "Аналитика": "Analytics",
    "Средств недостаточно": "There are not enough funds",
    "Счет с таким названием уже существует": "An account with the same name
already exists",
    "Категория с таким названием уже существует": "A category with the same name
already exists",

    "Общая сумма": "Total amount",
    "Доходы за месяц": "Incomes per month",
    "Расходы за месяц": "Expenses per month",
    "Название счета": "Name of bank account",
    "Дата": "Date,",
    'Сумма': 'Amount',
    'Комментарий': 'Comment',
    "Добавить доход": "Add income",
    "Категория": "Category",
    "Счет": "Bank account",
    'Добавление/Изменение категории': 'Add/Change category of',
    "Название категории": "Name of category",
    "Категории доходов и расходов": "Categories of income and expenses",
    'Тип': 'Type',
    'Описание': 'Description',
    'Название': "Name",
    'Добавление/Изменение счета': 'Add/Change bank account',
    'Добавить/Изменить': 'Add/Edit',
    'По': 'By',
    "От": 'From',
    'До (включительно)': 'To (inclusive)',
    "Построить график": "Build a graph",
    "Упс! Вы ввели данные в неверном формате!":
        "Oops! You entered data in the wrong format!",
    "К сожалению мы не можем выполнить запрос из-за некорректных введенных
данных.":
        "Unfortunately, we are unable to complete your request due to incorrect
data entered.",
    "Вернуться назад": "Come back",
    "Ой! Страница не найдена!": "Oh! Page not found!",
    "К сожалению мы не можем найти запрошенную вами страницу. Возможно, вы
неправильно ввели адрес.":
        "Unfortunately, we cannot find the page you requested. You may have
entered the address "
        "incorrectly.",
    'Выход': "Logout",
    "Категорию расходов": "Category of expenses",
    "Категорию доходов": "Category of incomes",
    "Экспорт данных в xlsx": "Export data to xlsx",
    "Экспорт данных в csv (zip)": "Export data to csv (zip)",
    '': '?lang=en',
    "Почта": "Email",
    "Пароль": "Password",
    'Запомнить меня': "Remember me",
    'Войти': 'Login',
    'ФИО': 'Full name',
    'Зарегистрироваться': 'Register',
    'Режим': 'Mode',
    "Создать аккаунт WebFinUp": "Create a WebFinUp account",
    "Войти в аккаунт WebFinUp": "Login to your WebFinUp account"
}
```

charts.py

```python
import matplotlib.pyplot as plt
from datetime import date, timedelta
import numpy as np
from app.logics.sql import *


def charts_by_categories(type: bool, time_mode: int, args):
    if type:
        categories = get_categories()
    else:
        categories = get_deposit_categories()
    labels = []
    sums = []
    for elem in categories:
        cursum = 0
        start = list(args[:time_mode]) + [1] * (3 - time_mode)
        end = list(args[time_mode:]) + [1] * (3 - time_mode)
        date_start = date(*start)
        date_stop = date(*end)
        for elem2 in get_purchase_deposit(type, elem):
            if date_start <= date(*map(int, elem2.date.split('.')[::-1])) <
date_stop:
                cursum += elem2.sum
        labels.append(elem.name)
        sums.append(cursum)
    return labels, sums


def plus_one(time_mode, date_start):
    if time_mode == 1:
        date_start = date(date_start.year + 1, date_start.month, date_start.day)
    elif time_mode == 2:
        if date_start.month == 11:
            date_start = date(date_start.year + 1, 1, date_start.day)
        else:
            date_start = date(date_start.year, date_start.month + 1,
date_start.day)
    else:
        date_start = date_start + timedelta(days=1)
    return date_start


def charts_by_date(type: bool, time_mode: int, args):
    labels = []
    cs = {}
    if type:
        categories = get_categories()
    else:
        categories = get_deposit_categories()
    for elem in categories:
        cs[elem.name] = []

    date_start = date(*(list(args[:len(args) // 2]) + [1] * (3 - time_mode)))
    date_next = plus_one(time_mode, date_start)
    date_stop = date(*(list(args[len(args) // 2:]) + [1] * (3 - time_mode)))
    while date_start < date_stop:
        labels.append('.'.join(date_start.strftime("%d.%m.%Y").split('.')[3 -
time_mode:]))
        arggs = [date_start.year, date_start.month, date_start.day][:time_mode]
+ \
                [date_next.year, date_next.month, date_next.day][:time_mode]
        for label, sum in zip(*charts_by_categories(type, time_mode, arggs)):
            cs[label].append(sum)
        date_start = date_next
        date_next = plus_one(time_mode, date_start)
    if len(labels) <= 6:
        return cs, labels
    new_labels = []
    n = (len(labels) - 1) / 5
    q = 0.0
    for i in range(0, len(labels)):
        if i == round(q):
            new_labels.append(labels[i])
            q += n
        else:
            new_labels.append('')

    return cs, new_labels


def charts(type: bool, mode: str, one: str,
           two: str):  # year_start, month_start, day_start, year_stop,
month_stop, day_stop
    time_mode = one.count(".") + 1
    if time_mode == 1:
        two = str(int(two) + 1)
    else:
        two = str(int(two.split('.')[0]) + 1) + two[two.index('.'):]
    args = [int(i) for i in one.split('.')[::-1] + two.split('.')[::-1]]
    fig, ax = plt.subplots(figsize=(6, 4))
    if mode == "plot":
        cs, labels = charts_by_date(type, time_mode, args)
        for category, sums in cs.items():
            ax.plot(np.arange(len(sums)), sums, label=category)
            plt.xticks(np.arange(len(sums)), labels, rotation='horizontal')

        ax.set_xlabel('Дата')
        ax.set_ylabel('Рубли')
        ax.set_title(f'График {"расходов" if type else "доходов"} за выбранный
период')
        ax.legend()
    elif mode == 'pie':
        labels, sums = charts_by_categories(type, time_mode, args)
        i = 0
        while i < len(labels):
            if sums[i] == 0:
                del sums[i]
                del labels[i]
            else:
                i += 1
        colors = plt.get_cmap('Oranges')(np.linspace(0.2, 0.7, len(labels)))
        ax.pie(sums, colors=colors, radius=3, center=(4, 4), labels=labels,
frame=True,
               wedgeprops={"linewidth": 1, "edgecolor": "white"},
autopct='%1.1f%%', shadow=True)
        ax.set_title(f'Процентное соотношение {"расходов" if type else
"доходов"} по категориям')
        ax.set(xlim=(0, 8), xticks=range(1, 1), ylim=(0, 8), yticks=range(1, 1))
        ax.axis('off')
    elif mode == 'bar':
        labels, sums = charts_by_categories(type, time_mode, args)
        i = 0
        while i < len(labels):
            if sums[i] == 0:
                del sums[i]
                del labels[i]
            else:
                i += 1
        x = np.arange(len(labels))
        width = 0.35
        fig, ax = plt.subplots()
        ax.bar(x, sums, width)
        ax.set_xlabel('Кактегории')
        ax.set_ylabel('Рубли')
        plt.xticks(x, labels, rotation='horizontal')
        ax.set_title(f'Суммы по категориям {"расходов" if type else "доходов"}')
    elif mode == 'bar2':
        cs, labels = charts_by_date(type, time_mode, args)
        ind = np.arange(len(tuple(cs.values())[0]))
        width = 0.35
        last = None
        for category, sums in cs.items():
            ax.bar(ind, sums, width, bottom=last, label=category)
            last = sums

        ax.axhline(0, color='grey', linewidth=0.8)
        ax.set_ylabel('Рубли')
        ax.set_xlabel('Дата')
        ax.set_title(f'{"Расходы" if type else "Доходы"} за выбранный период')
        plt.xticks(ind, labels, rotation='horizontal')
        ax.legend()

    plt.savefig('app/static/images/saved_figure.png', dpi=200)
```

csv_xslx.py

```
from flask import url_for

from app.logics.sql import get_all_data

import xlsxwriter
from zipfile import ZipFile

headers2 = {
    "incomes_categories": ("id_incomes_category", "name", "description"),
    "expenses_categories": ("id_expenses_category", "name", "description"),
    "bank_accounts": ("id_bank_account", "name", "current_sum", "description"),
    "incomes": ("id_income", "id_incomes_categories", "id_bank_account", "sum",
"date", "comment"),
    "expenses": ("id_expense", "id_expenses_category", "id_bank_account", "sum",
"date", "comment")
}


def export_xlsx():
    workbook = xlsxwriter.Workbook('app' +
                                   url_for('static',
filename='export/export_data_FinUp.xlsx'))
    for name, cur_list in zip(sorted(headers2), get_all_data()):
        headers = headers2[name]
        worksheet = workbook.add_worksheet(name=name)
        for column, i in enumerate(headers):
            worksheet.write(0, column, i)
        for row, i in enumerate(cur_list):
            for column, j in enumerate(i.as_list()):
                worksheet.write(row + 1, column, j)
    workbook.close()
    return "Осуществлен экспорт"


def export_csv():
    with ZipFile('app' + url_for('static',
filename='export/export_data_FinUp.zip'), 'w') as myzip:
        for name, cur_list in zip(sorted(headers2), get_all_data()):
            headers = headers2[name]
            string = ';'.join(headers) + '\n'
            for i in cur_list:
                string += ';'.join(map(str, i.as_list())) + '\n'
            myzip.writestr(f'export_data_FinUp_{name}.csv',
string.encode('windows-1251'))
    return "Осуществлен экспорт"
```

sql.py

```
from datetime import datetime as dt
from werkzeug.security import generate_password_hash, check_password_hash
from app.models import *


def format(string, *args):
    for i in args:
        ind1 = string.index('{')
        ind2 = string.index('}')
        string = string[:ind1] + str(i) + string[ind2 + 1:]
    return string


request_cookies = {'id_user': 0, 'full_name': None, 'email': None}


def write_request_cookies(object):
    global request_cookies
    request_cookies = object


def get_id_user():
    return int(request_cookies.get('id_user'))


def get_full_name():
    return request_cookies.get('full_name')


def get_username_email():
    return request_cookies.get('email')


def to_line_list(arr, cut=None):
    ans = []
    for i in arr:
        if cut is not None:
            i = i[:cut]
        ans.extend(map(str, i))
    return ans


def get_all_data():
    return [get_bank_accounts(), get_purchase(), get_categories(),
get_deposits(),
            get_deposit_categories()]


def login(username_email, password):
    ans = Users.query.filter_by(username_email=username_email).first()
    if ans and check_password_hash(ans.password_hash, password):
        id_user = ans.id_user
        full_name = ans.full_name
        return id_user, username_email, full_name
    return "Неверный логин или пароль"


default = ["Автомобиль", "Отдых и развлечения", "Продукты", "Кафе и растораны",
"Одежда",
           "Здоровье и фитнес", "Подарки", "Поездки"]
default1 = ["Зарплата", "Аренда", "Родственники"]


def register(username_email, password, full_name):
    ans = Users.query.filter_by(username_email=username_email).first()
    if ans:
        return "Аккаунт на эту почту уже зарегистрирован"
    new_user = Users(full_name, username_email,
generate_password_hash(password))
    db.session.add(new_user)
    db.session.commit()

    message = login(username_email, password)
    if message != "Неверный логин или пароль":
        write_request_cookies({'id_user': message[0], 'email': message[1],
'full_name': message[2]})
        for i in default:
            add_category(i, "")
        for j in default1:
            add_deposit_category(j, "")
        add_bank_account("Наличные", 0, "")
    return message


def add_category(name, description=""):
    if Categories.query.filter_by(id_user=get_id_user(), name=name).first():
        return "Категория с таким названием уже существует"
    new_category = Categories(get_id_user(), name, description)
    db.session.add(new_category)
    db.session.commit()
    return "Данные изменены"


def get_categories():
    return Categories.query.filter_by(id_user=get_id_user())


def edit_category(id_category, name, description):
    ans = Categories.query.filter_by(id_user=get_id_user(), name=name).first()
    if ans and ans.id_category != id_category:
        return "Категория с таким названием уже существует"
    category = Categories.query.filter_by(id_category=id_category).first()
    category.name = name
    category.description = description
    db.session.commit()
    return "Данные изменены"
```

```python
def add_deposit_category(name, description=""):
    if DepositCategories.query.filter_by(id_user=get_id_user(),
name=name).first():
        return "Категория с таким названием уже существует"
    new_category = DepositCategories(get_id_user(), name, description)
    db.session.add(new_category)
    db.session.commit()
    return "Данные изменены"


def get_deposit_categories():
    return DepositCategories.query.filter_by(id_user=get_id_user())


def edit_deposit_category(id_deposit_category, name, description):
    ans = DepositCategories.query.filter_by(id_user=get_id_user(),
name=name).first()
    if ans and ans != id_deposit_category:
        return "Категория с таким названием уже существует"
    category =
DepositCategories.query.filter_by(id_deposit_category=id_deposit_category).first
()
    category.name = name
    category.description = description
    db.session.commit()
    return "Данные изменены"


def add_bank_account(name, current_sum, description=""):
    if BankAccounts.query.filter_by(id_user=get_id_user(), name=name).first():
        return "Счет с таким названием уже существует"
    new_category = BankAccounts(get_id_user(), name, current_sum, description)
    db.session.add(new_category)
    db.session.commit()
    return "Данные изменены"


def get_bank_accounts():
    return BankAccounts.query.filter_by(id_user=get_id_user())


def edit_bank_account(id_bank_account, name, sum, description=""):
    ans = BankAccounts.query.filter_by(id_user=get_id_user(), name=name).first()
    if ans and ans.id_bank_account != id_bank_account:
        return "Счет с таким названием уже существует"
    bank_account =
BankAccounts.query.filter_by(id_bank_account=id_bank_account).first()
    bank_account.name = name
    bank_account.current_sum = sum
    bank_account.description = description
    db.session.commit()
    return "Данные изменены"


def add_purchase(id_category, id_bank_account, sum, date, comment):
    ans = BankAccounts.query.filter_by(id_bank_account=id_bank_account).first()
    if ans.current_sum < int(sum):
        return "Средств недостаточно"
    ans.current_sum -= int(sum)
    new_purchase = Purchases(id_category, id_bank_account, int(sum), date,
comment,
                             dt.now().strftime("%Y.%m.%d %H:%M:%S"))
    db.session.add(new_purchase)
    db.session.commit()
    return "Данные изменены"


def get_purchase():
    return Purchases.query.join(Categories, Categories.id_category ==
                                Purchases.id_category).filter(Categories.id_user
== get_id_user())


def add_deposit(id_deposit_category, id_bank_account, sum, date, comment):
    ans = BankAccounts.query.filter_by(id_bank_account=id_bank_account).first()
    ans.current_sum += int(sum)
    new_purchase = Deposits(id_deposit_category, id_bank_account, int(sum),
date, comment,
                            dt.now().strftime("%Y.%m.%d %H:%M:%S"))
    db.session.add(new_purchase)
    db.session.commit()
    return "Данные изменены"


def get_deposits():
    return Deposits.query.join(DepositCategories,
DepositCategories.id_deposit_category ==
                               Deposits.id_deposit_category).filter(
        DepositCategories.id_user == get_id_user())


def get_purchase_deposit(type, elem):
    if type:
        return Purchases.query.filter_by(id_category=elem.id_category)
    else:
        return
Deposits.query.filter_by(id_deposit_category=elem.id_deposit_category)


def get_sum():
    sum = 0
    for elem in get_bank_accounts():
        sum += elem.current_sum
    return str(sum) + ' ₽'


def get_sum_deposits():
    sum = 0
    now = [int(i) for i in dt.now().strftime("%m.%Y").split(".")]
    for elem in get_deposits():
        dd = [int(i) for i in elem.date.split(".")][1:]
        if dd == now:
            sum += elem.sum
    return str(sum) + ' ₽'


def get_sum_purchases():
    sum = 0
    now = [int(i) for i in dt.now().strftime("%m.%Y").split(".")]
    for elem in get_purchase():
        dd = [int(i) for i in elem.date.split(".")][1:]
        if dd == now:
            sum += elem.sum
    return str(sum) + ' ₽'


def get_category_name(id_category):
    return Categories.query.filter_by(id_category=id_category).first().name


def get_deposit_category_name(id_deposit_category):
    print(id_deposit_category)
    return
DepositCategories.query.filter_by(id_deposit_category=id_deposit_category).first
().name


def get_bank_acc_name(id_bank_account):
    return
BankAccounts.query.filter_by(id_bank_account=id_bank_account).first().name


def get_all():
    ans = []
    for i in get_deposits():
        ans.append([get_deposit_category_name(i.id_deposit_category),
                    get_bank_acc_name(i.id_bank_account), i.sum, i.date,
i.comment])
    for i in get_purchase():
        ans.append([get_category_name(i.id_category),
get_bank_acc_name(i.id_bank_account),
                    i.sum, i.date, i.comment])
    ans = sorted(ans, key=lambda x: [int(i) for i in x[3].split('.')[::-1]],
reverse=True)
    for i in range(len(ans)):
        ans[i].append(i)
    return ans
```