

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

Кафедра програмних засобів

ЗВІТ

Дисципліна «Розробка прикладних програм»

Робота №3

Тема «Розроблення вебдодатків та реалізація доступу до систем керування
базами даних через програмні інтерфейси»

Виконав варіант 19

Студент КНТ-122

Онищенко О. А.

Прийняли

Викладач

Дейнега Л. Ю.

2024

МЕТА РОБОТИ

Навчитися використовувати програмні інтерфейси для доступу до баз даних. Навчитися розробляти вебдодатки за допомогою фреймворку Django.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Вебдодаток керування фінансами, який забезпечує виконання одноразових та періодичних платежів клієнтами. Початкова кількість коштів на рахунку кожного клієнта та його кредитні ліміти визначаються менеджером, якому доступна вся інформація про клієнтів. Клієнт може переглядати всю інформацію про власний рахунок (разом з історією транзакцій), вносити кошти, сплачувати за послуги, призначати періодичні платежі.

ТЕКСТИ ФАЙЛІВ

Виділено жирним шрифтом місця з найважливішими змінами.

runner/settings.py

```
"""
Django settings for runner project.

Generated by 'django-admin startproject' using Django 5.1.3.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""

from pathlib import Path
```

```

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-
14#l+(td8@3g#=#m4)jw+%@jk!t6@&cjly6vyctboulo=f=^8p^'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'finance',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'runner.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

```

```

WSGI_APPLICATION = 'runner.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'USER': 'root',
        'PASSWORD': '1313',
        'HOST': 'localhost',
        'PORT': '3306',
        'NAME': 'data',
    }
}

# Password validation
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidat
or',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/

```

```

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

```

runner/urls.py

```

"""
URL configuration for runner project.

The `urlpatterns` list routes URLs to views. For more information
please see:
    https://docs.djangoproject.com/en/5.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('finance.urls')),
]

```

finance/template/client.html

```

<a href="/">Home</a>
<h1>{{client.name}}</h1>
<p>Balance: {{client.balance}}</p>
<p>Limit: {{client.credit}}</p>
<a href="{% url 'deposit' id=client.id%}">Deposit</a>
<a href="{% url 'withdraw' id=client.id%}">Withdraw</a>

{%if periodic_payments%}
<h2>Periodic Payments</h2>
<table>
  <tr>
    <td>Amount</td>
    <td>Purpose</td>
    <td>Period</td>

```

```

        <td>Next payment</td>
        <td>Action</td>
    </tr>
    {%for row in periodic_payments%}
    <tr>
        <td>{{row.amount}}</td>
        <td>{{row.purpose}}</td>
        <td>{{row.period}}</td>
        <td>{{row.next_date|date:'d.m.Y'}}</td>
        <td><a href="{%url 'pay_period' payment_id=row.id%}">Pay</a></td>
    </tr>
    {%endfor%}
</table>
{%endif%}
<a href="{%url 'periodic' id=client.id%}">Periodic</a>

```

```

{%if client.manager%}
<h2>Clients Data</h2>
<table>
    <tr>
        <td>ID</td>
        <td>Name</td>
        <td>Balance</td>
        <td>Credit</td>
        <td>Manager?</td>
        <td>Actions</td>
    </tr>
    {%for c in clients%}
    <tr>
        <td>{{c.id}}</td>
        <td>{{c.name}}</td>
        <td>{{c.balance}}</td>
        <td>{{c.credit}}</td>
        <td>{{c.manager}}</td>
        <td>
            <a href="{%url 'edit' id=c.id admin=client.id%}">Edit</a>
        </td>
    </tr>
    {%endfor%}
</table>
{%endif%}

```

```

{%if payments%}
<h2>Payments</h2>
<table>
    <tr>
        <td>Amount</td>
        <td>Purpose</td>
        <td>Operation</td>
        <td>Type</td>
        <td>Time</td>
    </tr>
    {%for row in payments reversed%}
    <tr>
        <td>{{row.amount}}</td>
        <td>{{row.purpose}}</td>
        <td>{{row.operation}}</td>
        <td>{{row.kind}}</td>
    </tr>
    {%endfor%}
</table>

```

```
        <td>{{row.timestamp|date:'d.m.Y H:i:s'}}</td>
    </tr>
    {%endfor%}
</table>
{%endif%}
```

finance/template/deposit.html

```
<a href="{%url 'client' id=client.id%}">Back</a>
<h1>Deposit for {{client.name}}</h1>
<form method="post">
    {%csrf_token%}
    {{form}}
    <button type="submit">Amen</button>
</form>
```

finance/template/edit.html

```
<a href="{%url 'client' id=admin%}">Back</a>
<h1>Edit {{client.name}}</h1>
<form method="post">
    {%csrf_token%}
    {{form}}
    <button type="submit">Amen</button>
</form>
```

finance/template/home.html

```
<h1>Welcome</h1>
<form method="post">
    {%csrf_token%}
    {{form}}
    <button type="submit">Amen</button>
</form>
```

finance/template/periodic.html

```
<a href="{%url 'client' id=client.id%}">Back</a>
<h1>Adding periodic payment for {{client.name}}</h1>
<form method="post">
    {%csrf_token%}
    {{form}}
    <button type="submit">Amen</button>
</form>
```

finance/template/withdraw.html

```
<a href="{%url 'client' id=client.id%}">Back</a>
<h1>Withdraw for {{client.name}}</h1>
<form method="post">
    {%csrf_token%}
    {{form}}
    <button type="submit">Amen</button>
</form>
```

finance/admin.py

```
from django.contrib import admin

from .models import Client, Payment, PeriodicPayment

class ClientAdmin(admin.ModelAdmin):
    list_display=['name', 'balance', 'credit', 'manager']

class PaymentAdmin(admin.ModelAdmin):

    list_display=['client', 'amount', 'purpose', 'operation', 'kind', 'timestamp',]

class PeriodicPaymentAdmin(admin.ModelAdmin):
    list_display=['client', 'amount', 'purpose', 'period', 'next_date',]

admin.site.register(Client, ClientAdmin)
admin.site.register(Payment, PaymentAdmin)
admin.site.register(PeriodicPayment, PeriodicPaymentAdmin)
```

finance/forms.py

```
from django import forms

from .models import PeriodicPayment

class NameForm(forms.Form):
    name = forms.CharField(label='Client name')

class DepositForm(forms.Form):
    amount = forms.IntegerField(label='Amount to deposit')
    purpose = forms.CharField()

class WithdrawForm(forms.Form):
    amount = forms.IntegerField(label='Amount to withdraw')
    purpose = forms.CharField()

class EditForm(forms.Form):
    name=forms.CharField()
```



```

balance=forms.IntegerField()
credit=forms.IntegerField()
manager=forms.BooleanField(required=False)

class PeriodicForm(forms.Form):
    amount = forms.IntegerField(label='Amount to pay')
    period = forms.ChoiceField(choices=PeriodicPayment.PERIODS)
    purpose = forms.CharField()

```

finance/models.py

```

from django.db import models

class Client(models.Model):
    name=models.CharField(max_length=127)
    balance=models.PositiveIntegerField(default=0)
    credit=models.PositiveIntegerField(default=0)
    manager=models.BooleanField(default=False)

    def __str__(self):
        return self.name

class Payment(models.Model):
    client=models.ForeignKey(Client,on_delete=models.CASCADE)
    timestamp=models.DateTimeField()
    amount=models.PositiveIntegerField()
    purpose=models.TextField()
    OPERATIONS=[
        ('Withdrawal','Withdrawal'),
        ('Deposit','Deposit'),
    ]
    operation=models.CharField(max_length=12,choices=OPERATIONS)
    KINDS=[
        ('Single','Single'),
        ('Periodic','Periodic'),
    ]
    kind=models.CharField(max_length=12,choices=KINDS)

    def __str__(self):
        return f'{self.purpose} on {self.timestamp.strftime('%d.%m.%Y at %H:%M:%S')} by {self.client.name} for {self.amount}'

class PeriodicPayment(models.Model):
    client=models.ForeignKey(Client,on_delete=models.CASCADE)
    amount=models.PositiveIntegerField()
    purpose=models.TextField()
    PERIODS=[
        ('Day','Day'),
        ('Month','Month'),
        ('Year','Year'),
    ]
    period=models.CharField(max_length=12,choices=PERIODS)
    next_date=models.DateField()

    def __str__(self):

```

```

        return f'Every {self.period.lower()} by {self.client.name} for
{self.amount} at {self.next_date.strftime('%d.%m.%Y')}'

```

finance/urls.py

```

from django.urls import path

from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('client/<int:id>', views.client, name='client'),
    path('deposit/<int:id>', views.deposit, name='deposit'),
    path('withdraw/<int:id>', views.withdraw, name='withdraw'),
    path('edit/<int:admin>/<int:id>', views.edit, name='edit'),
    path('periodic/<int:id>', views.periodic, name='periodic'),
    path('pay_period/<int:payment_id>', views.pay_period,
name='pay_period'),
]

```

finance/views.py

```

from django.forms.models import model_to_dict
from django.shortcuts import redirect, render
from django.utils.timezone import make_aware
import datetime

from .forms import *
from .models import Client, Payment, PeriodicPayment

def home(request):
    if request.method=='POST':
        form=NameForm(request.POST)
        if not form.is_valid():
            return redirect('home')
        name=form.cleaned_data['name']
        client=Client.objects.filter(name=name).first()
        if not client:
            client=Client(name=name)
            client.save()
        return redirect('client', id=client.id)
    if request.method=='GET':
        form = NameForm()
        return render(request, 'home.html', {'form':form})

def client(request, id):
    client=Client.objects.get(pk=id)
    clients=Client.objects.all().values() if client.manager else None
    payments=Payment.objects.filter(client=client).values()

    periodic_payments=PeriodicPayment.objects.filter(client=client).values(
)

```

```
    return render(request, 'client.html',
{'client':client,'clients':clients,'payments':payments,'periodic_paymen
ts':periodic_payments}))
```

```
def deposit(request, id):
    client=Client.objects.get(pk=id)
    if request.method=='POST':
        form=DepositForm(request.POST)
        if not form.is_valid():
            return redirect('client', id=client.id)
        amount=form.cleaned_data['amount']
        if client.credit < amount:
            return redirect('client', id=client.id)
        client.credit-=amount
        client.balance+=amount
        client.save()
        payment=Payment(
            client=client,
            timestamp=make_aware(datetime.datetime.now()),
            purpose=form.cleaned_data['purpose'],
            amount=amount,
            operation=dict(Payment.OPERATIONS) ['Deposit'],
            kind=dict(Payment.KINDS) ['Single']
        )
        payment.save()
        return redirect('client', id=client.id)
    if request.method=='GET':
        form=DepositForm()
        return render(request, 'deposit.html',
{'form':form,'client':client})
```

```
def withdraw(request, id):
    client=Client.objects.get(pk=id)
    if request.method=='POST':
        form=WithdrawForm(request.POST)
        if not form.is_valid():
            return redirect('client', id=client.id)
        amount=form.cleaned_data['amount']
        if client.balance < amount:
            return redirect('client', id=client.id)
        client.balance-=amount
        client.credit+=amount
        client.save()
        payment=Payment(
            client=client,
            timestamp=make_aware(datetime.datetime.now()),
            purpose=form.cleaned_data['purpose'],
            amount=amount,
            operation=dict(Payment.OPERATIONS) ['Withdrawal'],
            kind=dict(Payment.KINDS) ['Single']
        )
        payment.save()
        return redirect('client', id=client.id)
    if request.method=='GET':
        form=WithdrawForm()
        return render(request, 'withdraw.html',
{'form':form,'client':client})
```

```

def edit(request, id, admin):
    client=Client.objects.get(pk=id)
    if request.method=='POST':
        form=EditForm(request.POST)
        if not form.is_valid():
            return redirect('client', id=client.id)
        data=form.cleaned_data
        edited_client=Client.objects.filter(name=data['name']).first()
        new_name=data['name']
        new_balance=data['balance']
        new_credit=data['credit']
        new_status=data['manager']
        edited_client.name=new_name
        edited_client.balance=new_balance
        edited_client.credit=new_credit
        edited_client.manager=new_status
        edited_client.save()
        return redirect('client', id=admin)
    if request.method=='GET':
        form=EditForm(initial=model_to_dict(client))
        return render(request, 'edit.html',
{'form':form,'client':client,'admin':admin})

def periodic(request, id):
    client=Client.objects.get(pk=id)
    if request.method=='POST':
        form=PeriodicForm(request.POST)
        if not form.is_valid():
            return redirect('client',id=client.id)
        data=form.cleaned_data
        amount=data['amount']
        period=data['period']
        purpose=data['purpose']
        periodic_payment=PeriodicPayment(
            client=client,
            amount=amount,
            purpose=purpose,
            period=period,
            next_date=datetime.date.today()
        )
        periodic_payment.save()
        return redirect('client',id=client.id)
    if request.method=='GET':
        form=PeriodicForm()
        return render(request, 'periodic.html',
{'form':form,'client':client})

def pay_period(request, payment_id):
    payment=PeriodicPayment.objects.get(pk=payment_id)
    client=payment.client
    if client.balance < payment.amount:
        return redirect('client',id=client.id)
    client.balance-=payment.amount
    client.save()
    payment_log=Payment(
        client=client,
        timestamp=make_aware(datetime.datetime.now()),
        purpose=payment.purpose,

```

```
        amount=payment.amount,
        operation=dict(Payment.OPERATIONS) ['Withdrawal'],
        kind=dict(Payment.KINDS) ['Periodic']
    )
    payment_log.save()

next_year,next_month,next_day=payment.next_date.year,payment.next_date.
month,payment.next_date.day
    if payment.period=='Day':
        next_day=next_day+1
        print(next_day)
        if next_day>=28:
            next_day=1
            next_month=next_month+1
            if next_month>12:
                next_month=1
                next_year=next_year+1
    if payment.period=='Month':
        next_month=next_month+1
        if next_month>12:
            next_month=1
            next_year=next_year+1
    if payment.period=='Year': next_year+=1
    payment.next_date=datetime.date(next_year,next_month,next_day)
    print(payment.next_date)
    payment.save()
    return redirect('client',id=client.id)
```

РЕЗУЛЬТАТИ ВИКОНАННЯ

На вході до програми відкривається сторінка входу:

Welcome

Client name:

Amen

Рисунок 1.1 – Сторінка входу

При вході до існуючого акаунту відкривається сторінка з інформацією про користувача, періодичні виплати та історію усіх виплат:

[Home](#)

Oleg

Balance: 0

Limit: 9

[Deposit](#) [Withdraw](#)

Periodic Payments

Amount Purpose Period Next payment Action
3 taxes Day 19.11.2024 [Pay](#)
[Periodic](#)

Payments

Amount	Purpose	Operation	Type	Time
3	AMEN AND ALLELUJAH	Withdrawal	Single	19.11.2024 21:47:50
3	taxes	Withdrawal	Periodic	19.11.2024 21:47:23
3	taxes	Withdrawal	Periodic	19.11.2024 21:47:21
3	taxes	Withdrawal	Periodic	19.11.2024 21:42:17
3	ALLELUJAH PRAISE KING JESUS CHRIST our HOLY LORD GOD MOST HIGH AMEN	Withdrawal	Single	19.11.2024 20:24:39
12	AMEN GREAT JESUS IS my LORD GOD MOST HIGH	Deposit	Single	19.11.2024 20:21:32
12	ALLELUUJAH	Withdrawal	Single	19.11.2024 20:21:22
12	reaping	Withdrawal	Single	19.11.2024 20:11:26
12	ALLELUJAH	Deposit	Single	19.11.2024 20:09:36

Рисунок 1.2 – Сторінка існуючого користувача

Якщо введено ім'я неіснуючого користувача на сторінці входу, створюється і відкривається сторінка нового користувача:

[Home](#)

newuser

Balance: 0

Limit: 0

[Deposit](#) [Withdraw](#) [Periodic](#)

Рисунок 1.3 – Сторінка новоствореного користувача

При натисканні кнопки «Home» відкривається сторінка входу. При натисканні кнопки «Deposit» відкривається сторінка додавання грошей до балансу:

[Back](#)

Deposit for newuser

Amount to deposit:

Purpose:

Рисунок 1.4 – Сторінка додавання грошей

При натисканні кнопки «Back» відкривається сторінка того ж користувача. Гроші мають братися з кредитного ліміту. Для додавання грошей можна або зробити це через акаунт менеджера, або через адмін панель.

Для зміни кредитного ліміту через адмін панель потрібен адмін акаунт у застосунку. Після входу до нього через *localhost/admin* потрібно обрати таблицю для редагування (в цьому випадку Clients):

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	✎ Change
Users	+ Add	✎ Change
FINANCE		
Clients	+ Add	✎ Change
Payments	+ Add	✎ Change
Periodic payments	+ Add	✎ Change

Рисунок 1.5 – Вигляд адмін панелі

Тоді потрібно обрати необхідного користувача для редагування інформації:

Select client to change

ADD CLIENT +

Action: -----



Go

0 of 6 selected

<input type="checkbox"/>	NAME	BALANCE	CREDIT	MANAGER
<input type="checkbox"/>	newuser	0	0	✗
<input type="checkbox"/>	new	0	0	✗
<input type="checkbox"/>	admin	1168	336	✓
<input type="checkbox"/>	Oleg	0	9	✗
<input type="checkbox"/>	seesm	0	0	✗
<input type="checkbox"/>	seesmof	7	5	✗

6 clients

Рисунок 1.6 – Обрання потрібного користувача зі списку

Після зміни значення кредитного ліміту потрібно зберегти зміни через кнопку «SAVE»:

Change client

newuser

HISTORY

Name:

newuser

Balance:

0

Credit:

127

☐ Manager

SAVE

Save and add another

Save and continue editing

Delete

Рисунок 1.7 – Збереження змін до кредитного ліміту нового користувача

Тепер при переході на сторінку користувача на основному сайті можна побачити застосовані зміни:

[Home](#)

newuser

Balance: 0

Limit: 127

[Deposit](#) [Withdraw](#) [Periodic](#)

Рисунок 1.8 – Змінений кредитний ліміт нового користувача

Для зміни кредитного ліміту через панель менеджера потрібно
зайти у акаунт зі статусом менеджера (*client.manager: bool*):

[Home](#)

admin

Balance: 1168

Limit: 336

[Deposit](#) [Withdraw](#)

Periodic Payments

Amount	Purpose	Period	Next payment	Action
1	Monthly taxes JESUS THANK YOU LORD GOD ALMIGHTY ALLELUJAH AMEN	Month	19.01.2026	Pay
1	Daily taxes ALLELUJAH JESUS THANK YOU LORD GOD MOST HIGH ALLELUJAH AMEN	Day	08.01.2025	Pay
1	Yearly tax JESUS THANK YOU LORD GOD ALMIGHTY ALLELUJAH AMEN	Year	19.11.2029	Pay

[Periodic](#)

Clients Data

ID	Name	Balance	Credit	Manager?	Actions
1	seesmof 7	5	False		Edit
2	seesm	0	0	False	Edit
3	Oleg	0	9	False	Edit
4	admin	1168	336	True	Edit
5	new	0	0	False	Edit
6	newuser 0	127	False		Edit

Рисунок 1.9 – Сторінка користувача менеджера

На сторінці менеджера має бути інформація про усіх користувачів з можливістю редагувати дані кожного. При виборі необхідного користувача і натисканні кнопки «Edit» має відкритися сторінка редагування даних:

[Back](#)

Edit newuser

Name:

Balance:

Credit:

Manager: ☐

Рисунок 1.10 – Сторінка редагування даних нового користувача зі зміненим кредитним лімітом

Після редагування даних і натискання кнопки «Амін» має відкритися сторінка менеджера:

Clients Data

ID	Name	Balance	Credit	Manager?	Actions
1	seesmof	7	5	False	Edit
2	seesm	0	0	False	Edit
3	Oleg	0	9	False	Edit
4	admin	1168	336	True	Edit
5	new	0	0	False	Edit
6	newuser	0	1277	False	Edit

Рисунок 1.11 – Таблиця клієнтів на сторінці менеджера зі зміненим кредитним лімітом нового користувача

Після додавання грошей до кредитного ліміту можна додати їх до рахунку. Для цього потрібно знову перейти до сторінки нового користувача і натиснути кнопку «Deposit»:

[Back](#)

Deposit for newuser

Amount to deposit:

Purpose:

Рисунок 1.12 – Створення транзакції поповнення рахунку нового користувача

При створенні нової транзакції поповнення або зняття грошей завжди потрібно вказувати мету (purpose) транзакції. Після натискання кнопки «Амін» має відкритися сторінка нового користувача з оновленими даними рахунку та здійсненою транзакцією у історії:

[Home](#)

newuser

Balance: 12

Limit: 1265

[Deposit](#) [Withdraw](#) [Periodic](#)

Payments

Amount	Purpose	Operation Type	Time
12	Groceries shopping	Deposit	Single 19.11.2024 23:16:38

Рисунок 1.13 – Сторінка користувача з проведеною транзакцією поповнення рахунку на 12

При натисканні на кнопку «Withdraw» має відкритися сторінка зняття грошей з рахунку:

[Back](#)

Withdraw for newuser

Amount to withdraw:

Purpose:

Рисунок 1.14 – Сторінка зняття коштів зі вказаною причиною

Після вказання суми та причини операції користувач має знову повернутися на сторінку профіля з оновленою інформацією:

[Home](#)

newuser

Balance: 5

Limit: 1272

[Deposit](#) [Withdraw](#) [Periodic](#)

Payments

Amount	Purpose	Operation	Type	Time
7	Taxi	Withdrawal	Single	19.11.2024 23:18:23
12	Groceries shopping	Deposit	Single	19.11.2024 23:16:38

Рисунок 1.15 – Сторінка користувача з оновленими даними після зняття коштів з рахунку

При натисненні на кнопку «Periodic» має відкритися сторінка створення періодичної транзакції:

[Back](#)

Adding periodic payment for newuser

Amount to pay:

Period:

Purpose:

Рисунок 1.16 – Сторінка створення періодичної транзакції

Тут можна вказати суму транзакції, періодичність та мету. Після створення нової періодичної транзакції користувача має перенести до сторінки профілю з доданою періодичною транзакцією:

[Home](#)

newuser

Balance: 5

Limit: 1272

[Deposit](#) [Withdraw](#)

Periodic Payments

Amount	Purpose	Period	Next payment	Action
--------	---------	--------	--------------	--------

3	Taxes	Month	19.11.2024	Pay
---	-------	-------	------------	---------------------

[Periodic](#)

Payments

Amount	Purpose	Operation	Type	Time
7	Taxi	Withdrawal	Single	19.11.2024 23:18:23
12	Groceries shopping	Deposit	Single	19.11.2024 23:16:38

Рисунок 1.17 – Сторінка профілю нового користувача з оновленими даними

При натисканні кнопки «Pay» біля необхідної періодичної транзакції має здійснитися зняття коштів та додавання нової транзакції до історії:

[Home](#)

newuser

Balance: 2

Limit: 1272

[Deposit](#) [Withdraw](#)

Periodic Payments

Amount Purpose Period Next payment Action

3 Taxes Month 19.12.2024 [Pay](#)

[Periodic](#)

Payments

Amount	Purpose	Operation	Type	Time
3	Taxes	Withdrawal	Periodic	19.11.2024 23:22:03
7	Taxi	Withdrawal	Single	19.11.2024 23:18:23
12	Groceries shopping	Deposit	Single	19.11.2024 23:16:38

Рисунок 1.18 – Сторінка профіля після здійснення періодичної транзакції

КОНТРОЛЬНІ ПИТАННЯ

Які існують способи збереження даних в програмах написаних мовою Python?

Кілька способів:

- Як локальні змінні: тоді вони зберігатимуться лише на час роботи програми;
- Як файли: тоді їх потрібно вручну читати та записувати при змінах.

Поширені формати: JSON, CSV, TXT;

- У базі даних: для цього зазвичай використовують Систему керування базою даних. Поширені СКБД: MySQL, PostgreSQL, MongoDB.

Для чого призначена бібліотека MySQLdb?

Для підключення та взаємодії з СКБД MySQL.

Які існують високорівневі функції бібліотеки MySQLdb?

Найпоширеніші:

- `connect`: створює підключення з базою даних. Приймає як аргументи назву хоста, ім'я користувача, пароль, опціонально базу даних з якою працювати;
- `query`: формує та виконує запит до бази даних. Текст запиту пишеться мовою SQL для MySQL;
- `store_result`: завантажує результати запиту та зберігає їх локально в повному обсязі;
- `use_result`: завантажує результати запиту та зберігає їх на сервері, подає рядок за рядком;
- `fetch_row`: показує рядок результатів запиту.

[Джерело](#)

Яким чином виконати запит до бази даних та яким чином переглянути результати?

Через підключення до бази даних викликом методу `connect`, формування запиту викликом методу `query` з текстом запиту мовою SQL,

збереження результатів віддалено викликом методу `use_result`, та виведення результатів рядок за рядком викликом методу `fetch_row`.

АЛИЛУЯ

Що таке шоблон проєктування MVC?

Поділяє архітектуру програми на три рівні:

1. *model* (модель): містить всі інформаційні частини додатку;
2. *view* (представлення): описує логіку інтерфейсу та показує дані користувачеві;
3. *controller* (контроллер): інтерфейс між моделлю та представленням. Обробляє запити, модифікує дані, надсилає їх представленню.

[Джерело](#)

За допомогою яких команд виконується розроблення додатків у бібліотеці Django?

Важливі команди:

1. Додати теку де проєкт буде жити
`mkdir server`
2. Ініціалізувати сам проєкт
`django-admin startproject runner server`
3. Перейти у теку з проєктом
`cd server`
4. Зробити новий застосунок де будуть всі файли основні
`py manage.py startapp app_name`
5. Коли зробили зміни до моделі реєструємо міграцію
`py manage.py makemigrations app_name`
6. Тоді здійснюємо міграцію
`py manage.py migrate`
7. Коли треба до адмін панелі зайти, створити користувача
`py manage.py createsuperuser`

Вказати всі необхідні дані:

- імя користувача
- електронна пошта
- пароль

8. Запустити сервер аби побачити всі зміни на сайті

```
py manage.py runserver
```

З яких файлів складається проєкт Django?

При створенні містить таку структуру файлів:

```
project/  
  default_app/  
    __init__.py  
    asgi.py  
    settings.py  
    urls.py  
    wsgi.py  
  new_app/  
    migrations/  
      __init__.py  
      0001_initial.py  
      __init__.py  
    admin.py  
    apps.py  
    models.py  
    tests.py  
    urls.py  
    views.py  
  manage.py
```

Яку структуру мають додатки Django?

Типова структура:

```
назва_проєкту/  
  назва_додатку/  
  назва_додатку/  
  manage.py
```

Для чого необхідні та яким чином реалізуються моделі Django?

Модель це таблиця даних для бази даних. Реалізується створенням класів моделей у файлі `models.py` потрібного додатку:

```
from django.db import models
class BibleBook(models.Model):
    canonical_order_number=models.PositiveIntegerField()
    name=models.CharField(max_length=77)
    number_of_chapters=models.PositiveIntegerField()
```

Яким чином визначаються представлення Django?

У теці додатку має міститися файл `views.py`. До нього можна додати функцію, яка прийматиме аргумент *request* і повертатиме відповідь типу *HttpResponse*. Приклад такої функції нижче:

```
from django.shortcuts import render
from django.http import HttpResponse
def get_response(request):
    return HttpResponse("ІСУС ХРИСТОС – ГОСПОДЬ")
```

[Джерело](#)

Для чого необхідні та яким чином визначаються і підключаються шаблони?

Для створення шаблону потрібно створити нову теку у теці додатку під назвою *templates*. До неї можна вносити файли шаблонів типу HTML. Сам файл шаблону може містити HTML код, а також спеціальні вирази для відображення динамічного вмісту.

Для підключення шаблону до представлення необхідно модифікувати функцію у файлі `views.py` таким чином:

```
from django.shortcuts import render
from django.template import loader
def get_response(request):
    template=loader.get_template("template_name.html")
    return HttpResponse(template.render())
```

[Джерело](#)

Яким чином можна використати статичні ресурси в Django?

Для додавання статичних файлів до проєкту Django потрібно створити теку static у теці додатку. До теки static можна додавати статичні файли, як от зображення чи таблиці стилів.

Після додавання необхідних статичних файлів для їх відображення у шаблоні можна модифікувати його так:

```
{% load static %}
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="{% static 'file_name.css' %}">
<body>
ІСУС ХРИСТОС – ГОСПОДЬ
</body>
</html>
```

[Джерело](#)