МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления



**ОТЧЁТ**

**по лабораторной работе №12**

**на тему:**

**Авторизация и регистрация**

**по дисциплине:** Разработка программных приложений и WEB-программирование

Выполнили работу:

Студенты гр. АП-226, АВТФ

Ищенко С. Г.

«24» мая 2025 г.

Проверил работу:

Эстрайх И. В.

«\_\_\_» \_\_\_\_\_\_ 20\_\_г

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Цель**

Изучить технологию авторизации и регистрации пользователей.

**Задание**

1. Создать приложение users для регистрации и авторизации

пользователей.

2. При попытке доступа к определенной странице неавторизованного

пользователя обеспечить его перенаправление на эту страницу после

успешной авторизации.

3. Добавить возможность авторизации по E-mail адресу и паролю.

4. Создать страницу с профилем пользователя. Обеспечить

возможность смены пароля в профиле.

5. Реализовать процедуру восстановления пароля с отправкой письма

со ссылкой для восстановления пароля на реальный почтовый ящик.

При переходе по ссылке из письма должна появляться форма

изменения пароля, и после ее заполнения пароль должен быть

изменен.

6. Создать группы разрешений и назначить некоторым пользователям

разрешения, некоторым – группы. При необходимости добавить

пользователей.

7. Создать пользовательское разрешение. Продемонстрировать его

наличие в списке разрешений в окне админ-панели редактирования

пользователей.

**Описание хода выполнения работы**

1. Создать приложение users для регистрации и авторизации

пользователей.

1.1. Инициализация приложения

В терминале, из корня проекта, выполняем команду:  
python3 manage.py startapp users

Это создаст в директории проекта новую папку users/ со стандартной структурой Django (модули models.py, views.py, apps.py и т. д.).

**1.2. Подключение в настройках**

Открываем settings.py и в секцию INSTALLED\_APPS добавляем строку:

'users.apps.UsersConfig' ,

Это «регистрирует» наше приложение, и Django начнёт искать в нём модели, миграции, шаблоны и т.п.

**1.3. Проверка Middleware**  
Убедимся, что в MIDDLEWARE присутствуют строки:

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

Они нужны для работы с сессиями и встроенной авторизацией.

**1.4. Маршрутизация приложения**

1. В папке users/ создаём файл urls.py со следующим содержимым:

*from* django.urls *import* path

*from* . *import* views

app\_name = 'users'

urlpatterns = [

    path('login/',     views.login\_user,  *name*='login'),

    path('logout/',    views.logout\_user, *name*='logout'),

]

1. В корневом urls.py проекта (рядом с admin/) подключаем его:

*from* django.contrib *import* admin

*from* django.urls *import* path, include

*from* django.conf *import* settings

*from* django.conf.urls.static *import* static

urlpatterns = [

    path('', include('cs.urls')),

    path('admin/', admin.site.urls),

    path('users/', include('users.urls', *namespace*='users')),

]

*if* settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, *document\_root*=settings.MEDIA\_ROOT)

**1.5. Заглушки-представления**  
Пока создадим «заглушки», чтобы убедиться, что маршруты работают:

*from* .forms *import* EmailOrUsernameAuthenticationForm

def login\_user(*request*):

*return* HttpResponse("login (заглушка)")

def logout\_user(*request*):

*return* HttpResponse("logout (заглушка)")

**1.6. Тестируем**

* Запускаем python manage.py runserver.
* В браузере открываем http://127.0.0.1:8000/users/login/ — должны увидеть «login».
* По адресу …/users/logout/ — «logout».

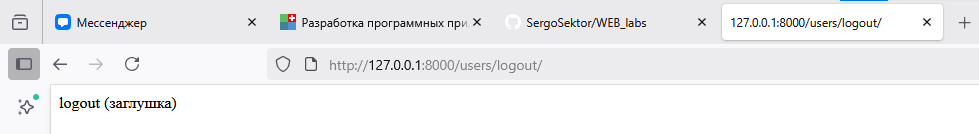
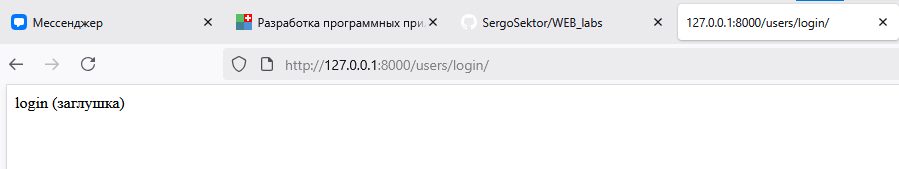


Рисунок 1 – отображение Logout.

  
Рисунок 2 – Отображение Login.

2. При попытке доступа к определенной странице неавторизованного

пользователя обеспечить его перенаправление на эту страницу после

успешной авторизации.

2.1. Настройка параметров в settings.py

Чтобы Django знал, на какую страницу отправлять неавторизованных пользователей и куда доставлять после входа, в settings.py добавим:

*# Путь к странице входа*

LOGIN\_URL = 'users:login'

*# Куда по умолчанию перенаправлять после успешного входа*

LOGIN\_REDIRECT\_URL = 'cs:home'

### 2.2. Реализация представлений login и logout

Вместо заглушек в users/views.py сделаем полноценные функции:

def login\_user(*request*):

    """

    Логин по e-mail или username с учётом next-параметра.

    """

    next\_url = *request*.GET.get('next') or *request*.POST.get('next') or ''

*if* *request*.method == 'POST':

        form = EmailOrUsernameAuthenticationForm(*request*, *data*=*request*.POST)

*if* form.is\_valid():

            login(*request*, form.get\_user())

*return* redirect(next\_url *or* 'cs:home')

*else*:

        form = EmailOrUsernameAuthenticationForm(*request*)

*return* render(*request*, 'users/login.html', {'form': form, 'next': next\_url})

def logout\_user(*request*):

    """

    Выход и редирект на главную.

    """

    logout(*request*)

*return* redirect('cs:home')

### 2.3. Шаблон страницы входа

Создаем файл users/templates/users/login.html:

{% extends 'cs/base.html' %}

{% block title %}Вход{% endblock %}

{% block content %}

  <h2>Вход на сайт</h2>

  <form method="post">

    {% csrf\_token %}

    {{ form.non\_field\_errors }}

    <p>

      {{ form.username.label\_tag }}<br>

      {{ form.username }}<br>

      {{ form.username.errors }}

    </p>

    <p>

      {{ form.password.label\_tag }}<br>

      {{ form.password }}<br>

      {{ form.password.errors }}

    </p>

*{# Скрытое поле next сохраняет URL запроса #}*

    <input type="hidden" name="next" value="{{ next }}">

    <button type="submit">Войти</button>

  </form>

{% endblock %}

### 2.4. Ограничение доступа с помощью миксина LoginRequiredMixin

Теперь нам нужно пометить те страницы, доступ к которым разрешён только авторизованным. Предположим, что создание и редактирование концепции (AddConceptCustomView, ConceptCreateView, ConceptUpdateView, ConceptDeleteView) должно требовать входа.

Откроем cs/views.py и для этих классов подключим LoginRequiredMixin:

*from* django.urls *import* reverse\_lazy

*from* django.views.generic *import* (

    TemplateView, ListView, DetailView,

    FormView, CreateView, UpdateView, DeleteView

)

*from* django.contrib.auth.mixins *import* LoginRequiredMixin

*from* .models *import* ComputerScienceConcept, FieldOfStudy, Tag

*from* .forms *import* ConceptForm, ConceptModelForm, UploadForm

*from* .utils *import* DataMixin

class HomeView(DataMixin, ListView):

    model = ComputerScienceConcept

    template\_name = 'cs/index.html'

    context\_object\_name = 'concepts'

    paginate\_by = 5

    queryset = ComputerScienceConcept.published.all()

    title = 'Главная'

class AboutView(DataMixin, TemplateView):

    template\_name = 'cs/about.html'

    title = 'О сайте'

class ConceptDetailView(DataMixin, DetailView):

    model = ComputerScienceConcept

    template\_name = 'cs/concept\_detail.html'

    context\_object\_name = 'concept'

    slug\_field = 'slug'

    slug\_url\_kwarg = 'concept\_slug'

    def get\_context\_data(*self*, \*\**kwargs*):

        context = super().get\_context\_data(\*\**kwargs*)

*# Заголовок страницы — название концепции*

        context['title'] = context['concept'].title

*return* context

class AddConceptCustomView(LoginRequiredMixin, DataMixin, FormView):

    form\_class = ConceptForm

    template\_name = 'cs/add\_concept\_custom.html'

    success\_url = reverse\_lazy('cs:home')

    title = 'Добавить (Form)'

    login\_url = reverse\_lazy('users:login')

    def form\_valid(*self*, *form*):

        cd = *form*.cleaned\_data

        concept = ComputerScienceConcept.objects.create(

*title*=cd['title'],

*slug*=cd['title'].lower().replace(' ', '-'),

*description*=cd['description'],

*difficulty*=cd['difficulty'],

*is\_published*=ComputerScienceConcept.Status.DRAFT

        )

        image = cd.get('image')

*if* image:

            concept.image = image

            concept.save()

*return* super().form\_valid(*form*)

class ConceptCreateView(LoginRequiredMixin, DataMixin, CreateView):

    model = ComputerScienceConcept

    form\_class = ConceptModelForm

    template\_name = 'cs/add\_concept\_model.html'

    success\_url = reverse\_lazy('cs:home')

    title = 'Добавить (ModelForm)'

    login\_url = reverse\_lazy('users:login')

class ConceptUpdateView(LoginRequiredMixin, DataMixin, UpdateView):

    model = ComputerScienceConcept

    form\_class = ConceptModelForm

    template\_name = 'cs/add\_concept\_model.html'

    success\_url = reverse\_lazy('cs:home')

    slug\_field = 'slug'

    slug\_url\_kwarg = 'concept\_slug'

    title = 'Редактировать концепцию'

    login\_url = reverse\_lazy('users:login')

class ConceptDeleteView(LoginRequiredMixin, DataMixin, DeleteView):

    model = ComputerScienceConcept

    template\_name = 'cs/concept\_confirm\_delete.html'

    success\_url = reverse\_lazy('cs:home')

    slug\_field = 'slug'

    slug\_url\_kwarg = 'concept\_slug'

    title = 'Удалить концепцию'

    login\_url = reverse\_lazy('users:login')

class UploadFileView(DataMixin, FormView):

    form\_class = UploadForm

    template\_name = 'cs/upload.html'

    success\_url = reverse\_lazy('cs:upload\_file')

    title = 'Загрузка файла'

    def form\_valid(*self*, *form*):

*form*.save\_file()

*return* super().form\_valid(*form*)

class FieldOfStudyDetailView(DataMixin, ListView):

    model = ComputerScienceConcept

    template\_name = 'cs/field\_of\_study\_detail.html' *# Шаблон для отображения концепций по области*

    context\_object\_name = 'concepts'

    paginate\_by = 5

    def get\_queryset(*self*):

*# Получаем слаг области науки из URL*

        field\_of\_study\_slug = *self*.kwargs['field\_of\_study\_slug']

*# Фильтруем концепции по этой области*

*return* ComputerScienceConcept.published.filter(*field\_of\_study\_\_slug*=field\_of\_study\_slug)

    def get\_context\_data(*self*, \*\**kwargs*):

        context = super().get\_context\_data(\*\**kwargs*)

*# Получаем объект FieldOfStudy, чтобы отобразить его название*

        field\_of\_study\_slug = *self*.kwargs['field\_of\_study\_slug']

        field\_of\_study = FieldOfStudy.objects.get(*slug*=field\_of\_study\_slug)

        context['title'] = f"Концепции в области: {field\_of\_study.name}"

        context['field\_of\_study'] = field\_of\_study

*return* context

После этого сделаем проверку перейдя по ссылке:  
  
http://127.0.0.1:8000/add-model/  
Появится форма входа (users/login/?next=/add-model/).  
После ввода валидных учётных данных пользователя вы попадёте на /add-model/ — то есть на именно ту страницу, к которой хотели доступ.

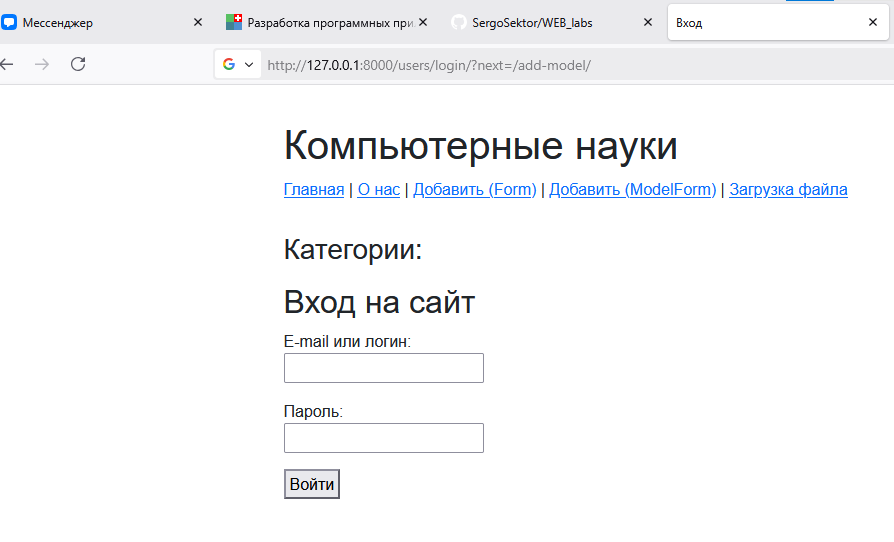
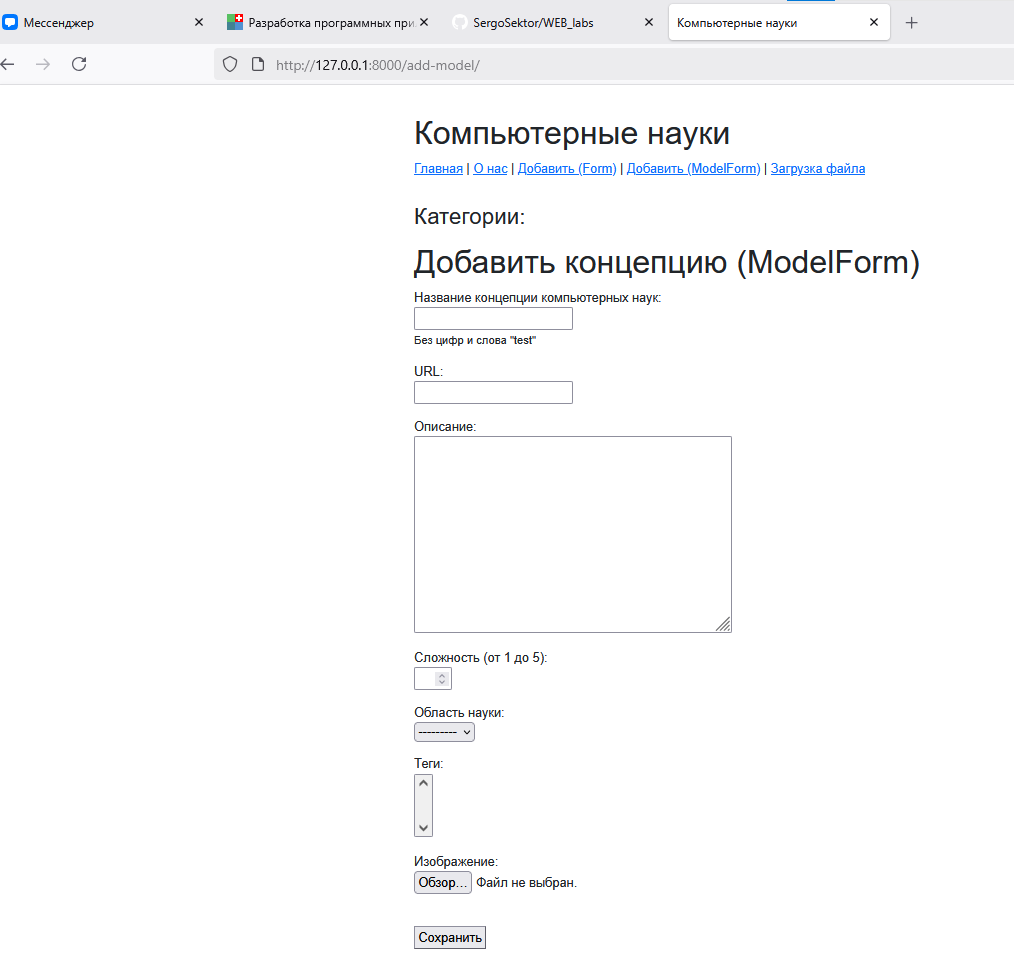


Рисунок 3 – Форма логина.

  
Рисунок 4 – Форма после ввода логина и пароля.

3. Добавить возможность авторизации по E-mail адресу и паролю.

3.1. Создание бэкенда аутентификации по E-mail

Файл: users/backends.py

*from* django.contrib.auth.backends *import* ModelBackend

*from* django.contrib.auth *import* get\_user\_model

User = get\_user\_model()

class EmailOrUsernameBackend(ModelBackend):

    """

    Аутентификация: сначала пробуем по email, если не нашли — по username.

    """

    def authenticate(*self*, *request*, *username*=None, *password*=None, \*\**kwargs*):

*if* *username* is None or *password* is None:

*return* None

        user = None

*# попробуем e-mail*

*if* '@' in *username*:

            user = User.objects.filter(*email\_\_iexact*=*username*, *is\_active*=True).first()

*# если не нашли по e-mail, попробуем username*

*if* user is None:

            user = User.objects.filter(*username\_\_iexact*=*username*, *is\_active*=True).first()

*if* user and user.check\_password(*password*):

*return* user

*return* None

### 3.2. Регистрация бэкенда в настройках

**Файл:** cs\_ty/settings.py

AUTHENTICATION\_BACKENDS = [

    'users.backends.EmailOrUsernameBackend',

 ]

### 3.3. Переопределение формы логина

Чтобы в шаблоне и в представлении не путаться, создадим собственную форму:

**Файл:** users/forms.py

*from* django *import* forms

*from* django.contrib.auth.forms *import* AuthenticationForm

class EmailOrUsernameAuthenticationForm(AuthenticationForm):

    """

    Форма логина: в одном поле пользователь вводит либо e-mail, либо username.

    """

    username = forms.CharField(

*label*='E-mail или логин',

*widget*=forms.TextInput(*attrs*={'autofocus': True})

    )

### 3.4. Обновление представления входа

**Файл:** users/views.py

*from* django.contrib.auth.views *import* (

    PasswordResetView, PasswordResetDoneView,

    PasswordResetConfirmView, PasswordResetCompleteView

)

*from* django.urls *import* reverse\_lazy

*from* django.contrib.auth *import* login, logout

*from* django.shortcuts *import* render, redirect

*from* django.contrib.auth.mixins *import* LoginRequiredMixin

*from* django.views.generic *import* TemplateView

*from* django.contrib.auth.views *import* PasswordChangeView, PasswordChangeDoneView

*from* .forms *import* EmailOrUsernameAuthenticationForm

def login\_user(*request*):

    """

    Логин по e-mail или username с учётом next-параметра.

    """

    next\_url = *request*.GET.get('next') or *request*.POST.get('next') or ''

*if* *request*.method == 'POST':

        form = EmailOrUsernameAuthenticationForm(*request*, *data*=*request*.POST)

*if* form.is\_valid():

            login(*request*, form.get\_user())

*return* redirect(next\_url *or* 'cs:home')

*else*:

        form = EmailOrUsernameAuthenticationForm(*request*)

*return* render(*request*, 'users/login.html', {'form': form, 'next': next\_url})

def logout\_user(*request*):

    """

    Выход и редирект на главную.

    """

    logout(*request*)

*return* redirect('cs:home')

### 3.5. Обновление шаблона входа

**Файл:** users/templates/users/login.html

{% extends 'cs/base.html' %}

{% block title %}Вход{% endblock %}

{% block content %}

  <h2>Вход на сайт</h2>

  <form method="post">

    {% csrf\_token %}

    {{ form.non\_field\_errors }}

    <p>

      {{ form.username.label\_tag }}<br>

      {{ form.username }}<br>

      {{ form.username.errors }}

    </p>

    <p>

      {{ form.password.label\_tag }}<br>

      {{ form.password }}<br>

      {{ form.password.errors }}

    </p>

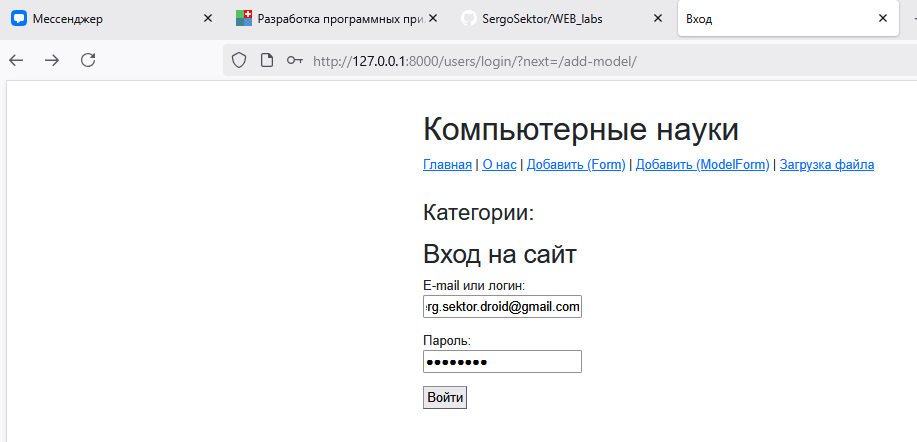
*{# Скрытое поле next сохраняет URL запроса #}*

    <input type="hidden" name="next" value="{{ next }}">

    <button type="submit">Войти</button>

  </form>

{% endblock %}

Теперь попробуем зайти по email Рисунок 5 – Вход по email

4. Создать страницу с профилем пользователя. Обеспечить

возможность смены пароля в профиле.

Мы добавим в приложение users три новых представления:

* ProfileView – отображает данные текущего пользователя.
* CustomPasswordChangeView – форма смены пароля.
* CustomPasswordChangeDoneView – страница после успешной смены.

Также добавим соответствующие URL-маршруты и шаблоны.  
  
4.1. Обновляем users/views.py

class ProfileView(LoginRequiredMixin, TemplateView):

    """

    Страница профиля текущего пользователя.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/profile.html'

    def get\_context\_data(*self*, \*\**kwargs*):

        ctx = super().get\_context\_data(\*\**kwargs*)

        ctx['user'] = *self*.request.user

*return* ctx

class CustomPasswordChangeView(LoginRequiredMixin, PasswordChangeView):

    """

    Форма смены пароля.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/password\_change\_form.html'

    success\_url = reverse\_lazy('users:password\_change\_done')

class CustomPasswordChangeDoneView(LoginRequiredMixin, PasswordChangeDoneView):

    """

    Страница подтверждения смены пароля.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/password\_change\_done.html'

class CustomPasswordResetView(PasswordResetView):

    """

    Шаг 1: форма запроса сброса пароля (ввод e-mail, отправка письма).

    """

    template\_name = 'users/password\_reset\_form.html'

    email\_template\_name = 'users/password\_reset\_email.html'

    subject\_template\_name = 'users/password\_reset\_subject.txt'

    success\_url = reverse\_lazy('users:password\_reset\_done')

class CustomPasswordResetDoneView(PasswordResetDoneView):

    """

    Шаг 2: страница, показываемая после отправки письма.

    """

    template\_name = 'users/password\_reset\_done.html'

4.2 Обновляем маршруты в urls.py

*from* django.urls *import* path

*from* . *import* views

app\_name = 'users'

urlpatterns = [

    path('login/',     views.login\_user,  *name*='login'),

    path('logout/',    views.logout\_user, *name*='logout'),

*# Сброс пароля:*

    path('password-reset/',

         views.CustomPasswordResetView.as\_view(),

*name*='password\_reset'),

    path('password-reset/done/',

         views.CustomPasswordResetDoneView.as\_view(),

*name*='password\_reset\_done'),

    path('reset/<uidb64>/<token>/',

         views.CustomPasswordResetConfirmView.as\_view(),

*name*='password\_reset\_confirm'),

    path('reset/done/',

         views.CustomPasswordResetCompleteView.as\_view(),

*name*='password\_reset\_complete'),

*# Профиль и смена пароля:*

    path('profile/',

         views.ProfileView.as\_view(),

*name*='profile'),

    path('password-change/',

         views.CustomPasswordChangeView.as\_view(),

*name*='password\_change'),

    path('password-change/done/',

         views.CustomPasswordChangeDoneView.as\_view(),

*name*='password\_change\_done'),

]

4.3 Создаем шаблоны  
В каталоге users/templates/users/ создаём три файла:

* **profile.html** – показывает имя и e-mail пользователя, ссылку «Сменить пароль».
* **password\_change\_form.html** – форма ввода старого и нового паролей.
* **password\_change\_done.html** – сообщение об успешной смен

users/templates/users/profile.html

{% extends 'cs/base.html' %}

{% block title %}Профиль{% endblock %}

{% block content %}

  <h2>Ваш профиль</h2>

  <p><strong>Логин:</strong> {{ user.username }}</p>

  <p><strong>E-mail:</strong> {{ user.email }}</p>

  <p><a href="{% url 'users:password\_change' %}">Сменить пароль</a></p>

{% endblock %}

users/templates/users/password\_change\_form.html

{% extends 'cs/base.html' %}

{% block title %}Смена пароля{% endblock %}

{% block content %}

  <h2>Сменить пароль</h2>

  <form method="post">

    {% csrf\_token %}

    {{ form.non\_field\_errors }}

    <p>{{ form.old\_password.label\_tag }}<br>{{ form.old\_password }}{{ form.old\_password.errors }}</p>

    <p>{{ form.new\_password1.label\_tag }}<br>{{ form.new\_password1 }}{{ form.new\_password1.errors }}</p>

    <p>{{ form.new\_password2.label\_tag }}<br>{{ form.new\_password2 }}{{ form.new\_password2.errors }}</p>

    <button type="submit">Сменить пароль</button>

  </form>

{% endblock %}

users/templates/users/password\_change\_done.html

{% extends 'cs/base.html' %}

{% block title %}Пароль изменён{% endblock %}

{% block content %}

  <h2>Пароль успешно обновлён!</h2>

  <p><a href="{% url 'users:profile' %}">Вернуться в профиль</a></p>

{% endblock %}

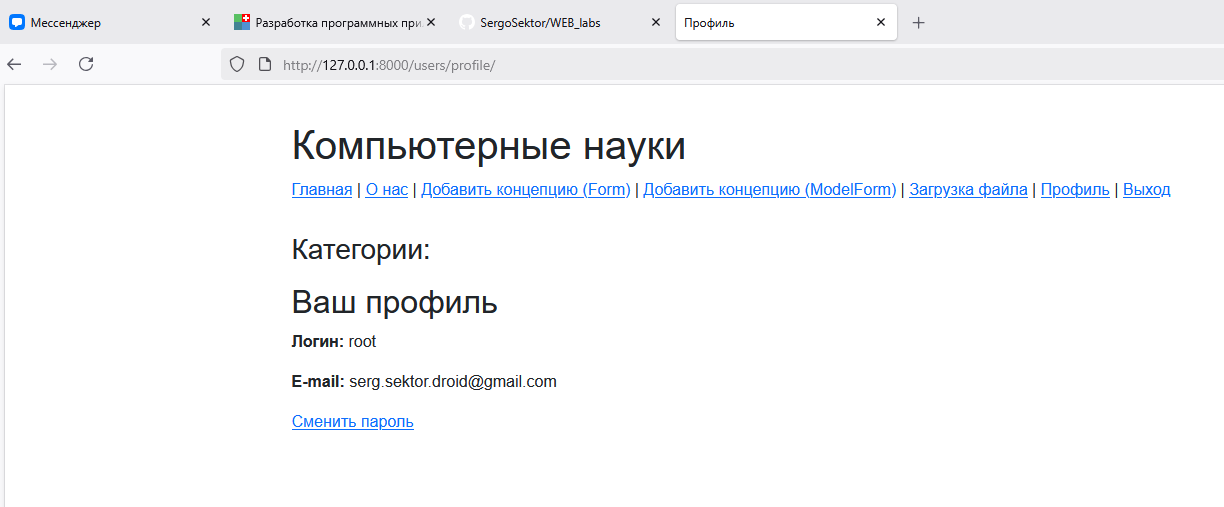
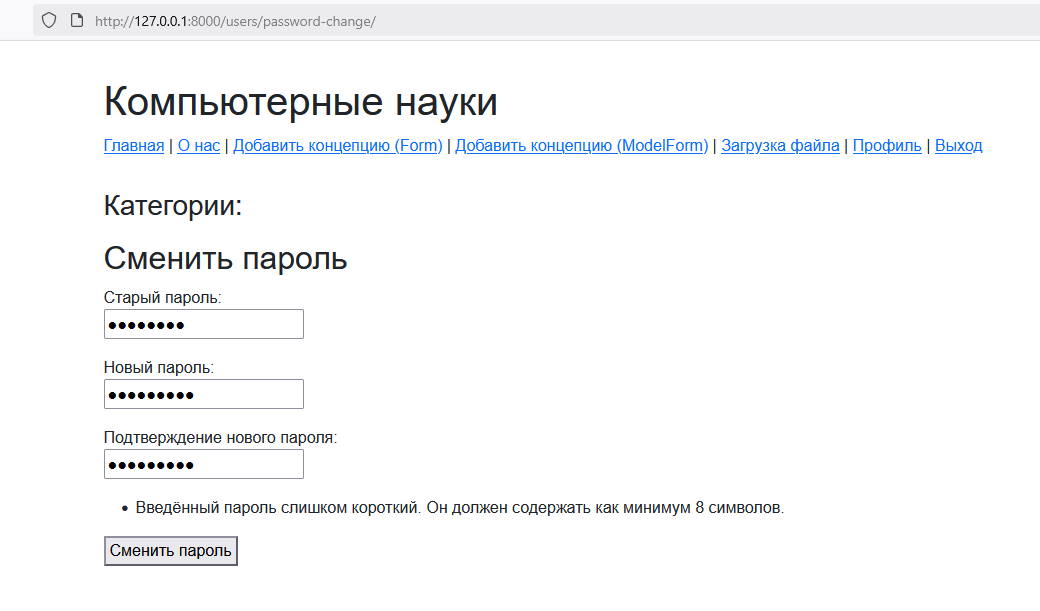
Рисунок 6 – Профиль.  


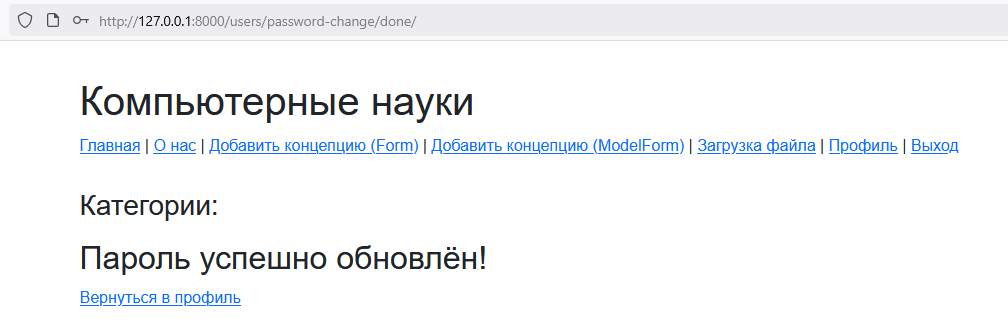
Рисунок 7 – Смена пароля.  


Рисунок 8 – Уведомление о смене.

5. Реализовать процедуру восстановления пароля с отправкой письма

со ссылкой для восстановления пароля на реальный почтовый ящик.

При переходе по ссылке из письма должна появляться форма

изменения пароля, и после ее заполнения пароль должен быть

изменен.

5.1 в settings.py нужно добавить параметры smtp сервера

EMAIL\_BACKEND       = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_HOST          = 'smtp.yandex.ru'

EMAIL\_PORT          = 587

EMAIL\_USE\_TLS       = True

EMAIL\_HOST\_USER     = 'email'

EMAIL\_HOST\_PASSWORD = 'smtp\_password'

DEFAULT\_FROM\_EMAIL  = 'Computer Science Concept <mail@yandex.ru>'

### 5.2 Добавление представлений сброса пароля

1. Открываем файл users/views.py.
2. В начало добавляем импорт встроенных классов сброса пароля:
3. После уже существующих классов добавляем четыре новых для отправки письма, запроса сброса, установки нового пароля, успешного восстановления:

*from* django.contrib.auth.views *import* (

    PasswordResetView, PasswordResetDoneView,

    PasswordResetConfirmView, PasswordResetCompleteView

)

*from* django.urls *import* reverse\_lazy

*from* django.contrib.auth *import* login, logout

*from* django.shortcuts *import* render, redirect

*from* django.contrib.auth.mixins *import* LoginRequiredMixin

*from* django.views.generic *import* TemplateView

*from* django.contrib.auth.views *import* PasswordChangeView, PasswordChangeDoneView

*from* .forms *import* EmailOrUsernameAuthenticationForm

def login\_user(*request*):

    """

    Логин по e-mail или username с учётом next-параметра.

    """

    next\_url = *request*.GET.get('next') or *request*.POST.get('next') or ''

*if* *request*.method == 'POST':

        form = EmailOrUsernameAuthenticationForm(*request*, *data*=*request*.POST)

*if* form.is\_valid():

            login(*request*, form.get\_user())

*return* redirect(next\_url *or* 'cs:home')

*else*:

        form = EmailOrUsernameAuthenticationForm(*request*)

*return* render(*request*, 'users/login.html', {'form': form, 'next': next\_url})

def logout\_user(*request*):

    """

    Выход и редирект на главную.

    """

    logout(*request*)

*return* redirect('cs:home')

class ProfileView(LoginRequiredMixin, TemplateView):

    """

    Страница профиля текущего пользователя.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/profile.html'

    def get\_context\_data(*self*, \*\**kwargs*):

        ctx = super().get\_context\_data(\*\**kwargs*)

        ctx['user'] = *self*.request.user

*return* ctx

class CustomPasswordChangeView(LoginRequiredMixin, PasswordChangeView):

    """

    Форма смены пароля.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/password\_change\_form.html'

    success\_url = reverse\_lazy('users:password\_change\_done')

class CustomPasswordChangeDoneView(LoginRequiredMixin, PasswordChangeDoneView):

    """

    Страница подтверждения смены пароля.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/password\_change\_done.html'

class CustomPasswordResetView(PasswordResetView):

    """

    Шаг 1: форма запроса сброса пароля (ввод e-mail, отправка письма).

    """

    template\_name = 'users/password\_reset\_form.html'

    email\_template\_name = 'users/password\_reset\_email.html'

    subject\_template\_name = 'users/password\_reset\_subject.txt'

    success\_url = reverse\_lazy('users:password\_reset\_done')

class CustomPasswordResetDoneView(PasswordResetDoneView):

    """

    Шаг 2: страница, показываемая после отправки письма.

    """

    template\_name = 'users/password\_reset\_done.html'

class CustomPasswordResetConfirmView(PasswordResetConfirmView):

    """

    Шаг 3: форма установки нового пароля (переход по ссылке из письма).

    """

    template\_name = 'users/password\_reset\_confirm.html'

    success\_url = reverse\_lazy('users:password\_reset\_complete')

class CustomPasswordResetCompleteView(PasswordResetCompleteView):

    """

    Шаг 4: страница, показываемая после успешного сброса пароля.

    """

    template\_name = 'users/password\_reset\_complete.html'

5.3 Обновляем маршруты в urls.py

*from* django.urls *import* path

*from* . *import* views

app\_name = 'users'

urlpatterns = [

    path('login/',     views.login\_user,  *name*='login'),

    path('logout/',    views.logout\_user, *name*='logout'),

*# Сброс пароля:*

    path('password-reset/',

         views.CustomPasswordResetView.as\_view(),

*name*='password\_reset'),

    path('password-reset/done/',

         views.CustomPasswordResetDoneView.as\_view(),

*name*='password\_reset\_done'),

    path('reset/<uidb64>/<token>/',

         views.CustomPasswordResetConfirmView.as\_view(),

*name*='password\_reset\_confirm'),

    path('reset/done/',

         views.CustomPasswordResetCompleteView.as\_view(),

*name*='password\_reset\_complete'),

*# Профиль и смена пароля:*

    path('profile/',

         views.ProfileView.as\_view(),

*name*='profile'),

    path('password-change/',

         views.CustomPasswordChangeView.as\_view(),

*name*='password\_change'),

    path('password-change/done/',

         views.CustomPasswordChangeDoneView.as\_view(),

*name*='password\_change\_done'),

]

5.4 Создаем шаблоны для восстановления пароля

Password\_reset\_form.html

{% extends 'cs/base.html' %}

{% block title %}Восстановление пароля{% endblock %}

{% block content %}

  <h2>Восстановление пароля</h2>

  <form method="post">{% csrf\_token %}

    {{ form.email.label\_tag }}<br>

    {{ form.email }}<br>

    {{ form.email.errors }}<br><br>

    <button type="submit">Отправить письмо</button>

  </form>

{% endblock %}

password\_reset\_done.html

{% extends 'cs/base.html' %}

{% block title %}Письмо отправлено{% endblock %}

{% block content %}

  <h2>Письмо с инструкциями отправлено!</h2>

  <p>Проверьте вашу почту.</p>

{% endblock %}

Password\_reset\_email.html

{% autoescape off %}

Чтобы сбросить пароль, перейдите по ссылке:

{{ protocol }}://{{ domain }}{% url 'users:password\_reset\_confirm' uidb64=uid token=token %}

{% endautoescape %}

password\_reset\_subject.txt  
  
Восстановление пароля на сайте «Computer Science Concept»

password\_reset\_confirm.html

{% extends 'cs/base.html' %}

{% block title %}Установка нового пароля{% endblock %}

{% block content %}

  <h2>Установите новый пароль</h2>

  <form method="post">{% csrf\_token %}

    {{ form.new\_password1.label\_tag }}<br>

    {{ form.new\_password1 }}<br><br>

    {{ form.new\_password2.label\_tag }}<br>

    {{ form.new\_password2 }}<br><br>

    <button type="submit">Сохранить</button>

  </form>

{% endblock %}

password\_reset\_complete.html

{% extends 'cs/base.html' %}

{% block title %}Пароль сброшен{% endblock %}

{% block content %}

  <h2>Ваш пароль успешно изменён!</h2>

  <p><a href="{% url 'users:login' %}">Войти</a></p>

{% endblock %}

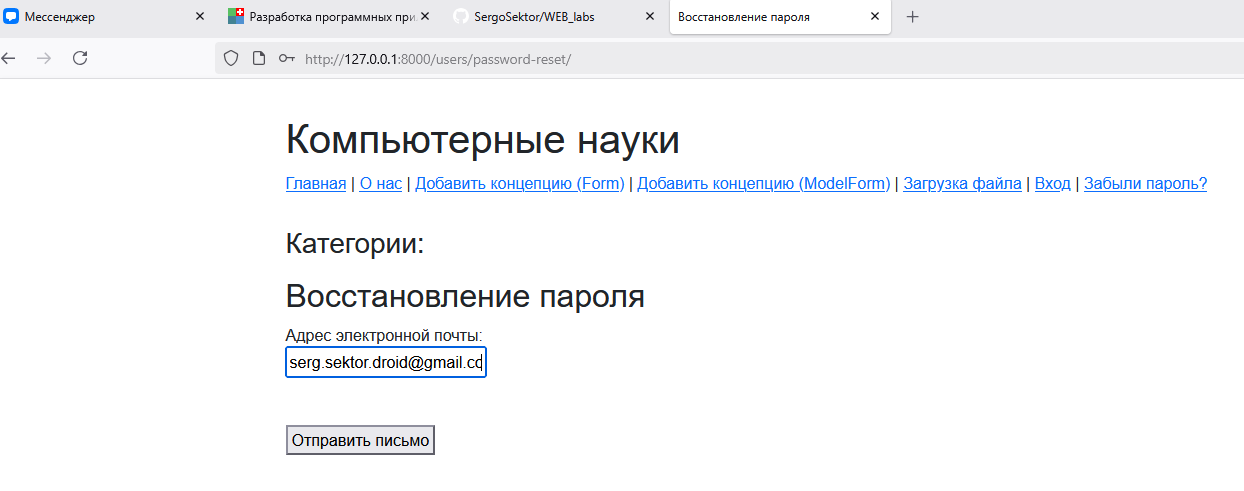


Рисунок 9 – Восстановлеие пароля.

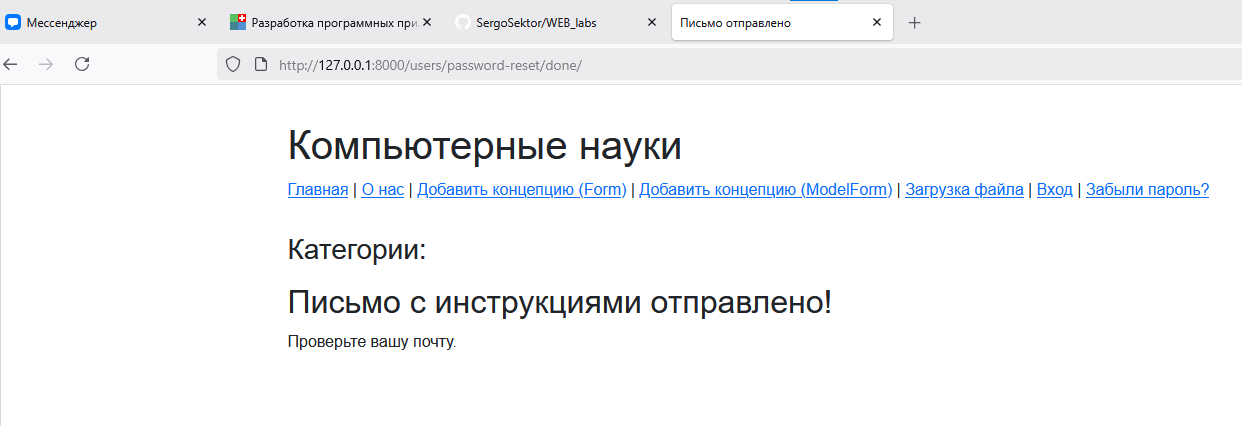
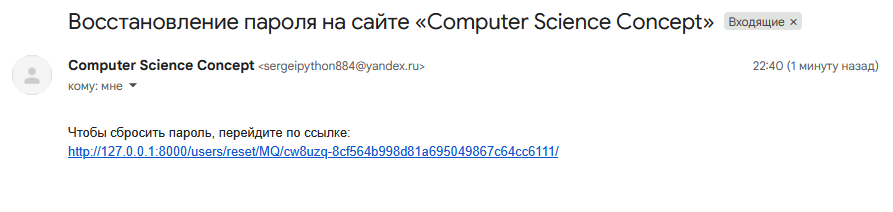
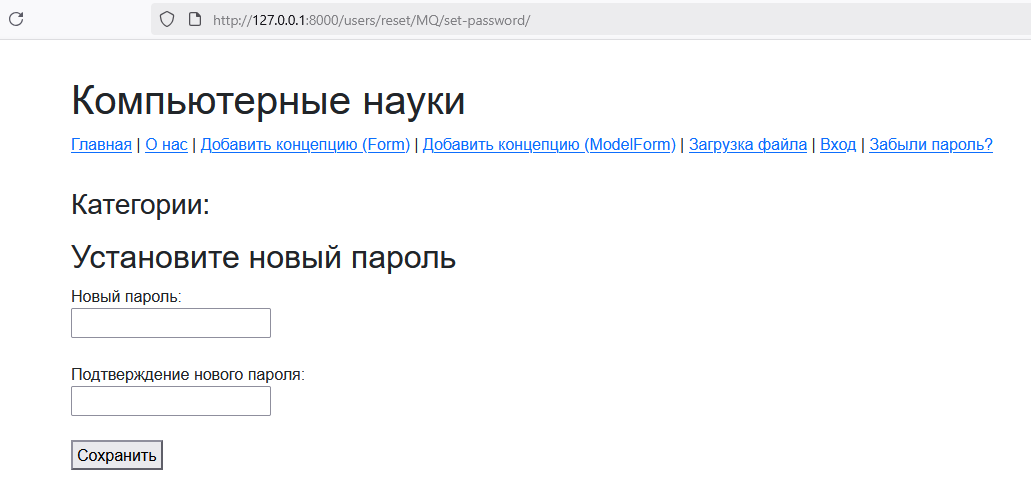
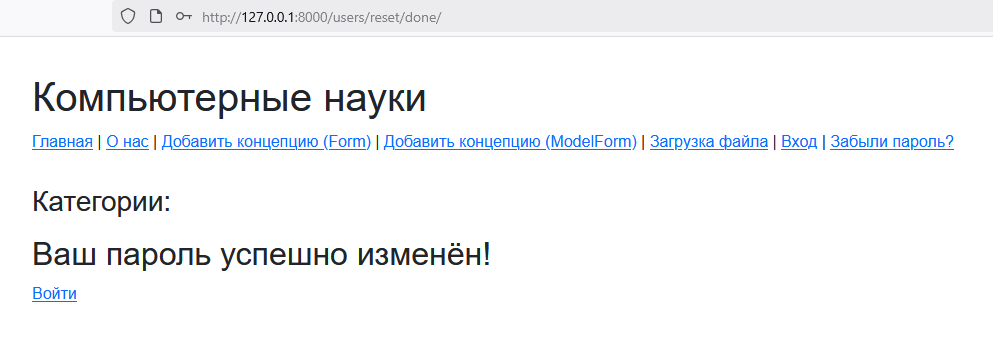
Рисунок 10 – Инструкция по восстановлению.  
Рисунок 11 – Письмо на почту.  


Рисунок 12 – Форма восстановления.  
  
Рисунок 13 – Пароль восстановлен.

После изменения пароль действительно обновился и стал актуальным.

6. Создать группы разрешений и назначить некоторым пользователям

разрешения, некоторым – группы. При необходимости добавить

пользователей.  
  
Для этого зайдем в Django shell и добавим пользователей права и группы:

*from* django.contrib.auth.models *import* Group, Permission, User

*from* cs.models *import* ComputerScienceConcept *# Импортируем нашу модель ComputerScienceConcept*

*# 1) Создадим две группы*

editors = Group.objects.create(*name*='Editors')

viewers = Group.objects.create(*name*='Viewers')

*# 2) Найдём нужные разрешения (Permissions) по codename*

*#    Например, право изменять концепции ("change\_computerscienceconcept") и право смотреть концепции ("view\_computerscienceconcept"):*

perm\_change\_concept = Permission.objects.get(*codename*='change\_computerscienceconcept')

perm\_view\_concept   = Permission.objects.get(*codename*='view\_computerscienceconcept')

perm\_publish\_concept = Permission.objects.get(*codename*='can\_publish\_concept') *# Пользовательское разрешение*

*# 3) Добавим эти права в группу Editors, а только просмотр — в группу Viewers*

editors.permissions.add(perm\_change\_concept, perm\_view\_concept, perm\_publish\_concept) *# Добавляем пользовательское разрешение*

viewers.permissions.add(perm\_view\_concept)

*# 4) Создадим нового пользователя*

bob = User.objects.create\_user(

*username*='bob', *email*='bob@example.com', *password*='Sss12345'

)

*# 5) Назначим Bob в группу Editors*

bob.groups.add(editors)

alice = User.objects.create\_user(

*username*='alice',

*email*='alice@example.com',

*password*='Sss12345'

)

*# 6) Для другого пользователя (alice) дадим индивидуальное разрешение*

*# Предположим, мы хотим, чтобы alice могла удалять концепции:*

perm\_delete\_concept = Permission.objects.get(*codename*='delete\_computerscienceconcept')

alice.user\_permissions.add(perm\_delete\_concept)

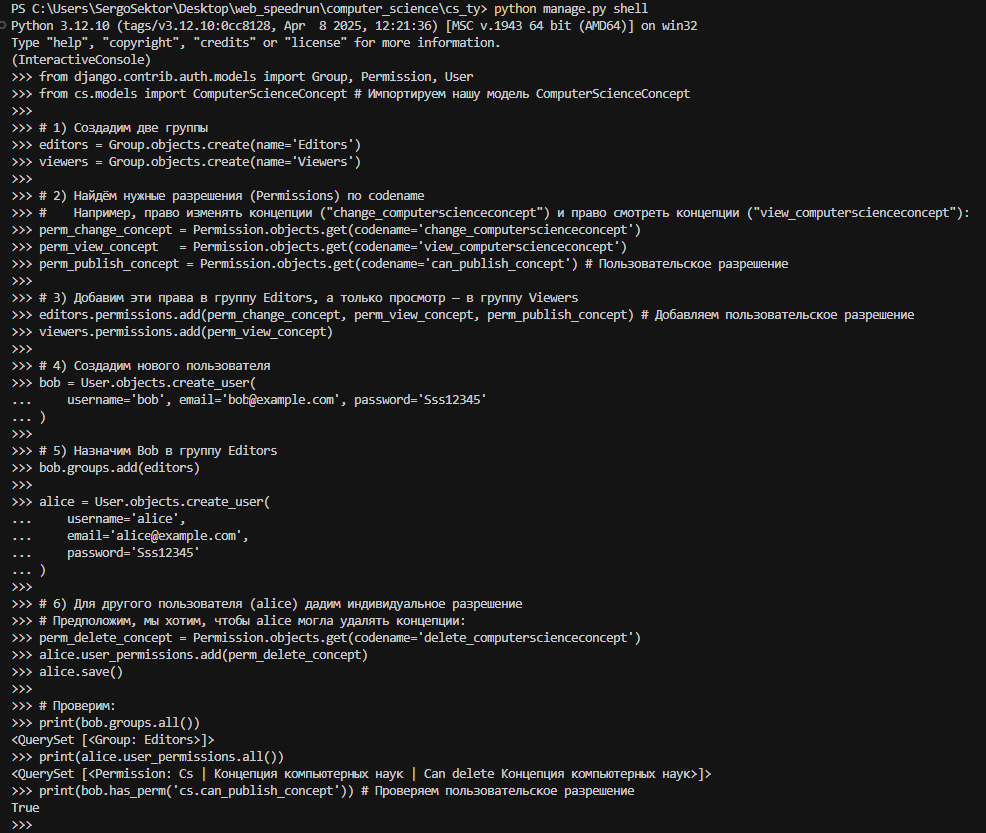
alice.save()

*# Проверим:*

print(bob.groups.all())

print(alice.user\_permissions.all())

print(bob.has\_perm('cs.can\_publish\_concept')) *# Проверяем пользовательское разрешение*

  
Рисунок 14 – Проверка прав.

7. Создать пользовательское разрешение. Продемонстрировать его

наличие в списке разрешений в окне админ-панели редактирования

пользователей.

Откроем файл cs/models.py в классе concept и секции class meta добавим туда атрибут permissions со списком кортежей:

   class Meta:

        permissions = [

            ('can\_publish\_concept', 'Может публиковать концепцию'),

        ]

        ordering = ['-time\_create']

        indexes = [models.Index(*fields*=['-time\_create'])]

        verbose\_name = "Концепция компьютерных наук"

        verbose\_name\_plural = "Концепции компьютерных наук"

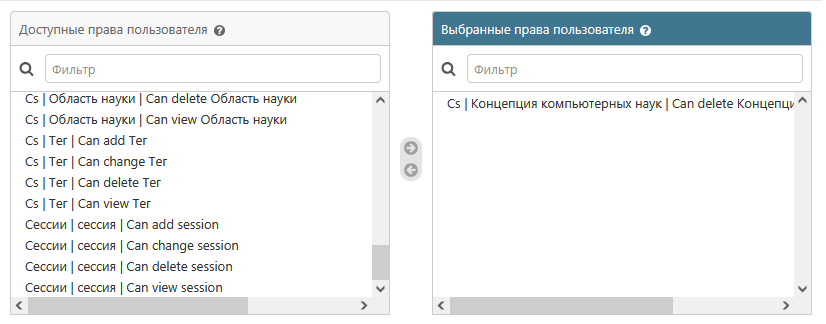


Рисунок 15 - Пользовательское разрешение.

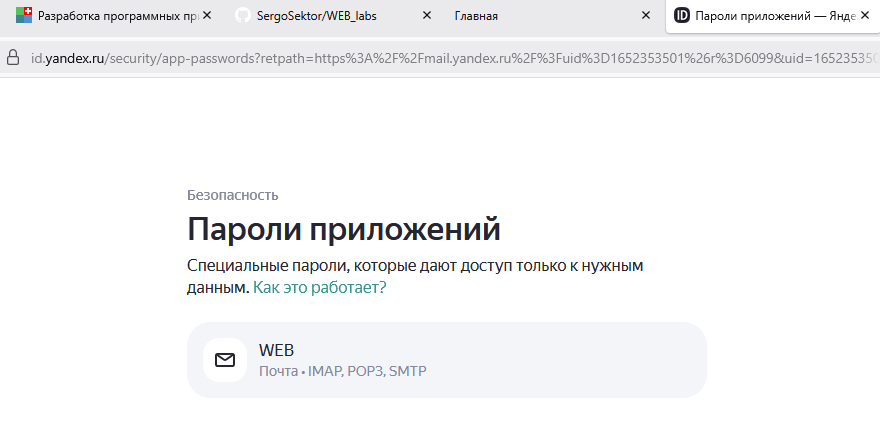


Рисунок 16 – Пароль приложения на Яндекс почте, необходимый для взаимодействия с приложением.

**Листинг программ**

cs/models.py

*from* django.db *import* models

*from* django.urls *import* reverse

*from* django.utils.text *import* slugify *# Импортируем slugify*

*# Модель для областей компьютерных наук*

class FieldOfStudy(models.Model):

    name = models.CharField(*max\_length*=255, *verbose\_name*="Область науки")

    slug = models.SlugField(*max\_length*=255, *unique*=True, *db\_index*=True, *verbose\_name*="URL")

    description = models.TextField(*blank*=True, *verbose\_name*="Описание области")

    class Meta:

        verbose\_name = "Область науки"

        verbose\_name\_plural = "Области науки"

    def \_\_str\_\_(*self*):

*return* *self*.name

    def save(*self*, \**args*, \*\**kwargs*):

*if* not *self*.slug:

*self*.slug = slugify(*self*.name)

*# Проверяем уникальность слага*

            base\_slug = *self*.slug

            num = 1

*while* FieldOfStudy.objects.filter(*slug*=*self*.slug).exists():

*self*.slug = f"{base\_slug}-{num}"

                num += 1

        super().save(\**args*, \*\**kwargs*)

    def get\_absolute\_url(*self*):

*return* reverse('cs:field\_of\_study\_detail', *kwargs*={'field\_of\_study\_slug': *self*.slug})

*# Пользовательский менеджер для выборки только опубликованных записей*

class PublishedManager(models.Manager):

    def get\_queryset(*self*):

*# Фильтруем записи по полю публикации с использованием перечисления*

*return* super().get\_queryset().filter(*is\_published*=ComputerScienceConcept.Status.PUBLISHED)

class ComputerScienceConcept(models.Model):

*# Класс-перечисление для статуса публикации*

    class Status(models.IntegerChoices):

        DRAFT = 0, 'Черновик'

        PUBLISHED = 1, 'Опубликовано'

    title = models.CharField(*max\_length*=255, *verbose\_name*="Название концепции")

    slug = models.SlugField(*max\_length*=255, *unique*=True, *db\_index*=True, *verbose\_name*="URL")

    description = models.TextField(*blank*=True, *verbose\_name*="Описание")

    difficulty = models.IntegerField(default=1, *verbose\_name*="Сложность (от 1 до 5)") *# Новое поле для сложности*

    time\_create = models.DateTimeField(*auto\_now\_add*=True, *verbose\_name*="Время создания")

    time\_update = models.DateTimeField(*auto\_now*=True, *verbose\_name*="Время обновления")

    is\_published = models.BooleanField(*choices*=Status.choices, default=Status.DRAFT, *verbose\_name*="Публикация")

    image = models.ImageField(*upload\_to*='concept\_images/%Y/%m/%d/', *blank*=True, *null*=True, *verbose\_name*='Изображение') *# New field*

*# Связь один-ко-многим с FieldOfStudy*

    field\_of\_study = models.ForeignKey(

        FieldOfStudy,

*on\_delete*=models.CASCADE,

*related\_name*='concepts',

*verbose\_name*="Область науки",

*null*=True,

*blank*=True

    )

    objects = models.Manager()         *# Стандартный менеджер*

    published = PublishedManager()       *# Пользовательский менеджер для опубликованных записей*

    class Meta:

        permissions = [

            ('can\_publish\_concept', 'Может публиковать концепцию'),

        ]

        ordering = ['-time\_create']

        indexes = [models.Index(*fields*=['-time\_create'])]

        verbose\_name = "Концепция компьютерных наук"

        verbose\_name\_plural = "Концепции компьютерных наук"

    def \_\_str\_\_(*self*):

*return* *self*.title

    def save(*self*, \**args*, \*\**kwargs*):

*if* not *self*.slug:

*self*.slug = slugify(*self*.title)

*# Проверяем уникальность слага*

            base\_slug = *self*.slug

            num = 1

*while* ComputerScienceConcept.objects.filter(*slug*=*self*.slug).exists():

*self*.slug = f"{base\_slug}-{num}"

                num += 1

        super().save(\**args*, \*\**kwargs*)

    def get\_absolute\_url(*self*):

*return* reverse('cs:concept\_detail', *kwargs*={'concept\_slug': *self*.slug})

*# Модель для расширенной информации о концепции (OneToOne)*

class ConceptDetail(models.Model):

    concept = models.OneToOneField(

        ComputerScienceConcept,

*on\_delete*=models.CASCADE,

*related\_name*='detail',

*verbose\_name*="Концепция"

    )

    core\_technologies = models.TextField(*blank*=True, *verbose\_name*="Ключевые технологии")

    prerequisites = models.TextField(*blank*=True, *verbose\_name*="Предварительные условия")

    estimated\_learning\_time = models.PositiveIntegerField(*verbose\_name*="Примерное время изучения (часы)", *null*=True, *blank*=True)

    class Meta:

        verbose\_name = "Детали концепции"

        verbose\_name\_plural = "Детали концепций"

    def \_\_str\_\_(*self*):

*return* f"Детали {*self*.concept.title}"

*# Модель для тегов (Many-to-Many)*

class Tag(models.Model):

    name = models.CharField(*max\_length*=50, *unique*=True, *verbose\_name*="Тег")

    slug = models.SlugField(*max\_length*=50, *unique*=True, *db\_index*=True, *verbose\_name*="URL тега") *# Убрал default='default-slug'*

    class Meta:

        verbose\_name = "Тег"

        verbose\_name\_plural = "Теги"

    def \_\_str\_\_(*self*):

*return* *self*.name

    def save(*self*, \**args*, \*\**kwargs*):

*if* not *self*.slug:

*self*.slug = slugify(*self*.name)

        super().save(\**args*, \*\**kwargs*)

    def get\_absolute\_url(*self*):

*return* reverse('concepts\_by\_tag', *kwargs*={'tag\_slug': *self*.slug})

*# Добавляем ManyToManyField в ComputerScienceConcept после определения Tag*

ComputerScienceConcept.add\_to\_class('tags', models.ManyToManyField(

    Tag,

*related\_name*='concepts',

*verbose\_name*="Теги",

*blank*=True

))

Users/backends.py

*from* django.contrib.auth.backends *import* ModelBackend

*from* django.contrib.auth *import* get\_user\_model

User = get\_user\_model()

class EmailOrUsernameBackend(ModelBackend):

    """

    Аутентификация: сначала пробуем по email, если не нашли — по username.

    """

    def authenticate(*self*, *request*, *username*=None, *password*=None, \*\**kwargs*):

*if* *username* is None or *password* is None:

*return* None

        user = None

*# попробуем e-mail*

*if* '@' in *username*:

            user = User.objects.filter(*email\_\_iexact*=*username*, *is\_active*=True).first()

*# если не нашли по e-mail, попробуем username*

*if* user is None:

            user = User.objects.filter(*username\_\_iexact*=*username*, *is\_active*=True).first()

*if* user and user.check\_password(*password*):

*return* user

*return* None

Users/forms.py

*from* django *import* forms

*from* django.contrib.auth.forms *import* AuthenticationForm

class EmailOrUsernameAuthenticationForm(AuthenticationForm):

    """

    Форма логина: в одном поле пользователь вводит либо e-mail, либо username.

    """

    username = forms.CharField(

*label*='E-mail или логин',

*widget*=forms.TextInput(*attrs*={'autofocus': True})

    )

Users/urls.py

*from* django.urls *import* path

*from* . *import* views

app\_name = 'users'

urlpatterns = [

    path('login/',     views.login\_user,  *name*='login'),

    path('logout/',    views.logout\_user, *name*='logout'),

*# Сброс пароля:*

    path('password-reset/',

         views.CustomPasswordResetView.as\_view(),

*name*='password\_reset'),

    path('password-reset/done/',

         views.CustomPasswordResetDoneView.as\_view(),

*name*='password\_reset\_done'),

    path('reset/<uidb64>/<token>/',

         views.CustomPasswordResetConfirmView.as\_view(),

*name*='password\_reset\_confirm'),

    path('reset/done/',

         views.CustomPasswordResetCompleteView.as\_view(),

*name*='password\_reset\_complete'),

*# Профиль и смена пароля:*

    path('profile/',

         views.ProfileView.as\_view(),

*name*='profile'),

    path('password-change/',

         views.CustomPasswordChangeView.as\_view(),

*name*='password\_change'),

    path('password-change/done/',

         views.CustomPasswordChangeDoneView.as\_view(),

*name*='password\_change\_done'),

]

users/views.py

*from* django.contrib.auth.views *import* (

    PasswordResetView, PasswordResetDoneView,

    PasswordResetConfirmView, PasswordResetCompleteView

)

*from* django.urls *import* reverse\_lazy

*from* django.contrib.auth *import* login, logout

*from* django.shortcuts *import* render, redirect

*from* django.contrib.auth.mixins *import* LoginRequiredMixin

*from* django.views.generic *import* TemplateView

*from* django.contrib.auth.views *import* PasswordChangeView, PasswordChangeDoneView

*from* .forms *import* EmailOrUsernameAuthenticationForm

def login\_user(*request*):

    """

    Логин по e-mail или username с учётом next-параметра.

    """

    next\_url = *request*.GET.get('next') or *request*.POST.get('next') or ''

*if* *request*.method == 'POST':

        form = EmailOrUsernameAuthenticationForm(*request*, *data*=*request*.POST)

*if* form.is\_valid():

            login(*request*, form.get\_user())

*return* redirect(next\_url *or* 'cs:home')

*else*:

        form = EmailOrUsernameAuthenticationForm(*request*)

*return* render(*request*, 'users/login.html', {'form': form, 'next': next\_url})

def logout\_user(*request*):

    """

    Выход и редирект на главную.

    """

    logout(*request*)

*return* redirect('cs:home')

class ProfileView(LoginRequiredMixin, TemplateView):

    """

    Страница профиля текущего пользователя.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/profile.html'

    def get\_context\_data(*self*, \*\**kwargs*):

        ctx = super().get\_context\_data(\*\**kwargs*)

        ctx['user'] = *self*.request.user

*return* ctx

class CustomPasswordChangeView(LoginRequiredMixin, PasswordChangeView):

    """

    Форма смены пароля.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/password\_change\_form.html'

    success\_url = reverse\_lazy('users:password\_change\_done')

class CustomPasswordChangeDoneView(LoginRequiredMixin, PasswordChangeDoneView):

    """

    Страница подтверждения смены пароля.

    """

    login\_url = reverse\_lazy('users:login')

    template\_name = 'users/password\_change\_done.html'

class CustomPasswordResetView(PasswordResetView):

    """

    Шаг 1: форма запроса сброса пароля (ввод e-mail, отправка письма).

    """

    template\_name = 'users/password\_reset\_form.html'

    email\_template\_name = 'users/password\_reset\_email.html'

    subject\_template\_name = 'users/password\_reset\_subject.txt'

    success\_url = reverse\_lazy('users:password\_reset\_done')

class CustomPasswordResetDoneView(PasswordResetDoneView):

    """

    Шаг 2: страница, показываемая после отправки письма.

    """

    template\_name = 'users/password\_reset\_done.html'

class CustomPasswordResetConfirmView(PasswordResetConfirmView):

    """

    Шаг 3: форма установки нового пароля (переход по ссылке из письма).

    """

    template\_name = 'users/password\_reset\_confirm.html'

    success\_url = reverse\_lazy('users:password\_reset\_complete')

class CustomPasswordResetCompleteView(PasswordResetCompleteView):

    """

    Шаг 4: страница, показываемая после успешного сброса пароля.

    """

    template\_name = 'users/password\_reset\_complete.html'

Password\_reset\_form.html

{% extends 'cs/base.html' %}

{% block title %}Восстановление пароля{% endblock %}

{% block content %}

  <h2>Восстановление пароля</h2>

  <form method="post">{% csrf\_token %}

    {{ form.email.label\_tag }}<br>

    {{ form.email }}<br>

    {{ form.email.errors }}<br><br>

    <button type="submit">Отправить письмо</button>

  </form>

{% endblock %}

password\_reset\_done.html

{% extends 'cs/base.html' %}

{% block title %}Письмо отправлено{% endblock %}

{% block content %}

  <h2>Письмо с инструкциями отправлено!</h2>

  <p>Проверьте вашу почту.</p>

{% endblock %}

Password\_reset\_email.html

{% autoescape off %}

Чтобы сбросить пароль, перейдите по ссылке:

{{ protocol }}://{{ domain }}{% url 'users:password\_reset\_confirm' uidb64=uid token=token %}

{% endautoescape %}

password\_reset\_subject.txt

Восстановление пароля на сайте «Computer Science Concept»

password\_reset\_confirm.html

{% extends 'cs/base.html' %}

{% block title %}Установка нового пароля{% endblock %}

{% block content %}

  <h2>Установите новый пароль</h2>

  <form method="post">{% csrf\_token %}

    {{ form.new\_password1.label\_tag }}<br>

    {{ form.new\_password1 }}<br><br>

    {{ form.new\_password2.label\_tag }}<br>

    {{ form.new\_password2 }}<br><br>

    <button type="submit">Сохранить</button>

  </form>

{% endblock %}

password\_reset\_complete.html

{% extends 'cs/base.html' %}

{% block title %}Пароль сброшен{% endblock %}

{% block content %}

  <h2>Ваш пароль успешно изменён!</h2>

  <p><a href="{% url 'users:login' %}">Войти</a></p>

{% endblock %}

**Вывод**

В ходе выполнения двенадцатой лабораторной работы было создано новое приложение users, обеспечивающее полную систему регистрации и управления учётными записями. Мы настроили маршрутизацию так, чтобы все защищённые страницы автоматически перенаправляли неавторизованного пользователя на форму входа, а после успешного логина возвращали его к изначально запрошенному ресурсу. В рамках задачи добавили возможность входа не только по username, но и по E-mail, создав собственный аутентификационный бэкенд.

Для каждого пользователя реализовали страницу профиля с возможностью изменить пароль — используя готовые представления PasswordChangeView и PasswordChangeDoneView — и оформили шаблоны так, чтобы интерфейс был един в оформлении с основным сайтом. Далее мы внедрили процедуру восстановления пароля: настроили реальный SMTP-сервер Яндекс.Почты, добавили четыре представления (PasswordResetView, PasswordResetDoneView, PasswordResetConfirmView, PasswordResetCompleteView) и комплект шаблонов для ввода e-mail, отправки письма, перехода по ссылке и установки нового пароля. Письма с ссылкой для сброса корректно отправляются на настоящий почтовый ящик и приводят пользователя к форме изменения пароля.

Затем, чтобы гибко управлять правами доступа, мы через административный интерфейс создали две группы — Editors и Viewers — и с помощью Django shell добавили им соответствующие разрешения (change\_concept, view\_concept). Были созданы тестовые пользователи и показано, как назначать группы клавишей «Add». Ещё один пользователь получил право удаления концепций индивидуально, без группового ролевого набора.

Наконец, в модели concept добавили собственное разрешение can\_publish\_concept, прописанное в Meta.permissions, сгенерировали и применили миграцию. В админ-панели при редактировании пользователя в списке «User permissions» появилось наше новое пользовательское право, которое теперь можно назначать конкретным учёткам.

Таким образом, лабораторная работа полностью охватывает все семь пунктов задания: от создания приложения и маршрутов до глубокой настройки аутентификации, восстановления пароля и тонкой системы прав и групп в админ-панели.