**Министерство образования и науки**

Белгородский государственный технологический университет

им. В.Г. Шухова

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине «Базы данных»

тема: «Магазин»

Автор работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подкопаев А.В.

(подпись) ВТ-31

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. пр.

(подпись) Панченко М.В.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Белгород

2019 г.

**Оглавление**

[Введение 3](#_Toc30393710)

[Постановка задачи 3](#_Toc30393711)

[Описание предметной области 4](#_Toc30393712)

[Выбор средств проектирования и СУБД 4](#_Toc30393713)

[Проектирование БД 4](#_Toc30393714)

[Классификация связей 6](#_Toc30393715)

[Реляционная модель БД 7](#_Toc30393716)

[Ключевые поля 8](#_Toc30393717)

[Физическое проектирование 8](#_Toc30393718)

[Реализация приложения 9](#_Toc30393719)

[Заключение 14](#_Toc30393720)

[Список используемых источников 14](#_Toc30393721)

[Приложение 1. Python файлы 15](#_Toc30393722)

[Приложение 2. HTML/CSS файлы 15](#_Toc30393723)

# Введение

В настоящее время жизнь человека зависит от различного рода информации, для управления которой требуются создания огромного количества баз и банков данных различного назначения.

Базы данных — это совокупность структур, предназначенных для хранения больших объемов информации и программных модулей, осуществляющих управление данными, их выборку, сортировку и другие подобные действия.

Понятие базы данных можно применять к любой связанной между по определенному признаку информации, хранимой и ограниченной особым образом. По сути база данных - некоторое подобие картотеки, электронного хранилища данных, которые хранятся в компьютере в виде одного или нескольких файлов.

Автоматизированные базы данных стали неотъемлемой частью практически всех компьютерных систем - от отрасли до отдельного предприятия. За последние несколько лет вырос уровень потребительских качеств систем управления базами данных (СУБД): разнообразие поддерживаемых функций, удобный для пользователя интерфейс, сопряжение с программными продуктами, в частности с другими СУБД, возможности для работы в сети и т.д. СУБД позволяет сводить воедино информацию из самых разных источников (электронные таблицы, другие базы данных) и помогает быстро найти необходимую информацию, выводить её с помощью отчетов, графиков или таблиц.

# Постановка задачи

Разработать автоматизированную информационную систему для работы с базой данных в рамках предметной области “Магазин”, разработать приложение с разграничением прав доступа для работы с БД, разработать средство создания резервных копий базы данных и восстановления из них.

# Описание предметной области

База данных магазина хранит в себе информацию о товарах, имеющихся на складах всей сети магазинов. Каждый товар имеет свое название, дату доставки и цену. С помощью базы данных можно отслеживать наличие товаров на складе, добавлять новые товары.

# Выбор средств проектирования и СУБД

Приложение будет представлять из себя программу на языке Python. В качестве СУБД была выбрана SQLite. Проектирование будет осуществляться средствами фреймворка Django – ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

# Проектирование БД

Проектирование БД осуществляется на основе трехуровневой архитектуры. Наглядно это представлено на рис 1.1

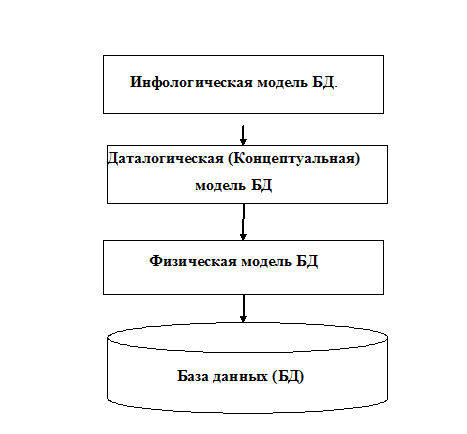


Рис. 1.1. Трехуровневая система проектирования БД

Инфологическая модель БД - модель, которая описывает данные предметной области с использованием естественного языка. Исходными данными могут быть стандартные справки, бланки или документы.

Даталогическая (Концептуальная) модель БД - логическая организация данных и их взаимосвязь. Структурирование данных выполняется в соответствии с выбранной СУБД, которая строится на основе выбранной модели представления данных: иерархической, сетевой или реляционной. На данном этапе проектировщик создает структуру данных и организует связь между объектами.

Физическая модель БД - структура БД, реализованная в среде выбранной СУБД.

Также при создании БД важно учитывать следующие параметры:

* Целостность БД - правильность данных в любой момент времени
* Избыточность БД - дублирование данных в нескольких таблицах (избыточность влечет за собой противоречивость данных, ее стараются исключить введением кодовых полей).

Цель проектирования БД - сокращение избыточных данных. Эта цель достигается с помощью теории нормализации баз данных.

Нормализация - разбиение таблиц на два или более, обладающими лучшими свойствами при внесении или изменении данных. В результате получается БД, в которой каждый факт появляется лишь в одном месте, что исключает избыточность информации.

**Нормализация**

Разбиение таблиц на два или более, обладающими лучшими свойствами при внесении или изменении данных называют нормализацией. В результате получается БД, в которой каждый факт появляется лишь в одном месте, что исключает избыточность информации.

Кодом выведено три нормальные формы и предложен механизм, позволяющий любое отношение преобразовать к третей нормальной форме. Приведем наши отношения к третей нормальной форме.

Первая НФ: Отношение называется нормализованным или приведенным к первой нормальной форме тогда и только тогда, когда все его атрибуты простые (неделимые). Таблица находится в первой нормальной форме тогда и только тогда, когда ни одна из ее строк не содержит в любом ее поле более одного значения, и не одно из ее ключевых полей не пусто.

Вторая НФ: Таблица находится во второй нормальной форме, если она удовлетворяет определению первой нормальной формы и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом.

Третья НФ: Таблица находится в третей нормальной форме, если она удовлетворяет определению второй нормальной формы и ни одно из ее не ключевых полей не зависит функционально от любого другого не ключевого поля. Отношения, представленные в данной БД приведены к третей нормальной форме.

## **Классификация связей**

При проектировании БД информацию обычно размещают в нескольких таблицах. Таблицы при этом связывают с семантикой информации. В реляционной СУБД для указания связей в таблице производят операции их связывания. Рассмотрим наиболее часто встречаемые бинарные связи:

Связи 1:1 образуется в случае, когда ключевые поля записи основной таблицы являются ключевыми полями записи дополнительной таблицы.

Связь 1:М (один ко многим) может быть в случае, когда одной записи основной таблицы соответствует несколько записей дополнительной таблицы.

Связь М:1 (многие к одному) может быть тогда, когда нескольким записям основной таблицы ставится в соответствии одна запись дополнительной.

Связь М:М (многие ко многим) возникает в том случае, когда нескольким записям основной таблицы соответствует несколько записей дополнительной. В реляционной БД связь М:М реализуется через дополнительные таблицы.

Диаграмма “сущность-связь” для данной предметной области будет иметь следующий вид:

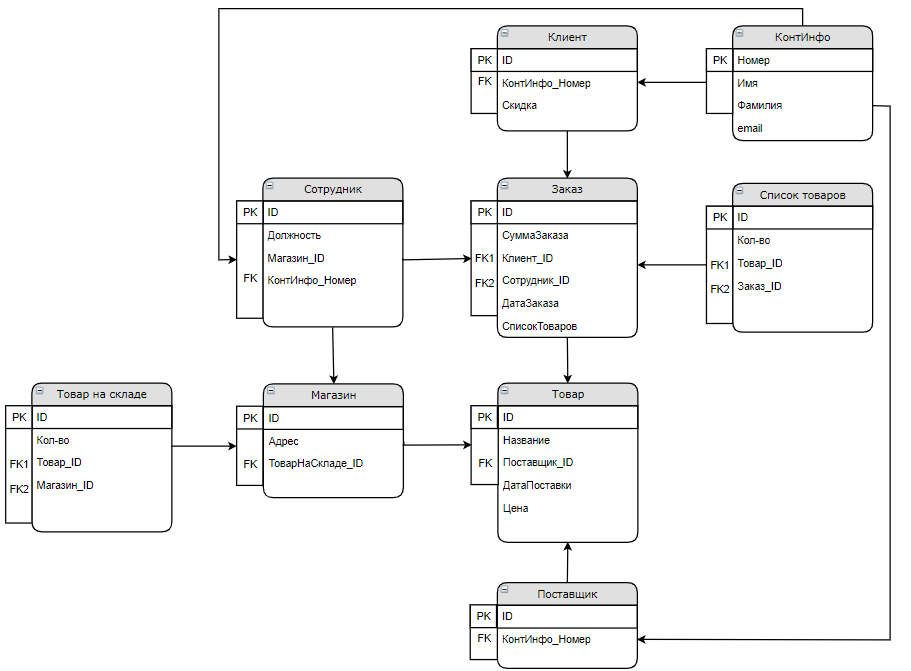
****

Рис. 2.1 – диаграмма “сущность связь” для предметной области “магазин”

# Реляционная модель БД

Реляционная модель баз данных была предложена сотрудником фирмы IBM Э. Кодом в начале 70-х годов. Будучи математиком, он предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность и Декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известных в математике как отношения.

Одна из главных идей заключается в том, что связи между данными должны устанавливаться в соответствии с их внутренними логическими взаимоотношениями. В реляционной модели одной командой могут обрабатываться целые файлы.

Реляционная БД представляет собой информацию об объекте, представленную в виде двумерного массива - таблицы объеденных определёнными связями.

## **Ключевые поля**

Атрибут значением, которого идентифицируется кортежи (строки таблицы) называется ключом. Отношение может содержать и несколько ключей, один из которых объявляется первичным. Все прочие ключи отношений являются возможными ключами.

Если в отношении кортеж идентифицируется соединением значений нескольких атрибутов, то такой ключ называется составным.

Атрибут представляющие собой копии ключей других отношений называется внешним ключом. Реляционная модель накладывает на внешние ключи ограничения для обеспечения целостности данных. Это означает, что к каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

В разрабатываемой БД ключами отношений будут являться целочисленные автоинкрементируемые идентификаторы.

# Физическое проектирование

Поскольку структура базы данных создаётся средствами ORM, реальная структура может отличаться от предполагаемой. В данном случае отсутствуют явные связи между отношениями, то есть вторичные ключи, и первичные ключи в вспомогательных таблицах.

Полученная структура базы данных с использованием ORM:

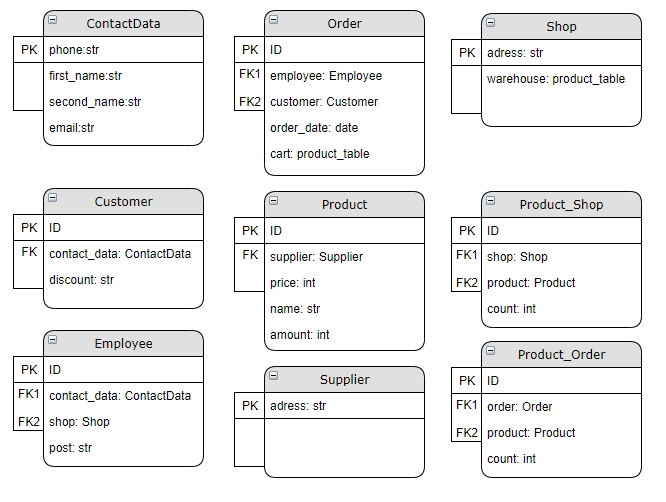


Рис. 3.1 – структура базы данных для предметной области “магазин”

# Реализация приложения

Разграничение прав доступа будет реализовываться на уровне БД. Для того, чтобы получить доступ к БД, пользователю нужно войти в свой аккаунт. Если его нет – его необходимо зарегистрировать, а затем получить доступ у администратора.

Администратор также имеет доступ ко всей БД, изменять поля таблиц, удалять и добавлять записи, редактировать права других пользователей.

После авторизации пользователь имеет доступ к функционалу базы данных (на данный момент – просмотр списка товаров, информацию о товарах и редактирование товаров, в дальнейшем планируется расширение функционала до функционала интернет-магазина: оформление заказов, личный кабинет пользователя, корзина покупок и т.д.)

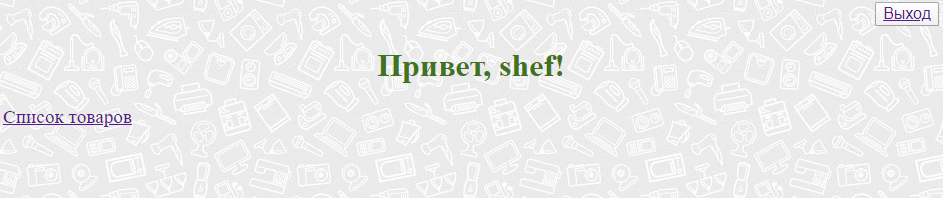


Рис. 4.1 – главная страница

Перейдя к списку товаров, пользователь увидит все товары, имеющиеся на данный момент в базе данных, и краткую информацию о них.



Рис. 4.2 – список товаров

Если требуется более подробная информация о товаре – пользователь может нажать на товар и поглядеть.

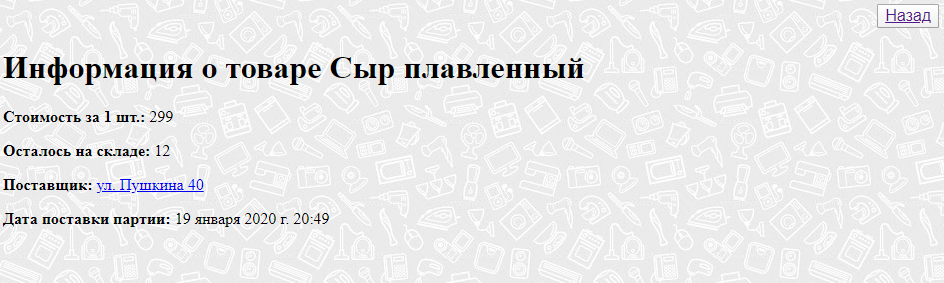


Рис. 4.3 – информация о товаре

Если требуется добавить товар – пользователь нажимает на соответствующую кнопку, и заполняет форму.



Рис. 4.4 – добавление товара

Стоит рассказать поподробнее о реализации авторизации и регистрации, а также защиты от “несанкционированного” доступа. Взглянем на это глазами пользователя, впервые зашедшего на сайт.

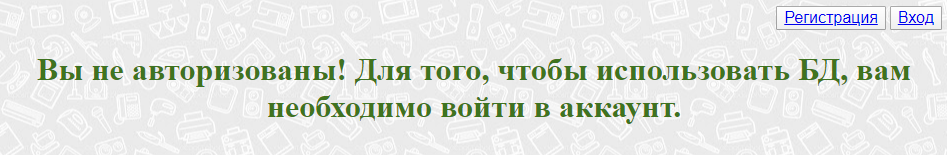


Рис. 4.5 – главная страница для пользователя, не совершившего авторизацию

Перед тем, как показать информацию, выполняется проверка пользователя на авторизацию. Если пользователь не авторизирован, он видит данную надпись, а также кнопку регистрации и входа, к ним мы вернемся позднее. Далее выполняется проверка авторизированного пользователя на доступ к определенному функционалу, например списку товаров. Если пользователь авторизирован, но не имеет прав на просмотр данных (т.е. не имеет группу “Работник”, то он увидит следующее:



Рис. 4.6 – главная страница для пользователя, не имеющего доступа к функционалу работника магазина.

Права “Работник магазина” выдаются администратором в админ-панели Django. (рис. 4.8.)

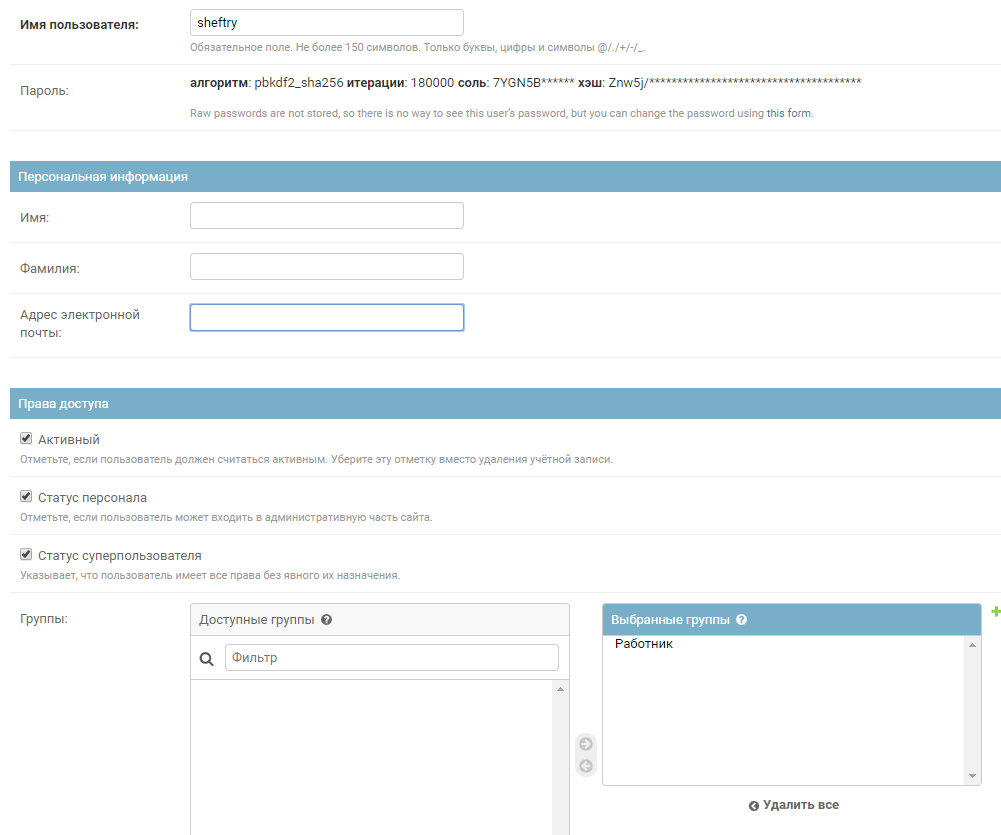


Рис. 4.8 – добавление пользователя к группе “Работник”.

Также администраторы могут редактировать БД магазина через админ-панель Django, если потребуется.

Возвращаясь к панели аутентификации и регистрации, то здесь используется версия аутентификации, встроенной в Django.

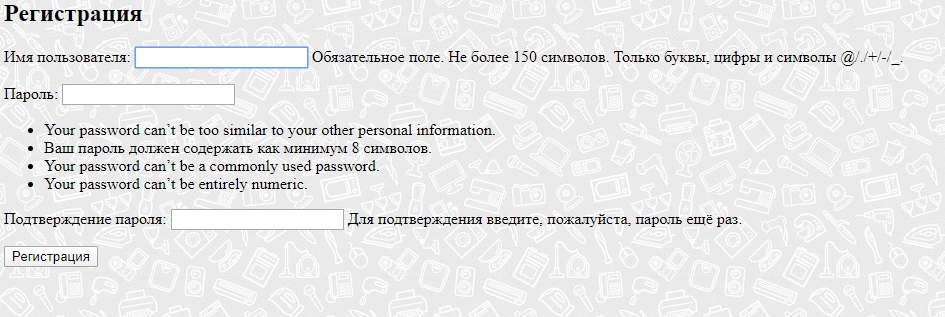


Рис. 4.9 – форма регистрации

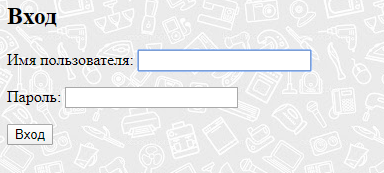


Рис. 4.10 – форма аутентификации

# Заключение

В результате выполнения курсовой работы была создана база данных магазина и разработан сайт для пользователей магазина, который не требует от них знания структуры БД. Также сайт имеет разграничение доступа, что позволяет скрыть ненужную информацию для обычных пользователей, а для работников магазина наоборот, получать нужную информацию.

В связи с тем, что проект является учебным, а не профессиональным, большая часть функционала в нём отсутствует, до конца не реализована большая часть функционала интернет-магазина, например, оформление заказов. Текущий функционал базы данных можно легко расширить при надобности.

# Список используемых источников

1. Голицына О. Л., Партыка Т. Л., Попов И. И. Основы проектирования баз данных; Форум - Москва, 2012. - 416 c.
2. Малыхина М.П. Базы данных: основы, проектирование, использование учеб. пособие / М.П.Малыхина. – СПб. БХВ‑Петербург, 2004. – 512с.
3. Теория и практика построения баз данных. 8-е изд. / Д.Крёнке. – СПб.: Питер, 2003. – 800 с.
4. Документация Django [Электронный ресурс]

URL: <https://docs.djangoproject.com/en/3.0/>

1. Документация HTML [Электронный ресурс]

URL: <https://devdocs.io/html/>

**Приложение 1. Python файлы**

Файл manage.py

#!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'kursovaya.settings')

try:

from django.core.management import execute\_from\_command\_line

except ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you "

"forget to activate a virtual environment?"

) from exc

execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Файл settings.py

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'vl%=n#v\_93kgqs^2oh6\_9s+-6v178p2$w2#gsx1y=#a1e\_wpye'

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

DEBUG = True

ALLOWED\_HOSTS = ['192.168.1.2', '77.247.239.83', '127.0.0.1', 'localhost']

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'accounts.apps.AccountsConfig',

'kursovaya',

]

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 = 'kursovaya.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

'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 = 'kursovaya.wsgi.application'

# Database

# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

# Password validation

# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'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/3.0/topics/i18n/

LANGUAGE\_CODE = 'ru'

TIME\_ZONE = 'Europe/Moscow'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC\_URL = '/static/'

LOGIN\_REDIRECT\_URL = 'home'

LOGOUT\_REDIRECT\_URL = 'home'

Файл models.py

from django.db import models

class ContactData(models.Model):

phone = models.CharField(max\_length= 255,blank = False)

first\_name = models.CharField(max\_length= 255,blank = False)

second\_name = models.CharField(max\_length= 255,blank = False)

email = models.CharField(max\_length= 255)

def \_\_str\_\_(self):

return self.phone

class Customer(models.Model):

contact\_data = models.ForeignKey(ContactData,models.CASCADE)

discount = models.CharField(max\_length= 255)

def \_\_str\_\_(self):

return self.contact\_data.first\_name + ' ' + self.contact\_data.second\_name

class Employee(models.Model):

contact\_data = models.ForeignKey(ContactData,models.CASCADE)

shop = models.ForeignKey('Shop',models.CASCADE)

post = models.CharField(max\_length= 255)

def \_\_str\_\_(self):

return self.contact\_data.first\_name + ' ' + self.contact\_data.second\_name

class Order(models.Model):

employee = models.ForeignKey(Employee,models.CASCADE)

customer = models.ForeignKey(Customer,models.CASCADE)

order\_date = models.DateTimeField()

cart = models.ManyToManyField(

'Product',

through='Product\_Order',

through\_fields=('order','product')

)

def \_\_str\_\_(self):

return str(self.id)

class Product(models.Model):

supplier = models.ForeignKey('Supplier',models.CASCADE)

price = models.PositiveIntegerField(default=0)

name = models.CharField(max\_length= 500)

supply\_date = models.DateTimeField()

amount = models.PositiveIntegerField(default=0)

def \_\_str\_\_(self):

return self.name

class Shop(models.Model):

adress = models.CharField(max\_length= 500,blank = False)

warehouse = models.ManyToManyField(

Product,

through='Product\_Shop',

through\_fields=('shop','product')

)

def \_\_str\_\_(self):

return self.adress

class Supplier(models.Model):

adress = models.CharField(max\_length= 500,blank = False)

def \_\_str\_\_(self):

return self.adress

class Product\_Shop(models.Model):

shop = models.ForeignKey(Shop,on\_delete = models.CASCADE)

product = models.ForeignKey(Product,on\_delete = models.CASCADE)

count = models.PositiveIntegerField(default=0)

def \_\_str\_\_(self):

return self.shop.adress

class Product\_Order(models.Model):

order = models.ForeignKey(Order,on\_delete = models.CASCADE)

product = models.ForeignKey(Product,on\_delete = models.CASCADE)

count = models.PositiveIntegerField(default=0)

def \_\_str\_\_(self):

return self.product.name + ' ' + str(self.order.id)

Файл forms.py

from django import forms

from .models import Product,Supplier

class ProductForm(forms.ModelForm):

class Meta:

model = Product

fields = ('name','price','amount','supplier','supply\_date')

Файл forms.py

from django.contrib import admin

from kursovaya.models import ContactData,Customer,Employee,Order,Product,Shop,Supplier,Product\_Order,Product\_Shop

admin.site.register(ContactData)

admin.site.register(Customer)

admin.site.register(Employee)

admin.site.register(Order)

admin.site.register(Product)

admin.site.register(Shop)

admin.site.register(Supplier)

admin.site.register(Product\_Order)

admin.site.register(Product\_Shop)

Файл urls.py

from django.contrib import admin

from django.urls import path, include

from django.views.generic.base import TemplateView

from . import views

urlpatterns = [

path('admin/', admin.site.urls),

path('accounts/', include('accounts.urls')),

path('accounts/', include('django.contrib.auth.urls')),

path('', TemplateView.as\_view(template\_name='home.html'), name='home'),

path('products/<int:product\_id>/',views.product\_details,name='products'),

path('products',views.products,name='products'),

path('product\_add',views.product\_add,name = 'product\_add')

]

Файл views.py

from django.http import HttpResponse

from django.template import loader

from django.shortcuts import render,redirect

from .forms import ProductForm

from .models import ContactData,Customer,Employee,Order,Shop,Product\_Order,Product\_Shop,Product,Supplier

import datetime

def index(request):

template = loader.get\_template('index.html')

context = {}

return HttpResponse(template.render(context,request))

def product\_details(request,product\_id):

product = Product.objects.raw(

'''SELECT \* FROM kursovaya\_product

WHERE

kursovaya\_product.id = %d'''%product\_id

)

template = loader.get\_template('product\_details.html')

context = {

'product': product[0]

}

return HttpResponse(template.render(context,request))

def products(request):

products = Product.objects.raw('SELECT \* FROM kursovaya\_product')

template = loader.get\_template('products.html')

context = {

'products': products

}

return HttpResponse(template.render(context,request))

def product\_add(request):

if request.method == "POST":

form = ProductForm(request.POST)

if form.is\_valid():

product = form.save(commit=False)

product.save()

else:

form = ProductForm()

return render(request, 'product\_add.html', {'form': form})

Файл auth\_extras.py

from django import template

from django.contrib.auth.models import Group

register = template.Library()

@register.filter(name='has\_group')

def has\_group(user, group\_name):

group = Group.objects.get(name=group\_name)

return True if group in user.groups.all() else False

Файл accounts/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('signup/', views.SignUp.as\_view(), name='signup'),

]

Файл accounts/views.py

from django.contrib.auth.forms import UserCreationForm

from django.urls import reverse\_lazy

from django.views import generic

class SignUp(generic.CreateView):

form\_class = UserCreationForm

success\_url = reverse\_lazy('login')

template\_name = 'signup.html'

**Приложение 2. HTML/CSS файлы**

Файл base.html

<!-- templates/base.html -->

<html>

<head>

<meta charset="utf-8">

<title>{% block title %}{% endblock %}</title>

</head>

<body>

<main>

<style>

body {background: url(https://html5book.ru/wpcontent/uploads/2015/06/background28.jpg) 50%; }

</style>

{% block content %}

{% endblock %}

</main>

</body>

</html>

Файл home.html

<!-- templates/home.html-->

{% extends 'base.html' %}

{% load auth\_extras %}

{% block title %}Домашняя страница{% endblock %}

{% block content %}

{% if user.is\_authenticated %}

<div style="text-align:right; width:100%; padding:0;">

<button><big><a href="{% url 'logout' %}">Выход</a></big></button>

</div>

<h1 style="text-align: center; color: #3f7320;">Привет, {{ user.username }}!</h1>

{% if request.user|has\_group:"Работник"%}

<big><a href='/products'>Список товаров<a></big>

{% endif %}

{% else %}

<div style="text-align:right; width:100%; padding:0;">

<button><big><a href="{% url 'signup' %}">Регистрация</a></big></button>

<button><big><a href="{% url 'login' %}">Вход</a></big></button>

</div>

<h1 style="text-align: center; color: #3f7320;">Вы не авторизованы! Для того, чтобы использовать БД, вам необходимо войти в аккаунт.</h1>

{% endif %}

{% endblock %}

Файл product\_add.html

{% extends 'base.html' %}

{% load static %}

{% block content %}

{% if user.is\_authenticated %}

<html>

<head>

<p style="text-align: right;">

<button><big><a href="{% url 'products' %}">Назад</a></big></button>

</p>

<title>Добавить товар</title>

<link rel="stylesheet" href="{% static 'css/info.css' %}">

<link rel="stylesheet" href="{% static 'css/form.css' %}">

</head>

<body>

<header class="info\_container">

<header class="info\_header">

<h1>Добавить товар</h1>

</header>

<div class="info\_body">

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

{{form.as\_p}}

<button style="padding-bottom: 1px ;" type="submit">Сохранить</button>

</form>

</div>

</div>

</body>

{% else %}

<div style="text-align:right; width:100%; padding:0;">

<button><big><a href="{% url 'login' %}">Вход</a></big></button>

</div>

<h1 style="text-align: center; color: #3f7320;">Вы не авторизованы! Для того, чтобы использовать БД, вам необходимо войти в аккаунт.</h1>

{% endif %}

{% endblock %}

Файл product\_details.html

{% extends 'base.html' %}

{% load static %}

{% block content %}

{% if user.is\_authenticated %}

<html>

<head>

<title>Информация о товаре {{product.name}}</title>

<link rel="stylesheet" href="{% static 'css/info.css' %}">

</head>

<p style="text-align: right;">

<button><big><a href="{% url 'products' %}">Назад</a></big></button>

</p>

<body>

<header class="info\_container">

<header class="info\_header">

<h1>Информация о товаре {{product.name}}</h1>

</header>

<div class="info\_body">

<p>

<b>Стоимость за 1 шт.:</b> {{product.price}}

</p>

<p>

<b>Осталось на складе:</b> {{product.amount}}

</p>

<p>

<b >Поставщик:</b> <a href="/supplier/{{product.supplier.id}}">{{product.supplier.adress}}</a>

</p>

<p>

<b>Дата поставки партии:</b> {{product.supply\_date}}

</p>

</div>

</div>

</body>

{% else %}

<div style="text-align:right; width:100%; padding:0;">

<button><big><a href="{% url 'login' %}">Вход</a></big></button>

</div>

<h1 style="text-align: center; color: #3f7320;">Вы не авторизованы! Для того, чтобы использовать БД, вам необходимо войти в аккаунт.</h1>

{% endif %}

{% endblock %}

Файл products.html

{% extends 'base.html' %}

{% load auth\_extras %}

{% load static %}

{% block content %}

{% if user.is\_authenticated %}

{% if request.user|has\_group:"Работник"%}

<html>

<head>

<title>Список товаров </title>

<link rel="stylesheet" href="{% static 'css/info.css' %}">

</head>

<body>

<p style="text-align: right;">

<button><big><a href="{% url 'home' %}">Главная страница</a></big></button>

</p>

<header class="table\_container">

<header class="table\_header">

<h1>Список товаров</h1>

</header>

<p style="text-align: right;">

<a href="/product\_add">Добавить продукт</a>

</p>

<div class="table\_body">

<table style="width:100%">

<tr>

<th>№</th>

<th>Название товара</th>

<th>Количество на складе</th>

<th>Цена</th>

</tr>

{% for r in products %}

<tr>

<th><a href = '/products/{{r.id}}'>{{r.id}}</header></a></th>

<th><a href = '/products/{{r.id}}'>{{r.name}}</a></th>

<th><a href = '/products/{{r.id}}'>{{r.amount}}</a></th>

<th><a href = '/products/{{r.id}}'>{{r.price}}</a></th>

</tr>

{% endfor %}

</table>

</div>

</div>

</body>

{% else %}

<p style="text-align: right;">

<button><big><a href="{% url 'home' %}">Главная страница</a></big></button>

</p>

<h1 style="text-align: center; color: #3f7320;">У вас нет доступа к этой информации! Если у вас должен быть доступ - обратитесь к системному администратору.</h1>

<h1 style="text-align: center; color: #3f7320;"> Почта: konstashapovalov@gmail.com</h1>

{% endif %}

{% else %}

<div style="text-align:right; width:100%; padding:0;">

<button><big><a href="{% url 'login' %}">Вход</a></big></button>

</div>

<h1 style="text-align: center; color: #3f7320;">Вы не авторизованы! Для того, чтобы использовать БД, вам необходимо войти в аккаунт.</h1>

{% endif %}

{% endblock %}

Файл signup.html

<!-- templates/signup.html -->

{% extends 'base.html' %}

{% block title %}Регистрация{% endblock %}

{% block content %}

<h2>Регистрация</h2>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

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

</form>

{% endblock %}

Файл login.html

<!-- templates/registration/login.html -->

{% extends 'base.html' %}

{% block title %}Вход{% endblock %}

{% block content %}

<h2>Вход</h2>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Вход</button>

</form>

{% endblock %}

Файл form.css

body{

background-color: rgb(241, 241, 241);

text-align: center;

}

a:link {

font-family: 'Roboto', sans-serif;

background-color: rgb(221, 214, 241);

color: rgb(0, 0, 0);

}

button{

margin-bottom: 1px !important;

margin-top: 1px !important;

margin-left: 1px !important;

margin-right: 1px !important;

}

Файл index.css

body{

background-color: rgb(241, 241, 241);

text-align: center;

}

a:link {

font-family: 'Roboto', sans-serif;

background-color: rgb(221, 214, 241);

color: rgb(0, 0, 0);

}

Файл info.css

body{

background-color: rgb(241, 241, 241);

}

:not(.info\_container) {

text-align: center;

margin-left: 20px !important;

margin-right: 20px !important;

font-family: 'Roboto', sans-serif;

}

:not(.info\_header) {

text-align: center;

font-family: 'Roboto', sans-serif;

}

:not(.info\_body) {

}

a:link {

background-color: rgb(221, 214, 241);

color: rgb(0, 0, 0);

}

Файл table.css

body{

background-color: rgb(241, 241, 241);

}

:not(.table\_container) {

text-align: center;

margin-left: 20px !important;

margin-right: 20px !important;

margin-top: 20px !important;

margin-bottom: 20px !important;

font-family: 'Roboto', sans-serif;

}

:not(.table\_header) {

text-align: center;

font-family: 'Roboto', sans-serif;

margin-left: 0px !important;

padding-bottom: 100px;

}

:not(.table\_body) {

padding-bottom: 10px;

}

a:link {

background-color: rgb(221, 214, 241);

color: rgb(0, 0, 0);

}

:not(.table){

margin-right: 20px !important;

}