Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторной работе №7**

Дисциплина: «Разработка профессиональных приложений»

Вариант №8

Выполнил:

студент группы ИВТАСбд-21

Зюзин Г.А.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Задание по варианту.**

Для предложенного варианта заданий необходимо реализовать веб-сайт в среде Django, удовлетворяющий следующим требованиям:

1. Количество таблиц в БД - не менее 5. Таблицы должны быть спроектированы в соответствии с тремя нормальными формами (https://habr.com/ru/post/254773/).

2. Должны быть реализованы 3 типа связей: one-to-one, one-to-many, man-to-many.

3. Таблицы создаются с использованием моделей в Django и миграций.

4. Как минимум, для 1 сущности должны быть реализованы CRUD операции (Create, read, update, delete) + представление списка сущностей.

5. Остальные таблицы должны быть доступны для редактирования в административной части сайта.

6. Необходимые пять таблиц должны быть разработаны для моделирования предметной области, заданной вариантом. Необходимо придумать эти таблицы, не забывая про здравый смысл.

Вариант 8. История посещений поликлиники: №, ФИО пациента, ФИО врача, причина обращения, длительность

**Описание реализации**.

Модели таблицы базы данных были созданы в файле models.py, в нём же были прописаны связи таблиц, которые были спроектированы ранее, после чего были произведены миграции.

|  |
| --- |
| from django.db import models  class Patient(models.Model):  full\_name = models.CharField(max\_length=255)  phone\_number = models.CharField(max\_length=20)  preferred\_doctors = models.ManyToManyField('Doctor', through='PreferredDoctor', related\_name='preferred\_patients')  class Doctor(models.Model):  full\_name = models.CharField(max\_length=255)  specialty = models.CharField(max\_length=255)  room\_number = models.CharField(max\_length=10)  class Visit(models.Model):  patient = models.ForeignKey(Patient, on\_delete=models.CASCADE)  doctor = models.ForeignKey(Doctor, on\_delete=models.CASCADE)  reason = models.CharField(max\_length=255)  duration = models.IntegerField()  class MedicalRecord(models.Model):  visit = models.OneToOneField(Visit, on\_delete=models.CASCADE)  text = models.TextField()  class MedicalCard(models.Model):  patient = models.OneToOneField(Patient, on\_delete=models.CASCADE)  text = models.TextField()  class PreferredDoctor(models.Model):  patient = models.ForeignKey(Patient, on\_delete=models.CASCADE)  doctor = models.ForeignKey(Doctor, on\_delete=models.CASCADE) |

CRUD операции реализованы для каждой сущности БД. В последующей работе будет введено ограничения для обычного пользователя.

Итоговая модель базы данных, включающая все таблицы и связи, будет выглядеть следующим образом:

Таблица "Пациенты":

* id (первичный ключ)
* ФИО
* Номер телефона

Таблица "Врачи":

* id (первичный ключ)
* ФИО
* Специальность
* Номер кабинета

Таблица "Посещения":

* id (первичный ключ)
* пациент\_id (внешний ключ, связь с таблицей "Пациенты")
* врач\_id (внешний ключ, связь с таблицей "Врачи")
* причина обращения
* длительность (секунды)

Таблица "Медицинские записи":

* id (первичный ключ)
* посещение\_id (внешний ключ, связь с таблицей "Посещения")
* текст записи

Таблица "Медицинские карты":

* id (первичный ключ)
* пациент\_id (внешний ключ, связь с таблицей "Пациенты")
* текст медицинской карты

Таблица "Предпочитаемые врачи":

* id (первичный ключ)
* пациент\_id (внешний ключ, связь с таблицей "Пациенты")
* врач\_id (внешний ключ, связь с таблицей "Врачи")

Эта модель базы данных позволяет хранить информацию о пациентах, врачах, посещениях, медицинских записях и медицинских картах. Взаимосвязи между таблицами устанавливаются с помощью внешних ключей, чтобы связать данные в разных таблицах.

В данной модели базы данных присутствуют следующие типы связей:

1. One-to-One (один к одному). Связь между таблицей "Пациенты" и таблицей "Медицинские карты". Каждая запись в таблице "Пациенты" может иметь только одну связанную запись в таблице "Медицинские карты", и наоборот.
2. One-to-Many (один ко многим).
   1. Связь между таблицей "Пациенты" и таблицей "Посещения". Один пациент может иметь несколько связанных записей в таблице "Посещения", но каждая запись в таблице "Посещения" может быть связана только с одним пациентом.
   2. Связь между таблицей "Врачи" и таблицей "Посещения". Один врач может иметь несколько связанных записей в таблице "Посещения", но каждая запись в таблице "Посещения" может быть связана только с одним врачом.
3. Many-to-Many (многие ко многим). Связь между таблицей "Пациенты" и таблицей "Врачи" через таблицу "Предпочитаемые врачи". Один пациент может предпочитать несколько врачей, и каждый врач может быть предпочитаемым для нескольких пациентов. Эта связь реализуется с использованием промежуточной таблицы "Предпочитаемые врачи", которая содержит внешние ключи для связи пациентов и врачей.

Таким образом, модель базы данных включает различные типы связей (one-to-one, one-to-many, many-to-many), которые позволяют устанавливать связи между записями в разных таблицах и представлять соответствующие отношения между данными.

Чтобы реализовать веб-сайт на основе предложенной модели данных в Django, вам следует следовать следующим шагам:

1. Создайте новое Django приложение, используя команду python manage.py startapp сlinic. Здесь clinic - это имя приложения.
2. Внесите необходимые изменения в файл настроек Django settings.py, чтобы добавить ваше новое приложение clinic в список установленных приложений.
3. В созданном приложении clinic создайте файлы models.py и admin.py.
4. Вставьте предложенные модели в файл models.py вашего приложения clinic (модели описаны выше).
5. Зарегистрируйте модели в файле admin.py, чтобы они были доступны в административной части сайта:

|  |
| --- |
| from django.contrib import admin  from .models import Patient, Doctor, Visit, MedicalRecord, MedicalCard, PreferredDoctor  admin.site.register(Patient)  admin.site.register(Doctor)  admin.site.register(Visit)  admin.site.register(MedicalRecord)  admin.site.register(MedicalCard)  admin.site.register(PreferredDoctor) |

1. Создайте миграции для вашей модели данных, выполнив команду python manage.py makemigrations в корневой папке проекта.
2. Примените миграции, выполнив команду python manage.py migrate.
3. Теперь вы можете создать представления (views) и шаблоны (templates) для реализации CRUD операций и отображения списка сущностей. Создайте соответствующие файлы внутри вашего приложения myapp и определите необходимую логику.
4. Добавьте URL-маршруты в файле urls.py вашего проекта, чтобы связать URL-адреса с вашими представлениями.
5. Запустите сервер разработки Django с помощью команды python manage.py runserver и проверьте функциональность вашего веб-сайта.

На примере view для отображения врачей покажу, как создавалось отображение.

Чтобы создать представление (view) для отображения списка врачей, вы можете использовать классы представлений Django, такие как ListView или TemplateView. В данном случае предлагается использовать ListView, который позволит вам легко отображать список объектов модели.

1. В файле views.py вашего приложения, импортируйте необходимые классы и модели:

|  |
| --- |
| from django.views.generic import ListView  from .models import Doctor  Создайте класс представления для отображения списка врачей:  python  Copy code  class DoctorListView(ListView):  model = Doctor  template\_name = 'doctor\_list.html'  context\_object\_name = 'doctors' |

1. Создайте соответствующий шаблон doctor\_list.html в папке templates вашего приложения myapp. Ниже приведен пример базового шаблона, отображающего список врачей:

|  |
| --- |
| <!-- doctor\_list.html -->  <!DOCTYPE html>  <html>  <head>  <title>Список врачей</title>  </head>  <body>  <h1>Список врачей</h1>  <ul>  {% for doctor in doctors %}  <li>{{ doctor.full\_name }}</li>  {% endfor %}  </ul>  </body>  </html> |

1. Обновите файл urls.py вашего приложения myapp, чтобы добавить URL-маршрут для представления DoctorListView:

|  |
| --- |
| from django.urls import path  from .views import DoctorListView  urlpatterns = [  path('doctors/', DoctorListView.as\_view(), name='doctor\_list'),  ] |

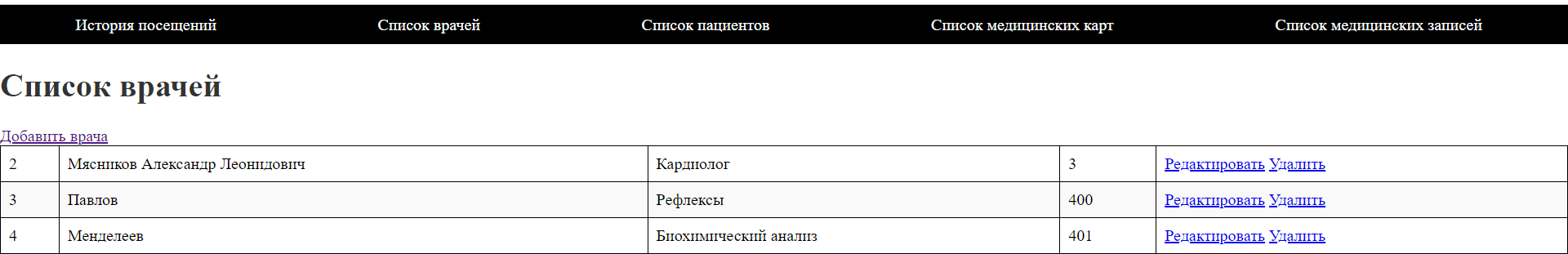


Рис. 1. Отображение списка врачей.

На Рис.1 помимо отображения списка есть ещё возможность добавления, редактирования и удаления врача. Разберём их ниже.

|  |
| --- |
| class DoctorCreateView(CreateView):  model = Doctor  template\_name = 'doctor\_create.html'  fields = ['full\_name', 'specialty', 'room\_number']  success\_url = reverse\_lazy('doctor\_list')  class DoctorUpdateView(UpdateView):  model = Doctor  template\_name = 'doctor\_update.html'  fields = ['full\_name', 'specialty', 'room\_number']  success\_url = reverse\_lazy('doctor\_list')  class DoctorDeleteView(DeleteView):  model = Doctor  template\_name = 'doctor\_delete.html'  success\_url = reverse\_lazy('doctor\_list') |

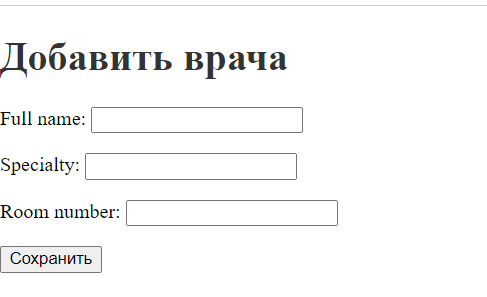
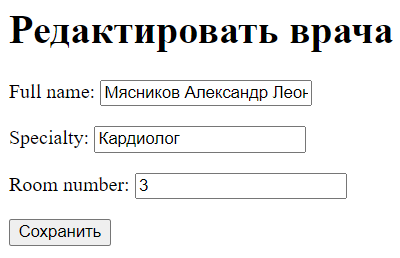
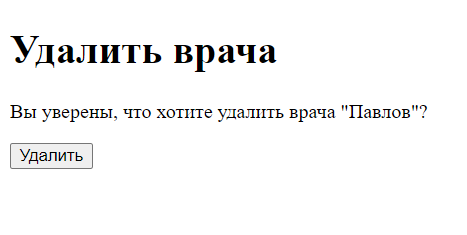
1. DoctorCreateView (рис. 2) - это представление, которое отображает форму для создания нового объекта Doctor. Оно наследуется от класса CreateView. Определены следующие атрибуты:
   1. model указывает модель, с которой работает представление (Doctor в данном случае).
   2. template\_name задает имя шаблона, который будет использоваться для отображения формы создания (doctor\_create.html).
   3. fields определяет, какие поля модели будут отображаться в форме создания. В данном случае это full\_name, specialty и room\_number.
   4. success\_url указывает URL, на который будет перенаправлен пользователь после успешного создания объекта. В данном случае используется reverse\_lazy для получения URL по имени doctor\_list.

Рис. 2. DoctorCreateView.

1. DoctorUpdateView (рис. 3) - это представление, которое отображает форму для обновления существующего объекта Doctor. Оно наследуется от класса UpdateView. Атрибуты, такие как model, template\_name, fields и success\_url, имеют такое же значение, как и в DoctorCreateView.

  
Рис.3. DoctorUpdateView.

1. DoctorDeleteView(Рис. 4) - это представление, которое отображает подтверждение удаления существующего объекта Doctor. Оно наследуется от класса DeleteView. Атрибуты, такие как model, template\_name и success\_url, имеют такое же значение, как и в предыдущих представлениях.

Рис. 4. DoctorDeleteView.

В общем, эти представления обеспечивают стандартный CRUD (создание, чтение, обновление и удаление) функционал для модели Doctor в Django, обрабатывая соответствующие операции и используя указанные шаблоны и поля формы, а также аналогичны методы для всех остальных сущностей. После успешного выполнения каждой операции, пользователь будет перенаправлен на страницу со списком всех врачей (doctor\_list).

**Описание возникших затруднений**.

Из-за отсутствия опыта работы с фреймворком Django при выполнении этой лабораторной работы возникло множество трудностей, связанных с ним.

Одна из таких - подключения фала CSS к html. Были попытки создания с помощью директории static. В итоге стили были добавлены с помощью <style> в HTML файлы.

**Описание альтернативных способов решения**

Всё-таки попробовать подключить стили через static.

**Вывод.**

В ходе работы был реализован веб-сайт в среде Django, соответствующий предложенным требованиям для моделирования истории посещений поликлиники.Для обеспечения надлежащего проектирования базы данных, было создано не менее пяти таблиц, согласно требованиям трех нормальных форм. Это позволяет улучшить эффективность хранения данных и устранить избыточность.Для реализации различных типов связей (one-to-one, one-to-many, many-to-many) были использованы соответствующие отношения между таблицами. Например, связь one-to-one могла быть использована между таблицами "Пациенты" и "Врачи" для связи каждого пациента с его лечащим врачом.