МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**ОТЧЕТ**

**по практике**

**Производственная практика: преддипломная (технологическая**

(наименование практики)

**(проектно-технологическая)) практика**

студентки Еремеевой Виктории Владимировны группы АВТ-009

ФИО студента(ки)

Место проведения практики: НГТУ, "СКБ Робототехника и искусственный

полное наименование предприятия (организации)

интеллект"

Сроки практики по учебному плану: с «11» марта 2024г. по «18» мая 2024г.

Руководитель практики от университета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись , оценка

Осипенко Иван Викторович, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО, должность

Руководитель практики от предприятия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись , оценка

Осипенко Иван Викторович, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО, должность

Оценка по итогам аттестации студента комиссией:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Ильиных С.П./

подпись , ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Малявко А.А./

подпись , ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Токарев В.Г./

подпись , ФИО

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

подпись , ФИО

НОВОСИБИРСК

2024

Оглавление

[1 Структура подразделения и организация производственно-технологического процесса в СКБ «Робототехника» 3](#_Toc166010734)

[2 Практическая часть 5](#_Toc166010735)

[Введение 5](#_Toc166010736)

[2.1 Анализ существующих решений 7](#_Toc166010737)

[2.2 Разработка архитектуры приложения 10](#_Toc166010738)

[2.2.1 Структура приложения 10](#_Toc166010739)

[2.2.2 Диаграммы классов и диаграммы «сущность-связь» 12](#_Toc166010740)

[2.2.3 Диаграмма компонент 13](#_Toc166010741)

[2.3 Реализация компонентов разработки 14](#_Toc166010742)

[Заключение 28](#_Toc166010743)

[Список источников 29](#_Toc166010744)

[Приложение 31](#_Toc166010745)

[A. Диаграмма сущность-связь БД (Entity-Relationship) для собственных моделей 31](#_Toc166010746)

[B. Диаграмма сущность-связь автоматически сформированная 32](#_Toc166010747)

[C. Диаграмма классов приложения. Классы моделей 33](#_Toc166010748)

[D. Диаграмма классов приложения. Классы представлений 34](#_Toc166010749)

[E. Скрины работы программы 35](#_Toc166010750)

# Структура подразделения и организация производственно-технологического процесса в СКБ «Робототехника»

В рамках Студенческого Конструкторского Бюро «Робототехника» проводятся занятия для школьников. Для студентов также доступны курсы повышения квалификации, программа переподготовки, лекции и мастер-классы. СКБ «Робототехника» располагается во втором корпусе на первом этаже в кабинете 58-65. Здесь можно воплотить свои творческие и технические идеи в жизнь. Предоставляется возможность печати плат и их программирования [1].

Основные цели конструкторского бюро "Робототехника" включают:

* Знакомство школьников, студентов и молодых специалистов с современным оборудованием, технологиями и материалами.
* Обеспечение возможностью для учебы, обмена опытом.
* Поддержка и развитие инновационного творчества студентов.
* Техническая и производственная поддержка проектов.
* Помощь в реализации творческих технических и инновационных идей, как молодым специалистам, так и начинающим предпринимателям.
* Исследование и разработка перспективных видов продукции и технологий;
* Коммерциализация идей и инновационных разработок.

Деятельность бюро охватывает широкий спектр направлений:

* программирование;
* робототехника;
* 3d моделирование;
* конструирование;
* 3d печать;
* схемотехника;
* интернет вещей.

Огромную помощь оказывает бюро с коммерциализацией и патентами. СКБ помогает с защитой авторского права и подготавливает к выходу на рынок.

СКБ участвует в различных конкурсах, выставках и фестивалях различного уровня, предоставляя участникам возможность соревноваться с другими участниками и выиграть ценные призы в виде грантов на дальнейшее развитие и реализацию проектов.

СКБ «Робототехника» предоставляет доступ к своей базе знаний для создания роботов, а также видеоурокам на YouTube-канале «Me-robotics» [1].

# Практическая часть

## Введение

В современном мире информация играет ключевую роль в эффективности любой организации. Растёт количество информациии. Процессы её обработки становится невозможно выполнять вручную. Системы требуют компьютеризации для повышения скорости и недопущения ошибок.

В образовательной сфере, создание информационной системы становится критически важным. Образовательная среда требует инноваций, включая использование общей базы данных для диспетчеризации режимной части школы.

Сегодняшние школы используют информационные системы более интенсивно и постоянно. Они активно заняты не только в урочное время, но и во внеучебное, проводя различные кружки и сотрудничая с внешними организациями, что способствует развитию учебного заведения, повышению его эффективности. Увеличилась численность обучающихся и преподавателей, при этом учителя теперь меньше загружены.

Из вышесказанного следует, что одной из первостепенных задач современной системы образования является автоматизация процессов расписания.

В обычной практике, данные для составления расписания заносятся разными людьми в таблицу Excel (учебный план, нагрузка), которая затем пересылается по электронной почте друг другу. Расписание обычно создается на миллиметровке с учителями в строках, днями и номерами уроков в столбцах. Затем это расписание переносится в таблицу Excel с классами в столбцах и днями, номерами уроков в строках. В процессе этого переноса часто обнаруживаются наложения и «окна», которые затем требуется исправлять. Процесс составления расписаний является очень трудоёмким.

Изложенные проблемы подчеркивают актуальность создания приложения «Расписание занятий» для образовательных учреждений. Автоматизация этого процесса поможет сократить трудозатраты и уменьшить вероятность ошибок, повышая эффективность учебного процесса. А также может помочь множество форм представления информации с возможностью выбора удобной конкретным пользователем.

**Целью** моей работы является проектирование пользовательского интерфейса приложения «Расписание». Для этого я поставила перед собой следующие **задачи**:

* Ознакомиться с организационной структурой СКБ, ознакомиться с организацией образовательного и технологического процесса, направлениями научной и научно-исследовательской работы.
* Провести анализ существующих решений.
* Разработать архитектуру приложения; создать модели; написать шаблоны страниц; оформить стили проекта.
* Определить функционала.
* Произвести тестирование интерфейса и функциональное тестирование.

## Анализ существующих решений

Существует множество программных решений для автоматизации составления расписания в образовательных учреждениях, каждое из которых обладает своими особенностями и преимуществами. Программные решения для управления расписанием в образовательных учреждениях представлены в виде десктопных приложений, предназначенных, как правило, для установки на одну машину. Все перечисленные программы способны составлять расписание, поддерживать его корректировки, работать с несколькими сменами и различными предметами у учителей. Многие поддерживают отчётность или экспорт информации в различные удобные форматы типа Word, Excel. Различия могут быть как в функциональности, так и в цене и удобстве использования. При выборе программы следует учитывать индивидуальные потребности и особенности образовательного учреждения.

1. Экспресс-расписание Колледж [3]

* Есть проверка аудиторий на вместимость в связи с возникшей проблемой COVID-19.
* Цена: 2 500 – 300 000 руб.

1. 1С:Автоматизированное составление расписания.Колледж [2]

* Преимуществом является учёт времени переходов между корпусами.
* Решение из Новосибирска.
* 70 заведений.
* Цена: не указана на сайте.

1. Система «АВТОРасписание» [10]

300 заведений.

* Развивается с 1996 года.
* Наличие подробной статистики и объективной оценки качества любого варианта расписания.
* Возможность поддержки любого национального языка (по желанию клиента).
* Цена: 20 000 – 150 000.

1. Хронограф [4]

* Трудно заполнять таблицы и вносить данные.
* Не интуитивно-понятный интерфейс.
* Сомнительное качество и мало возможностей.
* Цена: бесплатно

1. «НИКА-Колледж» [5]

* Возможности сдваивания и страивания.
* Экспорт в Exel.
* Учёт переходов между корпусами.
* Учёт пожеланий учителей и совместителей.
* Лицензия на 5 лет и помесячно (1, 3, 6).
* Возможность докупать отдельные модули.
* Цены: 11 000 – 25 000 на 5 лет. Демо ~2 500.

1. aSc [6]

* Зарубежная программа.
* По описанию имеет хорошее качество.
* Цена: 150 – 2 000 € в год ~ 15 000 – 200 000 рублей.

Рассмотренные выше сервисы обладают рядом достоинств и недостатков.

Достоинства:

* позволяют быстро создавать расписания с минимальными усилиями;
* могут учитывать различные ограничения и требования, что позволяет создавать более точные расписания;
* позволяют легко изменять расписания при изменении условий и требований;
* могут повысить эффективность работы организации;

Недостатки:

* правильная настройка может потребовать времени и усилий;
* могут не учитывать все особенности конкретной ситуации, что может привести к некачественному расписанию;
* если система перестанет работать, это может привести к проблемам с расписанием;
* некоторые системы могут быть довольно дорогими, особенно если они требуют постоянного обновления и поддержки.
* существующие решения не имеют доступа в интернет и не привязаны к образовательным платформам.

Естественно стремление каждого разработчика устранить существующие недостатки при реализации своей системы с сохранением достоинств других. И я не являюсь исключением. Особенно меня привлекла идея, чтобы моя программа учитывала возможность автоматического переноса на сайт школы и встраиваемость в образовательные платформы типа «Дневник.ру», «Электронная школа» и другие.

## Разработка архитектуры приложения

### Структура приложения

Проект будет разрабатываться с помощью фреймворка джанго. В корне проекта будет лежать папка .venv. Она необходима для изоляции зависимостей проекта от глобального окружения Python, что помогает предотвратить конфликты версий пакетов и обеспечить более стабильное выполнение кода.

Чтобы создать папку .venv в PyCharm ничего дополнительного делать не нужно. Она создаётся самостоятельно. На Linux нужно прописать команду

1. python -m venv .venv

для создания виртуального окружения. А после прописать

1. .venv\Scripts\activate

в терминале для активации виртуальной среды.

Далее устанавливаем все необходимые пакеты.

1. pip install -r requirements.txt

Далее будем придерживаться стандартной структуры джанго-проектов:

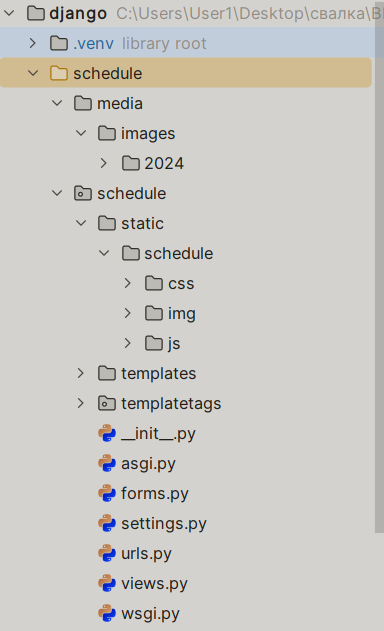


Рисунок 2.1. Файловая структура проекта

В корне приложения располагается главный файл приложения – **manage.py**. Папка **media** создаётся позже приложением самостоятельно. Туда будут сохраняться все файлы, загруженные на сервер.

Далее опишем главное приложение. В папке **static** находятся скрипты js, картинки и файлы стилей. В папке **templates** лежат html-шаблоны. В папке **templatetags** хранятся файлы для встраивания шаблонов в шаблоны. Там будет храниться главное меню head, которое будет отрисовываться на каждой странице.

Опишем стандартные файлы. Файл \_\_**init**\_\_ служит для инициализации пакета и является пустым. В файле **forms** можно хранить формы. В **settings** хранятся настройки проекта. В файле **urls** сопоставляются url-адреса функциям обработки и отрисовки. В файле **views** хранятся функции и классы обработки и отрисовки.

Теперь рассмотрим какое-нибудь приложение, содержащееся в проекте. Приложения создаются командой

1. python manage.py startapp teachers

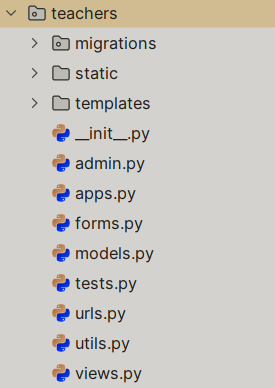


Рисунок 2.2. Файловая структура приложения teachers

Приложение отличается от главного тем, что по умолчанию в него добавляется папка **migrations**. Файлы в ней создаются автоматически после выполнения миграций. Также здесь присутствует файл **models.py,** в котором хранятся сущности приложения и файл **admin**, где задаются способы представления и управления моделями в админ-приложении. Файл **utils** мы создадим самостоятельно для хранения миксинов.

### Диаграммы классов и диаграммы «сущность-связь»

Диаграммы классов в основном связаны с моделями и представлены в приложениях. В приложении A представлены модели, существующие в приложении. Всего их три: классы, учителя, дисциплины. Это простые таблицы и они соответствуют 3-нормальной форме. Класс учителя содержит ФИО, фото, кабинет, датe создания и датe последнего обновления, слаг. Классы содержат цифру, букву и слаг. Дисциплины представляют собой список предметов и их сокращений, а также слаг. Позже эти таблицы можно расширить, разделив ФИО, например, по отдельным колонкам, добавив телефоны, руководителей МО, закрепление кабинетов за учителями и многое другое.

Структуру классов в пайтоне можно увидеть в приложении C.

Помимо созданных разработчиком моделей при выполнении миграций

1. python.exe manage.py makemigrations
2. python.exe manage.py migrate

создаются служебные таблицы для создания групп и пользователей, их авторизации, хранения прав. Они представлены в приложении B.

Помимо таблиц в нормальной форме содержится ещё 3 таблицы, реализующие связи «многие-ко-многим». К ним относится таблица **program**, которая содержит столбцы класса и предмета, а также количество занятий в неделю. Таблица **teacherSubject** необходима, чтобы показать, какие предметы ведёт учитель. В ней 2 столбца: учитель и предмет. Там указываются те предметы, которые может вести учитель, то есть по которым учитель имеет специальность. Также существует главная таблица, которая состоит из двух связей «многие-ко-многим». Это таблица **teacherSubjectClass**. В ней указывается, в каком именно классе на текущий момент учитель преподаёт свой предмет. Повторения в ней недопустимы, поскольку несколько учителей не могут вести предмет в классе. Это временная мера в процессе разработки, потому что в реальности у класса может существовать 2 учителя физкультуры, английского или технологии. То есть когда классы разбиваются на подгруппы.

### Диаграмма компонент

На рисунке ниже представлена стандартная диаграмма компонент django-приложение. Наше приложение целиком соответствует этой диаграмме. При запросе в веб-браузере происходит запрос на сервер в django. Адрес сопоставляется с известными адресами, которое наше приложение умеет обрабатывать. В случае, если приложение не знает такого адреса, то отправит ответ вида 404. Файл **urls** ищет функции в файле **views**. Он в свою очередь оперирует БД (при необходимости), отрисовывает шаблоны из файлов **templates**, передаёт туда данные. **Templates** могут обращаться к статическим файлам типа стилей и скриптов.

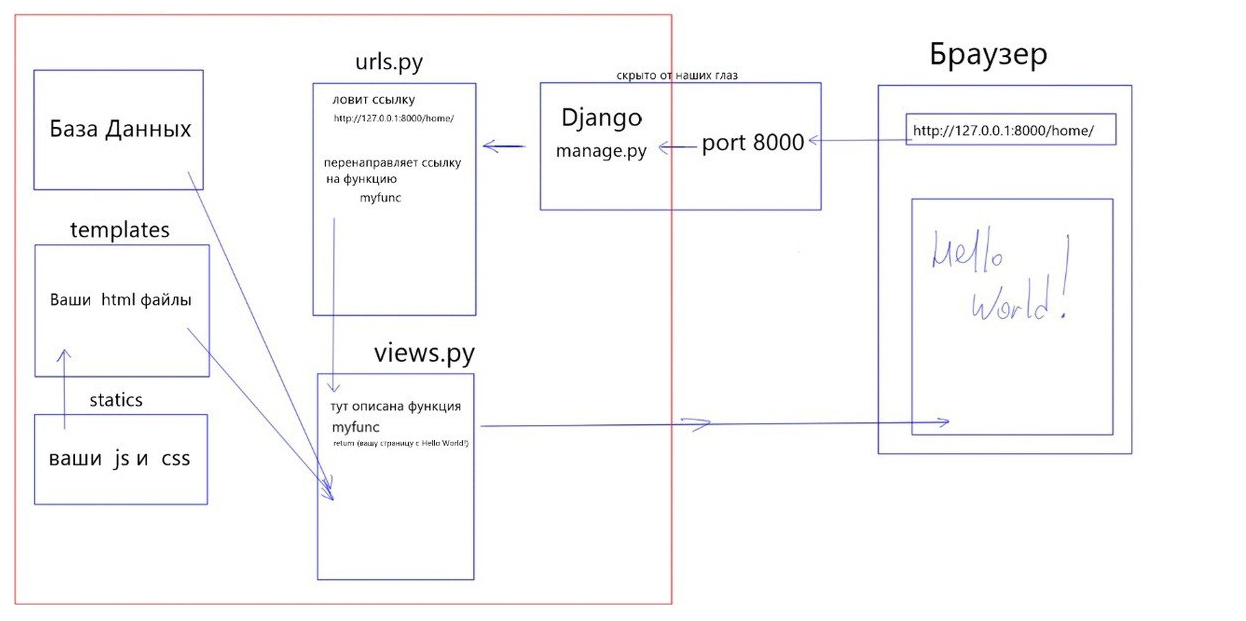


Рисунок 2.3. Диаграмма компонент

## Реализация компонентов разработки

как создать приложение (нормально описать) (с инстал апс)

python manage.py shell

почему транзакшн атомик

Разработка приложения начинается с файла настроек **settings**. Покажем важные настройки приложения. Нужно настроить разрешённые хосты. Пока это только наш компьютер:

1. DEBUG = True
2. ALLOWED\_HOSTS = [
3. '127.0.0.1',
4. ]

Затем вписываем нашу БД:

1. DATABASES = {
2. 'default': {
3. 'ENGINE': 'django.db.backends.postgresql',
4. 'NAME': 'schedule',
5. 'USER': 'postgres',
6. 'PASSWORD': '32767',
7. 'HOST': 'localhost',
8. 'PORT': '3333',
9. }
10. }

Добавим папку, которая будет хранить все статические файлы. Теоретически внутри каждого приложения проекта имеется папка static. Практически на сервере все файлы из этих папок собираются в одну внешнюю папку проекта. Именно поэтому нужно указывать несколько настроек.

1. STATIC\_URL = '/static/'
2. STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')
3. STATICFILES\_DIRS = [
4. os.path.join(BASE\_DIR, 'schedule/static'),
5. ]

По тем же причинам настраиваем папку media для динамического сохранения файлов.

1. MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')
2. MEDIA\_URL = '/media/'

А также на этапах отладки добавим дебаггер в веб-браузер:

1. INSTALLED\_APPS = […,
2. 'debug\_toolbar',
3. ]
4. MIDDLEWARE = […,
5. "debug\_toolbar.middleware.DebugToolbarMiddleware",
6. ]

Покажем, создание администратора:

python manage.py createsuperuser

Заполним остальные поля, которые потребует терминал.

Далее создадим меню и добавим его на главную страницу. В файле views напишем все пункты меню, напишем класс представления, который наследуется от готового [7]. В нём задаётся имя шаблона и передаваемые параметры:

1. menus = [
2. {'url': '/home', 'title': 'Главная'},
3. {'url': '/teachers', 'title': 'Учителя'},
4. {'url': '/classes', 'title': 'Классы'},…
5. ]
6. class HomeView(TemplateView):
7. template\_name = 'main\_base.html'
8. extra\_context = {
9. 'menu\_selected': menus[0]['url'],
10. }

В самой первой строке разрешается поиск путей до статических файлов в папке **static**. Далее подгружаются наши теги:

{%load static%}

{% load base\_tags %}

О них поговорим чуть позже. Ссылки href до статических файлов в django выглядят так [8]:

href="{% static 'schedule/css/style.css' %}"

Чтобы шаблоны правильно вставлялись друг в друга нужно в местах возможной вставки прописывать блоки как в строке 5:

1. <head>
2. …
3. <title>Школьное расписание</title>
4. <link rel="icon" type="image/x-icon" href="{% static 'schedule/img/samolet.ico' %}">
5. {% block exta\_head%}{% endblock %}
6. </head>

Покажем вызов функции и передачу аргументов в неё (строка 3):

1. <header class="d-flex justify-content-center py-3">
2. <ul class="nav nav-pills">
3. {% get\_menu menu\_selected %}
4. </ul>
5. </header>

Теперь посмотрим на файл **base\_tags** в папке **templatetags**:

1. from django import template
2. from schedule.views import menus
3. register = template.Library()
4. @register.inclusion\_tag('menus.html')
5. def get\_menu(menu\_selected=menus[0]['url']):
6. return {'menu\_selected': menu\_selected, 'menus': menus}

В эту функцию всегда будет поступать выбранный пункт и будет отрисовываться дополнительный шаблон декоратора **menus.html**:

1. {% for menu in menus %}
2. {% if menu.url in menu\_selected %}
3. <li class="nav-item"><a href="{{menu.url}}" class="nav-link active">{{menu.title}}</a></li>
4. {% else %}
5. <li class="nav-item"><a href="{{menu.url}}" class="nav-link">{{menu.title}}</a></li>
6. {% endif %}
7. {% endfor %}

Это всё содержимое файла. в нём используется цикла для отрисовки всех элементов и если какой-то элемент выбран, то он будет дополнительно подсвечен стилями класса active.

Далее необходимо соединить адрес запроса с обрабатывающей функцией в файлу urls(4). Также добавим в urls сайт администратора (он уже лежит в пакете django)(2). И боковую панель для отладки debug(5). Вставим перенаправление 3 из готовых классов django.views.generic:

1. urlpatterns = [
2. path('admin/', admin.site.urls),
3. path('', RedirectView.as\_view(url='/home/', permanent=True)),
4. path('home/', HomeView.as\_view(), name='home'),
5. path("\_\_debug\_\_/", include("debug\_toolbar.urls")),
6. ]

Также в этом файле нужно прописать настройки для папки media, так как при отладке используются другие настройки сервера. Этого требует документация [8].

1. if settings.DEBUG:
2. urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Приложение запускается с помощью команды

python.exe manage.py runserver

Можно прописать после этой команды порт, на котором запускаем приложение. В нашем проекте работает стандартный. Перейдём по ссылке:

http://127.0.0.1:8000/

Посмотрим на результат:

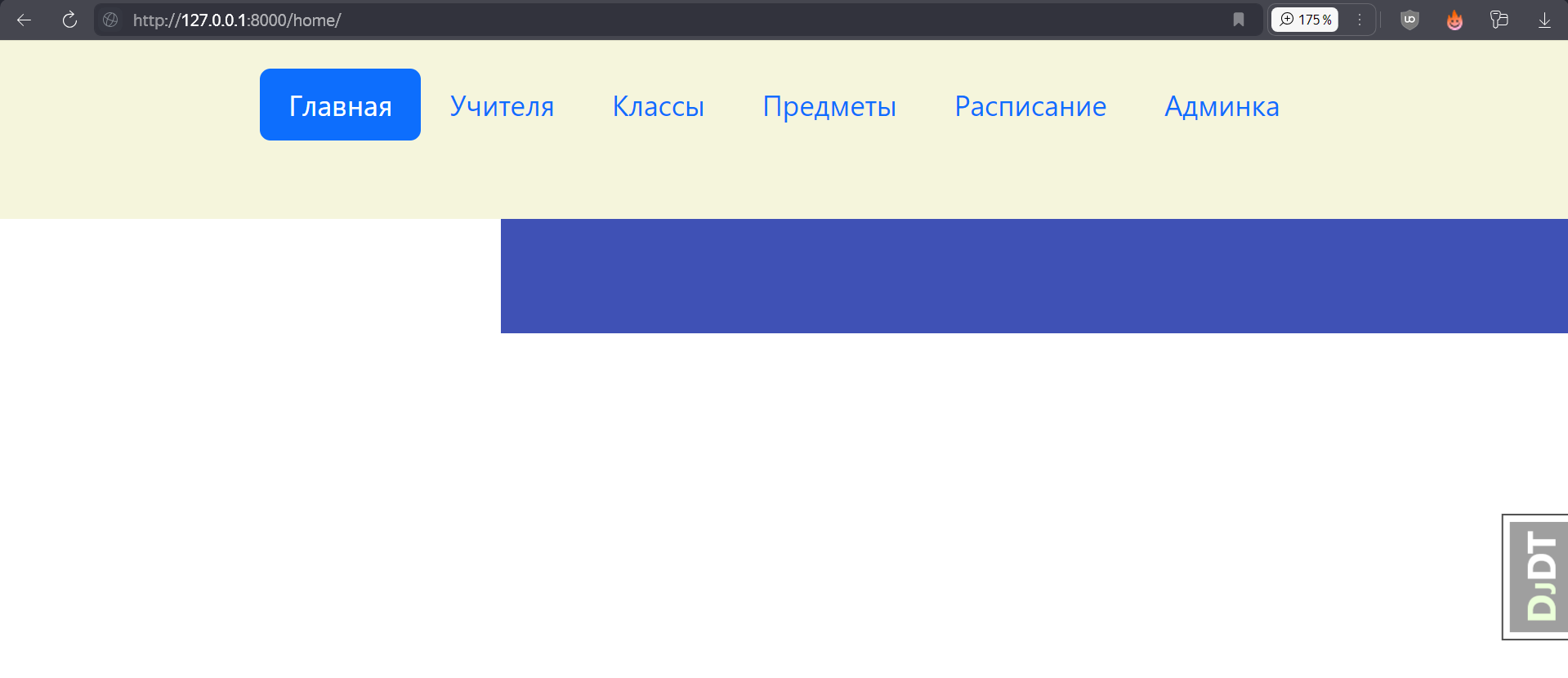


Рисунок 2.4. Главная страница приложения

На картинке уже применены некоторые стили. А также подгружен **bootstrap**. Выбранный раздел меню подсвечивается. Справа есть панель отладки.

Теперь создадим другое приложение с помощью команды:

python.exe manage.py startapp teachers

Покажем написание класса модели:

1. class Teacher(models.Model):
2. fio = models.CharField(verbose\_name='ФИО', max\_length=255)
3. slug = models.SlugField(max\_length=255, unique=True, db\_index=True, verbose\_name='URL')
4. photo = models.ImageField(upload\_to='images/%Y/%m/%d', null=True, blank=True, verbose\_name='Фото')
5. room = models.PositiveIntegerField(verbose\_name='Кабинет', null=True, blank=True)
6. date\_create = models.DateField(auto\_now\_add=True, verbose\_name='Дата создания')
7. date\_update = models.DateField(auto\_now=True, verbose\_name='Дата обновления')
8. subject = models.ManyToManyField('Discipline', blank=True, verbose\_name='Предметы')
9. class Meta:
10. verbose\_name = 'Учитель'
11. verbose\_name\_plural = 'Учителя'
12. ordering = ['fio']
13. def get\_absolute\_url(self):
14. return reverse('teacher\_read', kwargs={'slug': self.slug})
15. def \_\_str\_\_(self):
16. return self.fio
17. def save(self, \*args, \*\*kwargs):
18. self.slug = slugify(self.\_\_str\_\_())
19. super(Teacher, self).save(\*args, \*\*kwargs)
20. @property
21. def serializable(self):
22. return {
23. 'id': self.id,
24. 'fio': self.fio,
25. 'slug': self.slug,
26. 'str': self.\_\_str\_\_()
27. }

Для создания модели нужно отнаследоваться от класса django.db.models.Model. Далее описываем поля модели с помощью специальных классов. В них во всех указываем verbose\_name. Это имя, которое будет использоваться для подписей на сайте автоматически. Например, в формах элемент **label-for**. Почти все поля имеют такие параметры, как обязательность/необязательность, значение по умолчанию, уникальность, индекс, макимальный и минимальный размер и многое другое. Подробнее хочется остановиться на поле фото (4), которое имеет путь, куда будут загружаться и где будут искаться фотографии. Он имеет форму шаблона, куда подставляются год, месяц и день. Это сделано для более удобного поиска по дате и чтобы снизить повторяемость фотографий. Но если в один и тот же день будут загружены фото с одинаковыми именами, методы django автоматически припишут к нему уникальные символы и самостоятельно припишут их в путь в БД. Есть удобные параметры для заполнения дат **auto\_now**. Они автоматически запишут текущее время при создании и изменении. Перейдём к формированию связей **ManyToManyField** [9]. Оно нужно для создания дополнительной таблицы, которая содержит id обоих объектов. Туда нужно передавать класс второго объекта. В модели автоматически создаётся поле id. В моделях с зависимыми ключами или множественными связями к полям сзади приписывается \_id. Все таблицы в БД хранятся в виде <имя приложения><имя модели>.

После описания полей пропишем способы отображения модели на странице во вложенном классе **Meta**. В методе get\_absolute\_url автоматически формируется путь до объекта. Магический метод str показывает, как печатать объект. В методе сохранения прописывается автоматическое формирование слага. В данный момент отсутствуют проверки на ошибки повтора и прочие. Метод сериализации нужен для упаковки объекта в json. С ними мы работаем в js.

Покажем теперь функцию обработки, чтобы вывести все объекты на экран:

1. class TeacherListView(DateMixin, ListView):
2. model = Teacher
3. template\_name = 'teachers/all.html'
4. context\_object\_name = 'teachers'
5. title = 'Учителя'
6. def get\_context\_data(self, \*\*kwargs):
7. context = super().get\_context\_data(\*\*kwargs)
8. return self.get\_mixin\_context(context, menu\_selected=self.request.path, \*\*kwargs)

Здесь просто указывается модель и все остальные запросы производит родительский класс. Также можно задавать параметры, которые будут передаваться в контекст шаблона. Проблема в том, что мы не имеем в данном месте доступа к динамическим полям типа путь перехода, который нам необходим для подсветки меню. Поэтому мы используем специальный метод get\_context\_data и класс миксинов, которые будут передавать данные в наш контекст [7]:

1. class DateMixin:
2. title = None
3. selected\_menu = None
4. paginate\_by = 2
5. extra\_context = {}
6. def \_\_init\_\_(self):
7. if self.title:
8. self.extra\_context['title'] = self.title
9. if self.selected\_menu:
10. self.extra\_context['selected\_menu'] = self.selected\_menu
11. def get\_mixin\_context(self, context, \*\*kwargs):
12. context.update(\*\*kwargs)
13. return context

Тут мы объявляем все поля и отправляем в контекст с помощью метода context.update().

Покажем отрывок html-файла. Здесь иннтересно посмотреть на функцию url (4), которая будет формировать url по имени шаблона и передавать туда параметр:

1. {% for teacher in teachers %}
2. <li class="grid\_\_item">
3. <a class="link" href="{% url 'teacher\_update' slug=teacher.slug %}">
4. <article class="product">
5. {%if teacher.photo%}
6. <img class="product\_\_image"
7. src="{{ teacher.photo.url }}"
8. alt="{{ teacher.fio }} photo"/>
9. {%endif%}
10. <h2 class="product\_\_name">{{teacher.fio}}</h2>
11. <p class="product\_\_price">Учитель</p>
12. </article>
13. </a>
14. </li>
15. {% endfor %}

Покажем соответствующие url:

path('<slug:slug>/update', UpdateTeacher.as\_view(), name='teacher\_update'),

path('', TeacherListView.as\_view(), name='teachers'),

Ссылка teachers не печатается в приложении, потому что мы обозначили, что все ссылки идут через этот путь в главном приложении в urls.py:

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

Посмотрим на результат:

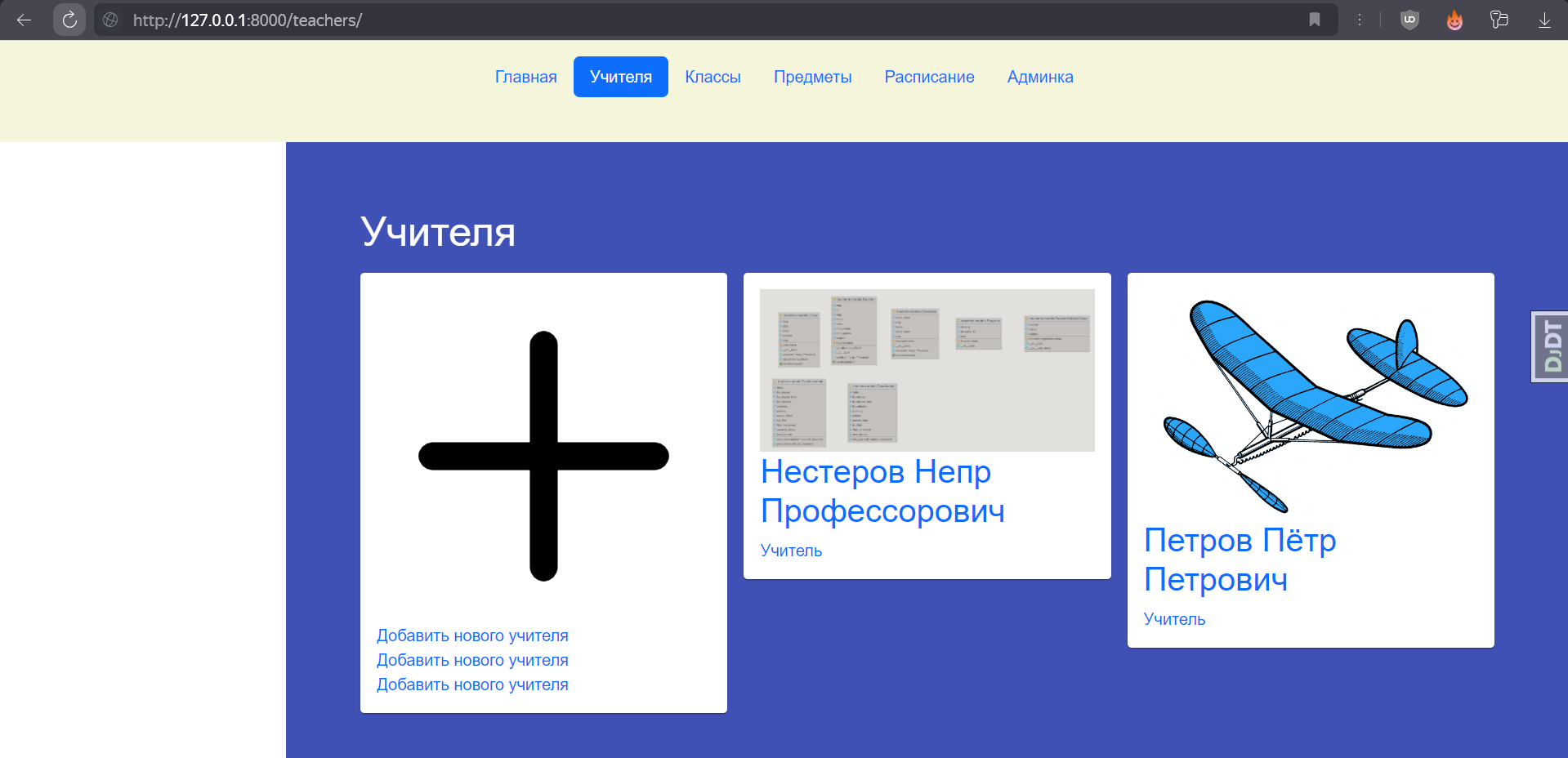


Рисунок 2.5. Страница учителей

Плюс сформирован тем же образом, что и остальные карточки.

Интересно показать страничку обновления учителя, к которой переходишь по клику на учителе:

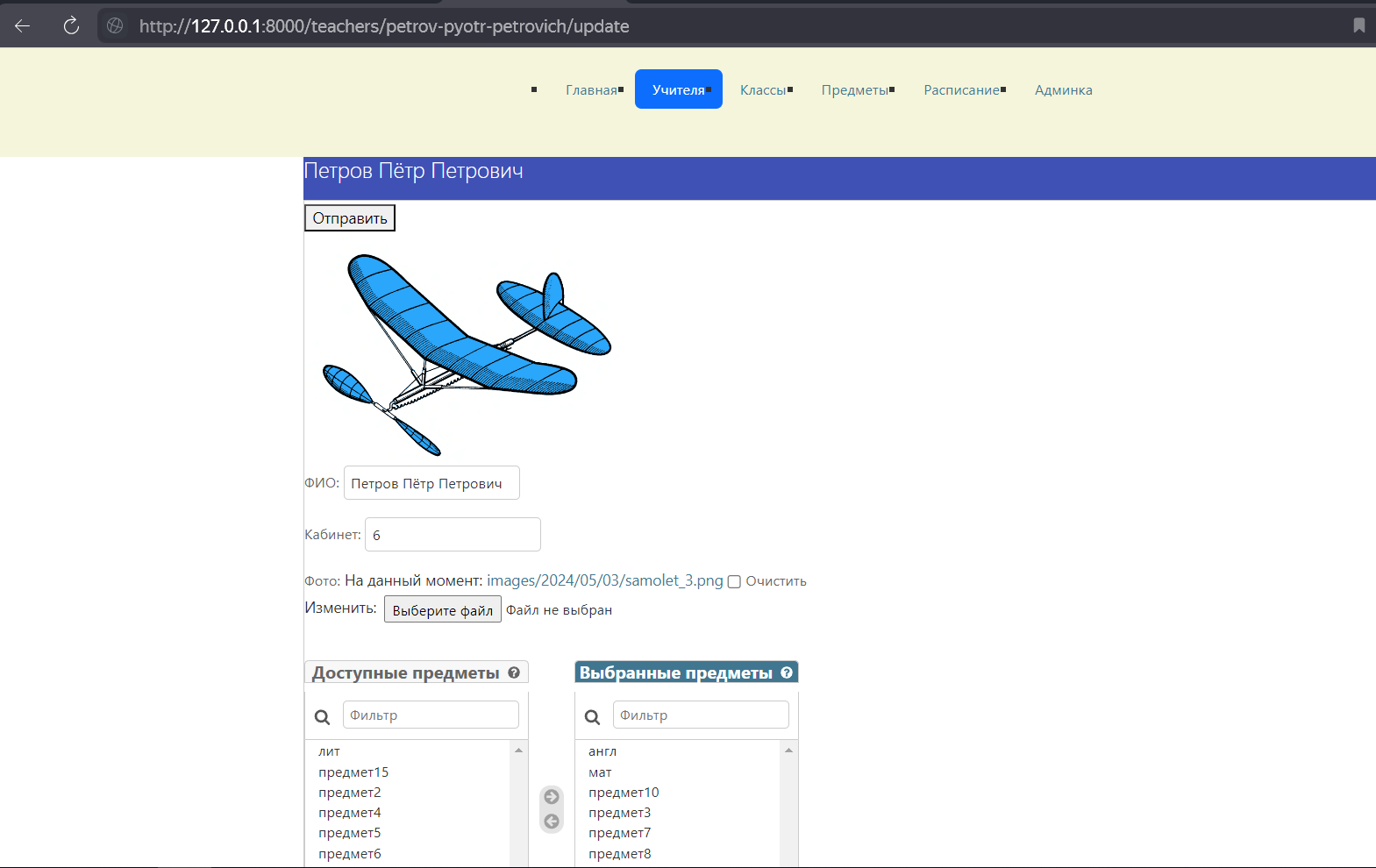


Рисунок 2.6. Страница обновления учителя

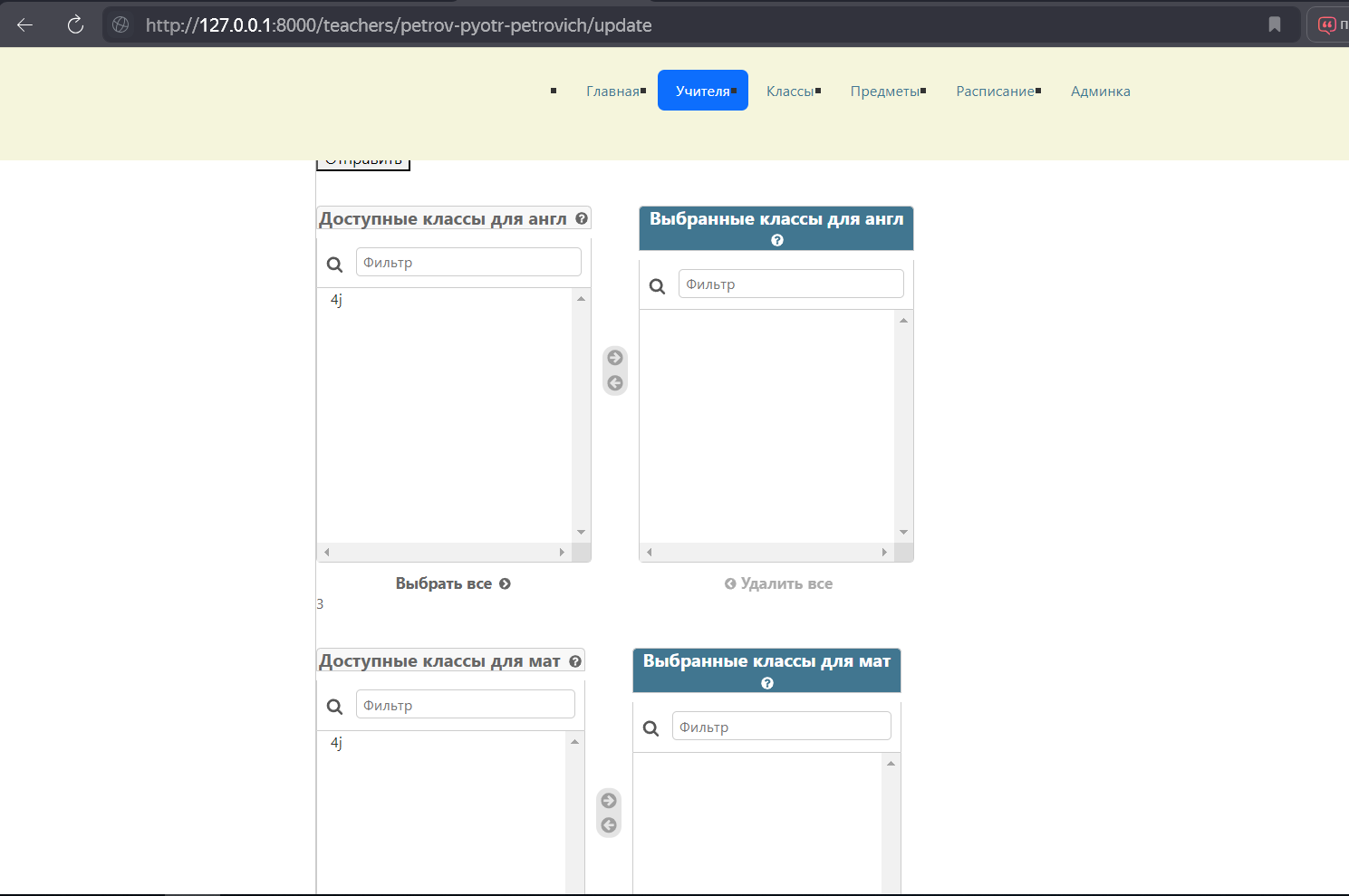


Рисунок 2.7. Выпадение списка классов

В ней есть множественный селектор, оформленный двумя списками для выбора предмета. А также по клику на предмет снизу выпадает список классов, имеющих этот предмет в программе. Это реализуется с помощью кода js. Во-первых, для этого нужно подключить файл тех же включений, что и в панели администратора (css и js):

1. {%block exta\_head%}
2. {% include 'classes/admin\_includes.html' %}
3. <script type="text/javascript" src="{% static '/teachers/js/teachers.js' %}?v={{ generate\_unique\_hash\_or\_timestamp"></script>
4. {% endblock %}

Во-вторых, сделаем форматирование списка как в панели администратора джанго:

1. <script type="text/javascript">
2. $(window).on('load', function (e) {
3. SelectFilter.init("id\_subject", " предметы", 0, "/static/admin/");
4. });
5. </script>

И посмотрим, как легко и просто отображается наша форма:

1. <form method="post" enctype="multipart/form-data" id="form" onsubmit="handleFormSubmit(event)">
2. <input type="hidden" id="teacherId" name="objectId" value="{{ id }}">
3. <img src="{{ form.photo.url }}" alt="{{ teacher.fio }} photo">
4. {% csrf\_token %}
5. {{ form.as\_p }}
6. <h1>Выберите классы</h1>
7. <p>
8. <button type="submit">Отправить</button>
9. </p>
10. </form>

Здесь мы устанавливаем обработчик для её отправки в строке 1. Использование enctype="multipart/form-data" в форме HTML указывает браузеру на то, что форма будет содержать файлы, которые будут загружены на сервер. В строке 5 мы устанавливаем специальный токен csrf (Cross-Site Request Forgery), который означает, что формы мы отправляем самостоятельно, а не производим DDOS-атаки. Пользователю выдаётся уникальный токен, вторая копия которого хранится на сервере. И воспользоваться им можно только единожды. В 6 вставляем форму. Джанго сам способен её развернуть во все label и правильные поля ввода.

Далее посмотрим на mutationObserver. Так как js-код мы писали не самостоятельно, а взяли готовый, то будет трудно копаться в чужих файлах и добавлять необходимые обработки. Сначала установим слушателя на появление элемента с предметами:

1. const observer = new MutationObserver((mutationsList, observer) => {
2. mutationsList.forEach(mutation => {
3. if (mutation.addedNodes && mutation.addedNodes.length > 0) {
4. if (Array.from(mutation.addedNodes).some(node => node.id === 'id\_subject\_to')) {
5. setListeners();
6. }
7. }
8. });
9. });
10. const config = {childList: true, subtree: true};
11. observer.observe(document, config);

Потом мы установим слушателя, который при изменении правого списка будет выполнять обработку, а именно добавлять ещё один список для выбора классов:

1. // Основная функция программы
2. function setListeners() {
3. const selectorTo = document.querySelector('#id\_subject\_to');
4. oldOptions = selectorTo.querySelectorAll('option');
5. oldOptions = Array.from(oldOptions);
6. …
7. const observerTo = new MutationObserver(handleSelectTo);
8. observerTo.observe(selectorTo, {childList: true});
9. document.querySelectorAll('#id\_subject\_to').forEach(function (selectorTo) {
10. addOptions(selectorTo);
11. });
12. }

Сначала мы смотрим изначальные выбранные элементы и запоминаем их. И применяем к уже содержащимся на странице элементам функцию добавления списков. А также ставим mutationObserver. Рассмотрим функцию handleSelectTo:

1. function handleSelectTo(mutationsList, observer) {
2. newOptions = selectorTo.querySelectorAll('option');
3. newOptions = Array.from(newOptions);
4. const addedOptions = [];
5. const removedOptions = [];
6. newOptions.forEach(el\_A => {
7. if (!oldOptions.some(el\_B => el\_B.value === el\_A.value)) {
8. addedOptions.push(el\_A);
9. }
10. });
11. oldOptions.forEach(el\_A => {
12. if (!newOptions.some(el\_B => el\_B.value === el\_A.value)) {
13. removedOptions.push(el\_A);
14. }
15. })
16. if (addedOptions.length > 0) {
17. addOptions(addedOptions);
18. }
19. if (removedOptions.length > 0) {
20. removedOptions.forEach(option => {
21. document.getElementById("p-select-" + option.value).remove();
22. });
23. }
24. oldOptions = newOptions;
25. }

Пришлось написать очень громоздкую функцию, потому что простое решение выдавало ошибки. У автора при добавлении к трём предметам четвётрого сначала удалялись все 3, а затем добавлялись снова четыре. Поэтому придётся хранить старые опции и находить различие их с новыми (2-18). Затем удаляем списки для удалённых опций и вызываем функцию создания списков для добавленных:

1. function addOptions(options) {
2. values = [];
3. options = Array.from(options);
4. options.forEach(function (option) {
5. values.push(option.value)
6. });
7. $.ajax({
8. type: 'GET',
9. url: '/teachers/getDataFromDB/',
10. data: {
11. selectedValues: values,
12. teacherId: document.getElementById('teacherId').value
13. },
14. success: function (response) {
15. response['array'].forEach(function (elem) {
16. subject = elem.subject;
17. classes = elem.classes;
18. selectedClasses = elem.selectedClassesId;
19. var pElement = $('<p id="p-select-' + subject.id + '" name="p-select-' + subject.id + '">');
20. var labelElement = $('<label for="select-' + subject.id + '">' + subject.id + '</label>');
21. var selectElement = $('<select name="select-' + subject.id + '" multiple id="id\_select-' + subject.id + '">');
22. classes.forEach(function (classObj) {
23. var optionElement = $('<option value="' + classObj.id + '">' + classObj.str + '</option>');
24. if (selectedClasses.includes(classObj.id)) {
25. optionElement.attr('selected', 'selected');
26. }
27. selectElement.append(optionElement);
28. });
29. container = document.getElementById("form");
30. pElement.append(labelElement);
31. pElement.append(selectElement);
32. $('#form').append(pElement);
33. SelectFilter.init("id\_select-" + subject.id, subject.str, 0, "/static/admin/");
34. });
35. },
36. error: function (err) {
37. console.error('Произошла ошибка при получении данных из базы данных');
38. }
39. });
40. }

В этой функции мы отправляем на сервер джанго ajax-запрос с выбранными опциями (7-13). Затем мы получаем в ответе json из классов, соответствующих предемету и выбранных классов (15-18). Оборачиваем значения в списки (20-40).

Теперь рассмотрим функцию по url: '/teachers/getDataFromDB/':

1. def getDataFromDB(request):
2. selected\_values = request.GET.getlist('selectedValues[]')
3. teacher\_id = request.GET.get('teacherId')
4. teacher = get\_object\_or\_404(Teacher, id=teacher\_id)
5. classes\_by\_subjects = {'array': [], }
6. for selectedValue in selected\_values:
7. subject = get\_object\_or\_404(Discipline, id=selectedValue)
8. classes\_with\_subject = Class.objects.filter(subject=subject)
9. selected\_classes\_strs = TeacherSubjectClass.objects.filter(subject=subject, teacher=teacher)
10. subject\_data = {
11. 'subject': subject.serializable,
12. 'classes': [cls.serializable for cls in classes\_with\_subject],
13. 'selectedClassesId': [selected\_str.\_class.id for selected\_str in selected\_classes\_strs],
14. }
15. classes\_by\_subjects['array'].append(subject\_data)
16. return JsonResponse(classes\_by\_subjects)

В ней мы принимаем значения (2-3). Далее происходит выборка из таблиц тех классов, в программе которых есть выбранный предмет и выбранные классы из таблицы, где учителю уже назначен класс (8-19). Далее упаковываем все элементы в json и отправляем в js.

Рассмотрим теперь сохранение формы. Мы сразу поставили на неё обработчик. Заглянем в js:

1. function handleFormSubmit(event) {
2. // event.preventDefault(); // Предотвращение стандартного поведения отправки формы
3. var selects = document.body.querySelectorAll('select[id^="id\_select-"][id$="to"]');
4. var selectOptions = [];
5. selects.forEach(function (select) {
6. var options = Array.from(select.options).map(function (option) {
7. return option.value;
8. });
9. selectOptions.push({
10. 'id\_subject': parseInt(select.id.match(/\d+/)[0]),
11. 'classes': options
12. })
13. });
14. var data = {
15. 'teacher\_id': document.getElementById('teacherId').value,
16. 'array': selectOptions,
17. };
18. var xhr = new XMLHttpRequest();
19. xhr.open('POST', '/teachers/classes\_field\_form/', true);
20. xhr.setRequestHeader('Content-Type', 'application/json');
21. xhr.onreadystatechange = function () {
22. if (xhr.readyState === 4 && xhr.status === 200) {
23. console.log('AJAX request successful');
24. }
25. };
26. xhr.send(JSON.stringify(data));
27. }

В этой функции мы, помимо стандартного сохранения модели учитель (2), будем отсылать дополнительные данные на сервер. Сначала возьмём все выпадающие списки по регулярному выражению (3). Затем в каждом списке будем брать все и выбранные элементы, а также самого учителя. Далее формируем в json все выбранные элементы (17-20) и передаём на сервер (23-33).

Посмотрим на обработку этих элементов на сервере:

1. @csrf\_exempt
2. def classes\_field\_form(request):
3. if request.method == 'POST':
4. data = json.loads(request.body)
5. subjects\_array = data.get('array')
6. teacher\_id = data.get('teacher\_id')
7. teacher = get\_object\_or\_404(Teacher, id=teacher\_id)
8. old\_objects = TeacherSubjectClass.objects.filter(teacher=teacher)
9. new\_objects = []
10. for subject in subjects\_array:
11. for class\_id in subject['classes']:
12. new\_objects.append(TeacherSubjectClass(
13. teacher=teacher,
14. subject=get\_object\_or\_404(Discipline, id=subject['id\_subject']),
15. \_class=get\_object\_or\_404(Class, id=class\_id),
16. ))
17. deleted\_objects = [obj.id for obj in old\_objects if obj not in new\_objects]
18. added\_objects = [obj for obj in new\_objects if obj not in old\_objects]
19. all\_objects = TeacherSubjectClass.objects.all()
20. for old\_obj in all\_objects:
21. for add\_abj in added\_objects:
22. if old\_obj.\_class == add\_abj.\_class and old\_obj.subject == add\_abj.subject:
23. deleted\_objects.append(old\_obj.id)
24. TeacherSubjectClass.objects.filter(id\_\_in=deleted\_objects).delete()
25. TeacherSubjectClass.objects.bulk\_create(added\_objects)
26. return HttpResponse('ok')

Сначала отключим проверку межсайтовый запросов 1. Затем возьмём все элементы из тела запроса 4. Извлечём teacher и классы-предмет 5-7. Возьмём все классы, в которых вёл учитель. Затем возьмём новые данные 11-17. Сравним, какие предметы стали новыми (их нужно будет добавить) 20. И узнаем, какие следует удалить 19. Затем среди добавленных классов проверим, что в них никакой другой учитель не ведёт 22 -26, все существующие записи нужно будет удалить 27. Проведём процедуры массового удаления и массовой вставки 29-30.

Аналогичные части приложений будем реализовывать вышеописанным образом. Скрины работы приводятся в приложении E.

# Заключение

Прохождение практики началось с ознакомления со структурой подразделения и организацией производственно-технологического процесса.

Для разработки прототипа приложения на первом этапе были рассмотрены и изучены аналоги существующих решений.

В процессе создания проекта были установлены необходимые библиотеки для работы приложения. Затем был разработан фронтенд, начиная с создания каркаса приложения и заготовок для различных страниц. Далее произошло создание моделей данных и выполнение миграций для хранения информации в базе данных. Оформление стилей и визуальное оформление приложения позволили создать привлекательный дизайн. Настройка админ-панели обеспечила возможность управления данными. Создание форм и валидация полей ввода пользователя помогли обеспечить корректный ввод данных. Для заполнения информации в приложении были собраны данные о предметах, преподавателях и аудиториях.

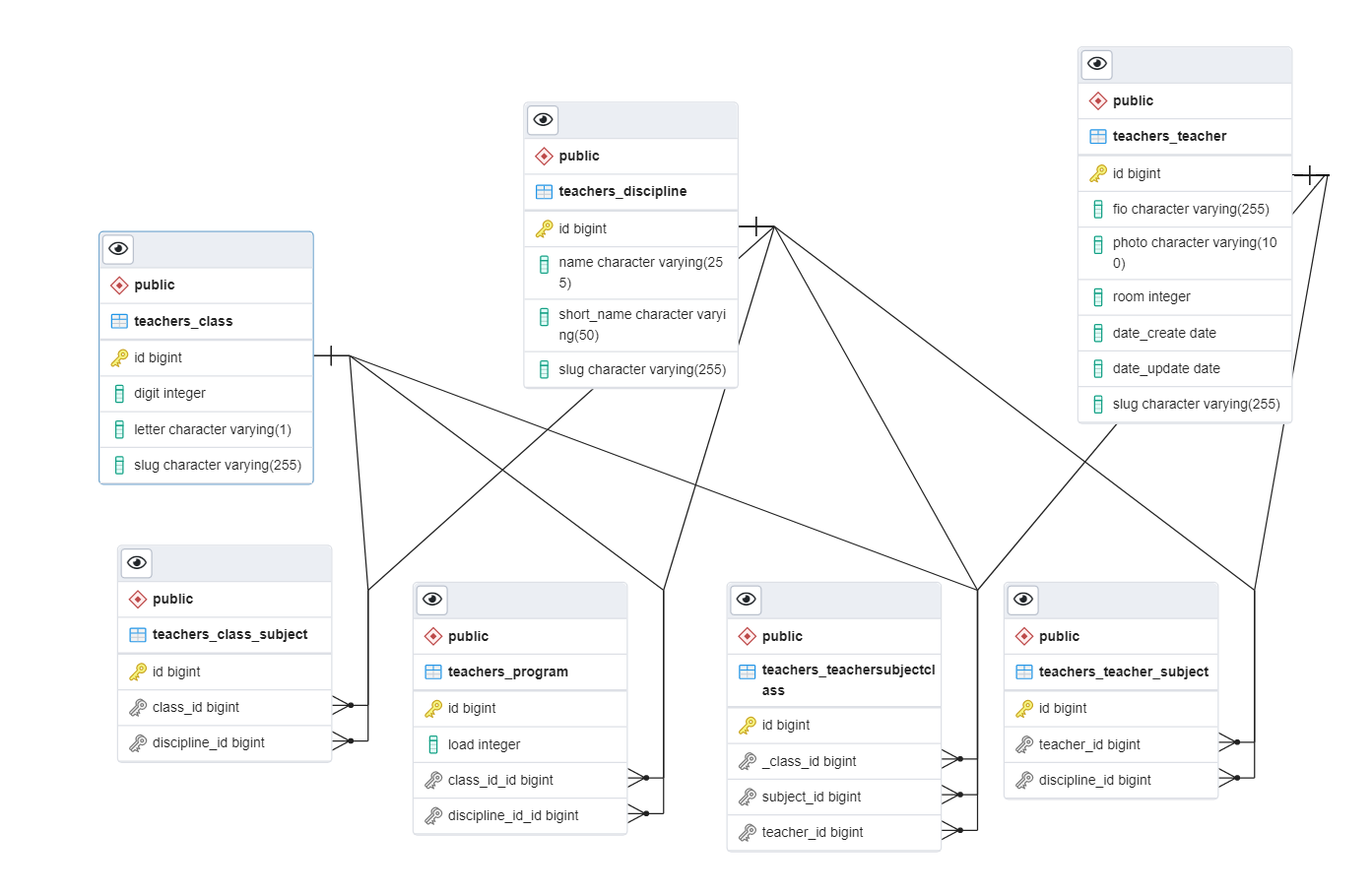
Таким образом, в результате прохождения всех этапов была создана заготовка приложения «Расписание», предоставляющее пользователям удобный и функциональный инструмент для сбора информации и просмотра расписания учебных занятий.

# Список источников

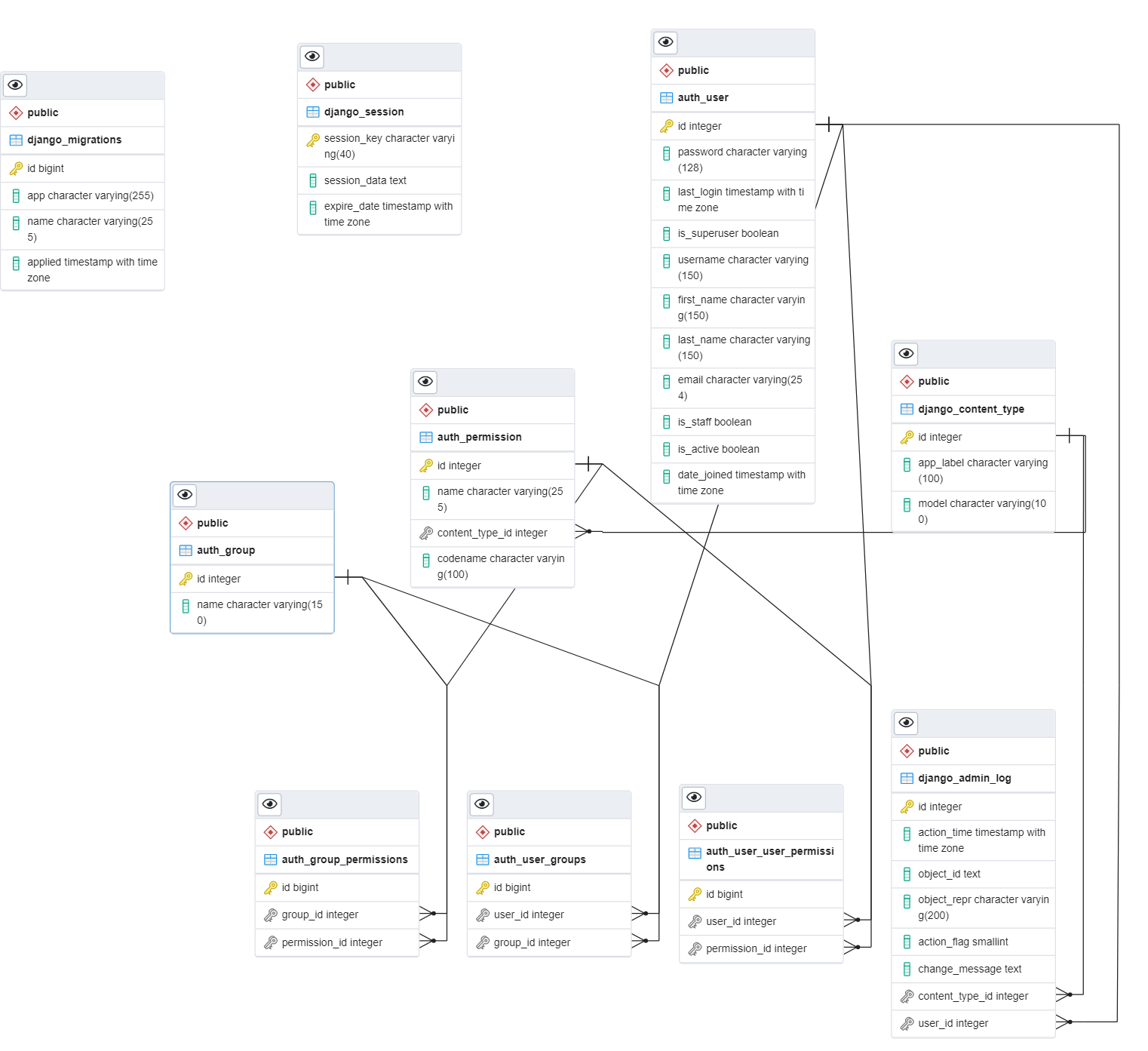
1. Студенческое конструкторское бюро «Робототехника» (СКБ «Робототехника»). URL: <https://www.miet.ru/structure/s/2786/e/94105/35> (дата обращения: 27 апреля 2024) .
2. БИТ.Расписание Лайт. URL: <https://novosibirsk.1cbit.ru/1csoft/bit-raspisanie-lite/?utm_source=yandex&utm_medium=cpc&utm_campaign=marina&utm_content=10589223974&utm_term=программа%20для%20составления%20учебного%20расписания&utm_referrer=https%3A%2F%2Fyandex.ru%2F> (дата обращения: 30 апреля 2024) .
3. Экспресс-расписание. URL: <https://pbprog.ru/catalog/timetable> (дата обращения: 25 апреля 2024).
4. Хронограф для расписания уроков. URL: <https://uogornogo.obr.sakha.gov.ru/ikt-v-obrazovanii/hronograf-dlja-raspisanija-urokov> (дата обращения: 22 апреля 2024).
5. Ника Софт - Ника Колледж. URL: <https://nikasoft.ru/nika-college/> (дата обращения: 03 мая 2024).
6. ASC Timetables. URL: <https://www.asctimetables.com/> (дата обращения: 01 мая 2024).
7. Уроки по Django. URL: <https://proproprogs.ru/django4> (дата обращения: 28 апреля 2024).
8. Документация по Django. URL: <https://docs.djangoproject.com/en/5.0/> (дата обращения: 20 апреля 2024).
9. Django | Отношения Многие ко многим. URL: <https://metanit.com/python/django/5.7.php> (дата обращения: 5 мая 2024)
10. Приложение «АВТОРасписание» | Ютел. URL: <http://utelksp.ru/programs/prilozhenie-avtoraspisanie> (дата обращения: 7 мая 2024)
11. <https://learn.javascript.ru/ajax-xmlhttprequest>
12. <https://www.webpupil.ru/javascript_view.php?id=243>
13. <https://learn.javascript.ru/fetch>
14. <https://metanit.com/python/django/5.14.php>
15. <https://learn.javascript.ru/formdata>
16. <https://habr.com/ru/articles/128704/>
17. <https://mathmod.asu.edu.ru/images/File/ebooks/GAfinal.pdf>
18. <https://science-start.ru/ru/article/view?id=1065> (Ковалева Е.А ГЕНЕТИЧЕСКИЙ АЛГОРИТМ В ЗАДАЧЕ СОСТАВЛЕНИЯ РАСПИСАНИЯ // Старт в науке. – 2018. – № 5-1. ;  
    URL: https://science-start.ru/ru/article/view?id=1065 (дата обращения: 02.06.2024).)
19. <https://www.geeksforgeeks.org/comparison-of-fastapi-with-django-and-flask/>
20. <https://habr.com/ru/articles/733942/>
21. <https://habr.com/ru/articles/578744/>
22. <https://docs.djangoproject.com/en/5.0/ref/templates/language/>

# Приложение

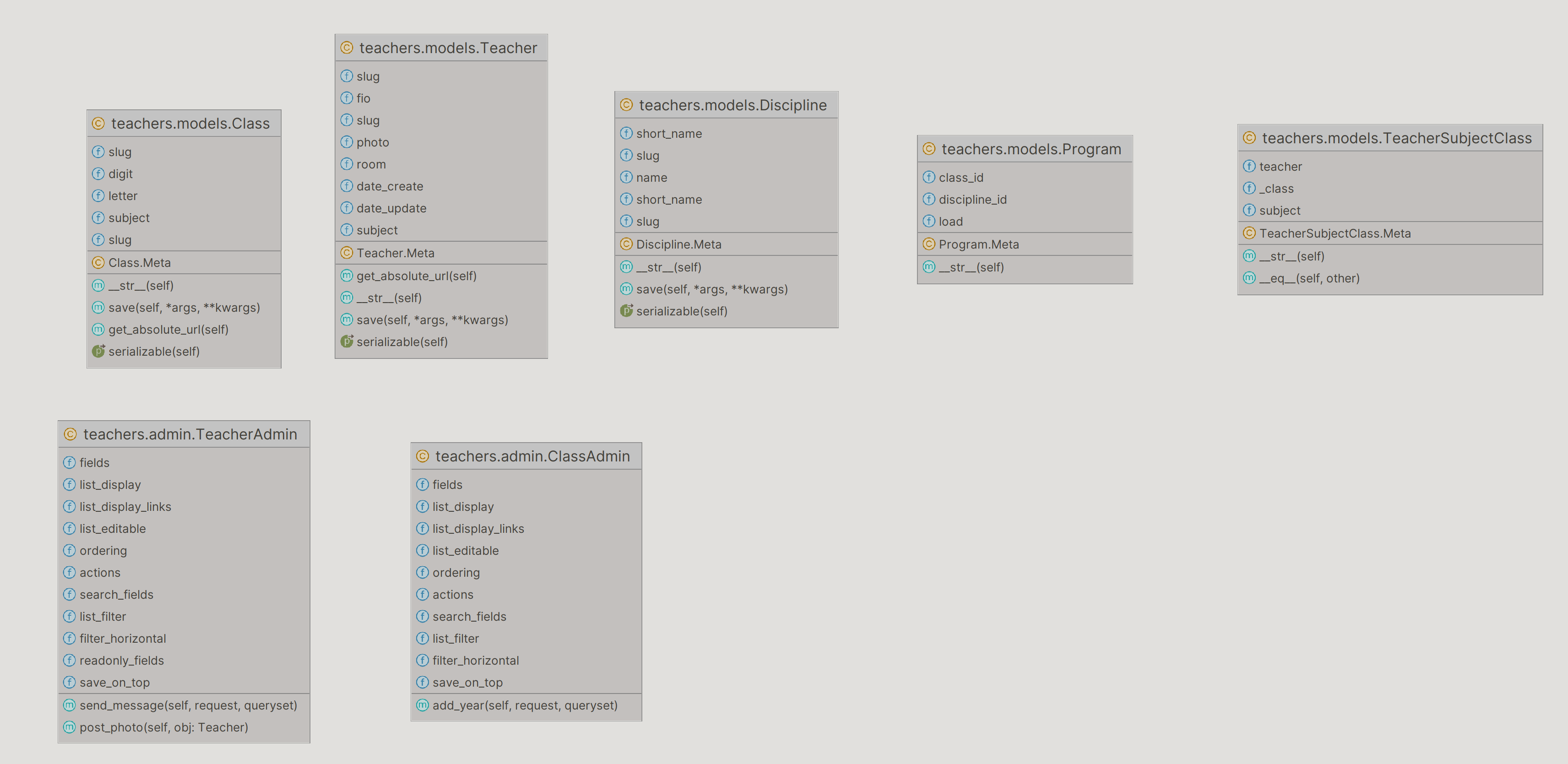
1. Диаграмма сущность-связь БД (Entity-Relationship) для собственных моделей



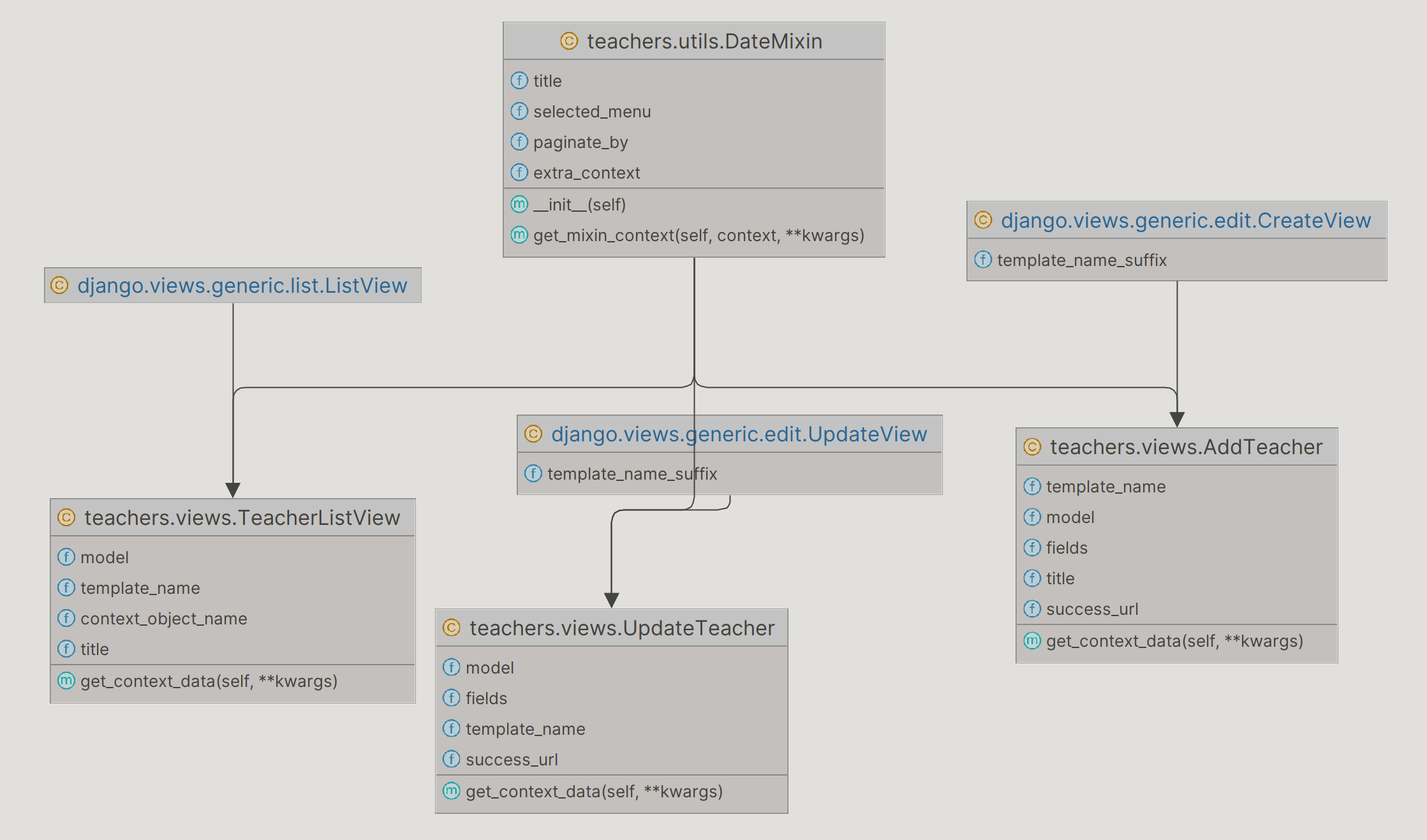
1. Диаграмма сущность-связь автоматически сформированная



1. Диаграмма классов приложения. Классы моделей



1. Диаграмма классов приложения. Классы представлений



1. Скрины работы программы

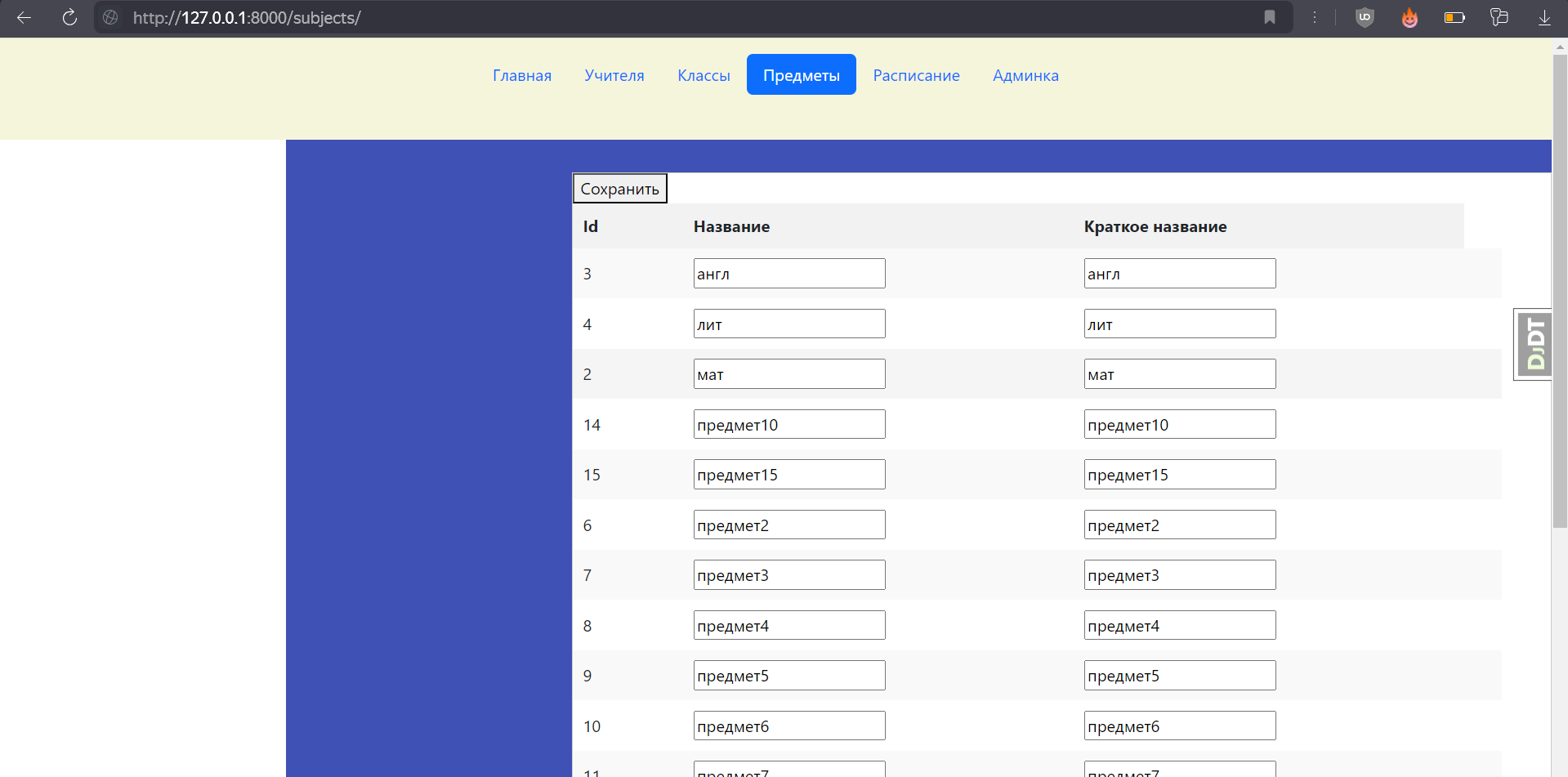


Рисунок 0.1. Список предметов. Мультиформы

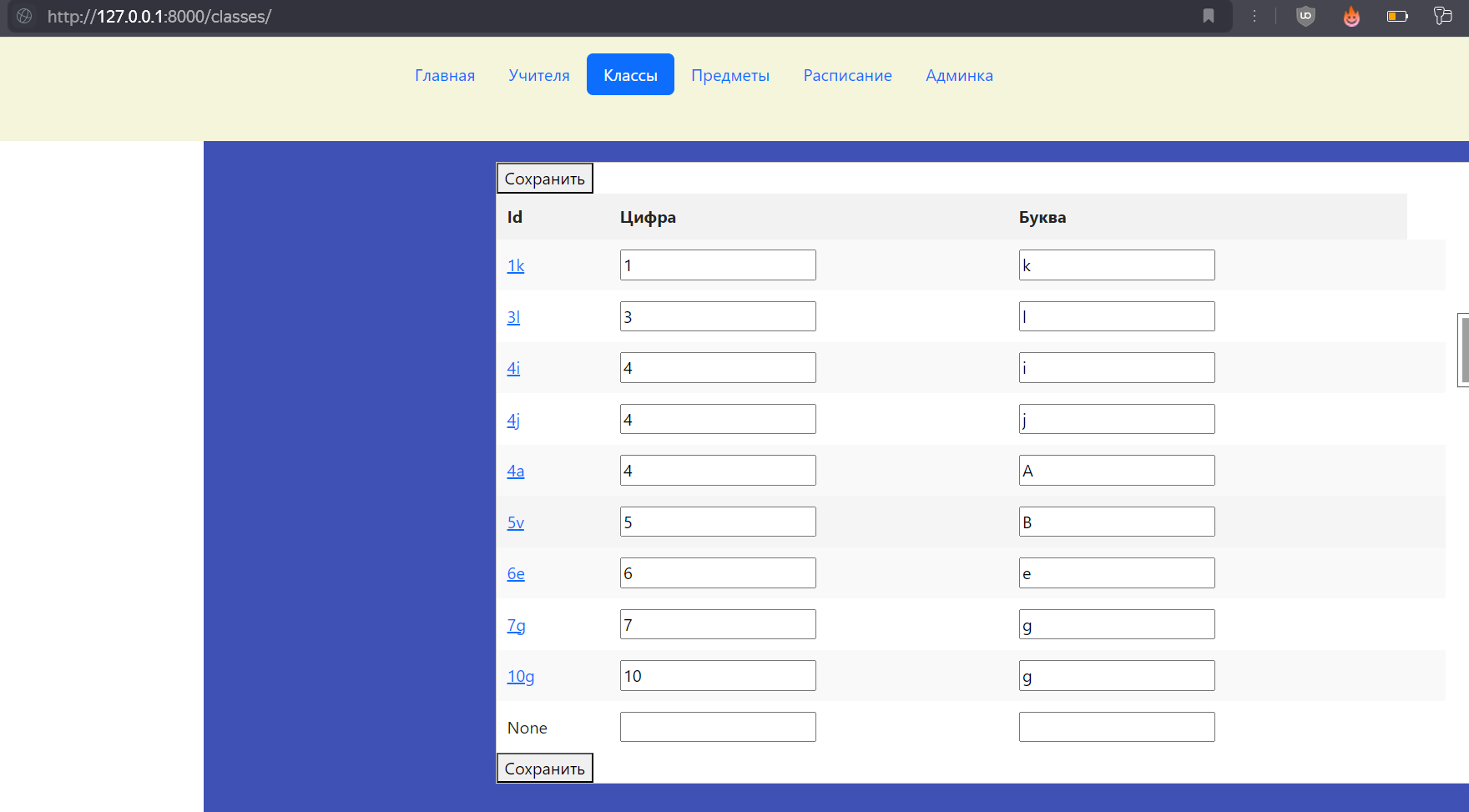


Рисунок 0.2. Список классов. Мультиформы

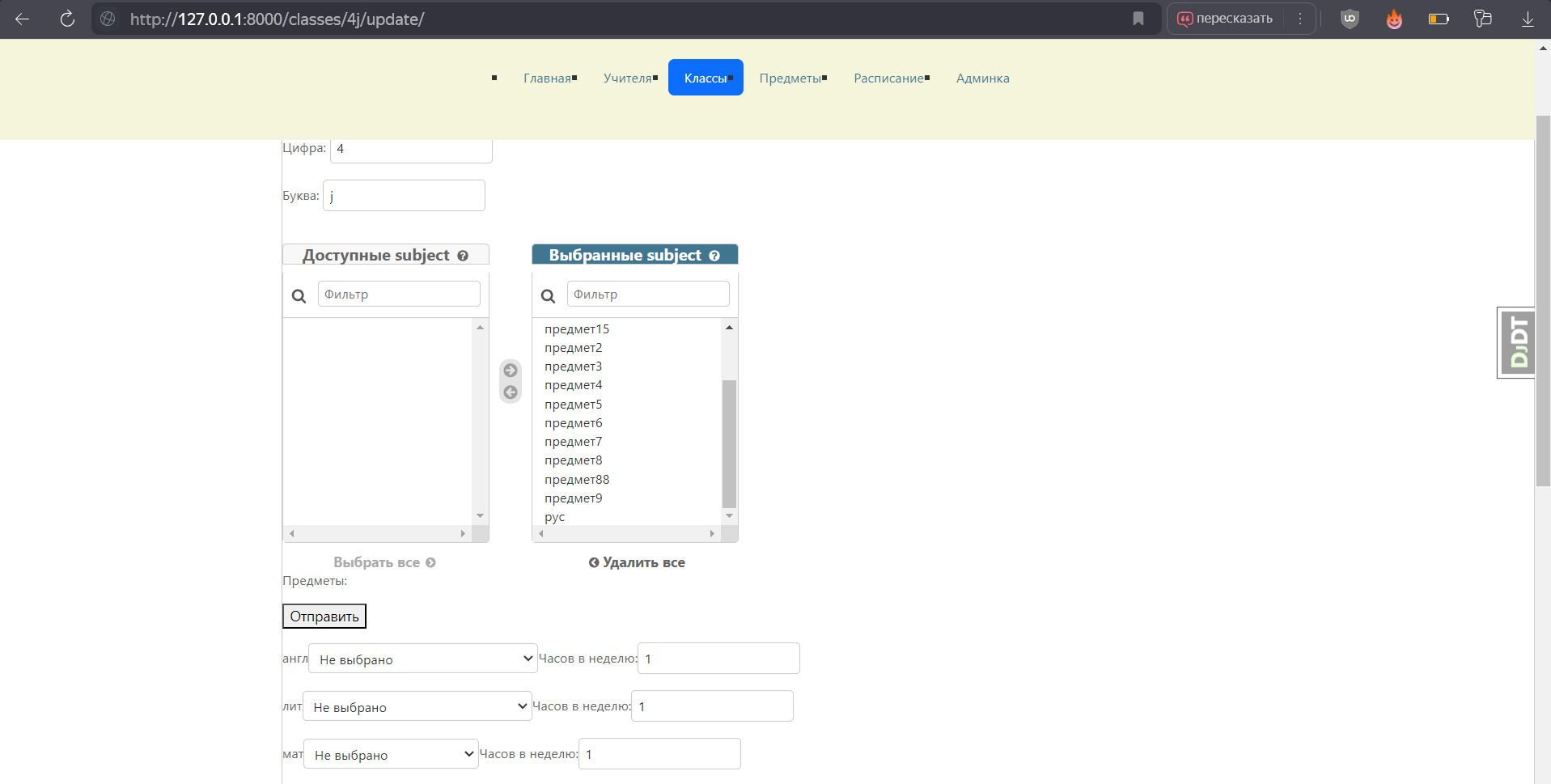


Рисунок 0.3. Изменение класс

1. @startuml
2. left to right direction
3. skinparam actor {
4. BackgroundColor LightBlue
5. BorderColor DarkBlue
6. }
7. actor Teacher
8. actor (Student or Parent)
9. actor Dispatcher
10. actor Administrator
11. Teacher --> (Send Wishes to Administrator)
12. Teacher --> (View Work Schedule)
13. Administrator --> (Send Message)
14. Administrator --> (Send Data)
15. Administrator --> (Student or Parent): Transfer
16. Administrator --> (Student or Parent): Demote
17. Administrator --> Teacher: Dismiss
18. Administrator --> Teacher: Hire
19. (Send Schedule) --> (Send Data) <<extends>>
20. (Send Workload) --> (Send Data) <<extends>>
21. (Send Establishment Schedule) --> (Send Data) <<extends>>
22. (Send Personal Data) --> (Send Data) <<extends>>
23. (Send Wishes) --> (Send Data) <<extends>>
24. (Send Program) --> (Send Data) <<extends>>
25. Dispatcher --> Teacher :(Send Schedule)
26. Dispatcher --> (Student or Parent) :(Send Schedule)
27. Dispatcher --> Administrator :(Send Schedule)
28. Student -->Schedule: View
29. Teacher -->Schedule: View
30. @enduml