МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |  |  |
| --- | --- | --- | --- |
| Кафедра | *Вычислительной техники* | | |
|  | (полное название кафедры) | | |
|  |  | Утверждаю | |
| **Зав. кафедрой** | *Якименко А.А.* |
|  | |
| (подпись, инициалы, фамилия) | |
| «16» марта 2024 г. | |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

|  |  |
| --- | --- |
| *Еремеева Виктория Владимировна* | |
| (фамилия, имя, отчество студента – автора работы) | |
| *Разработка приложения "Расписание"* | |
| (тема работы) | |
| *Факультет автоматики и вычислительной техники* | |
| (полное название факультета) | |
| Направление подготовки | *Информатика и вычислительная техника 09.03.01* |
|  | (код и наименование направления подготовки бакалавра) |
| *(очная форма)* | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель**  **от НГТУ** |  |  |  | **Автор выпускной квалификационной работы** |
| *Осипенко И.В.* |  |  |  | *Еремеева В.В.* |
| (фамилия, имя, отчество) |  |  |  | (фамилия, имя, отчество) |
| *Старший преподаватель* |  |  |  | *АВТФ, АВТ-009* |
| (ученая степень, ученое звание) |  |  |  | (факультет, группа) |
|  |  |  |  |  |
| (подпись, дата) |  |  |  | (подпись, дата) |

|  |
| --- |
|  |

Новосибирск 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |  |  |
| --- | --- | --- | --- |
| Кафедра | *Вычислительной техники* | | |
|  | (полное название кафедры) | | |
|  |  | УТВЕРЖДАЮ | |
| Зав. кафедрой | *Якименко А.А.* |
| (фамилия, имя, отчество) | |
|  |  |
|  |  |
|  | (подпись, дата) |

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

|  |  |  |  |
| --- | --- | --- | --- |
| студенту | *Еремеевой Виктории Владимировне* | | |
|  | (фамилия, имя, отчество) | | |
| Направление подготовки | | *Информатика и вычислительная техника 09.03.01* | |
|  | | (код и наименование направления подготовки бакалавра) | |
| *Факультет автоматики и вычислительной техники* | | | |
| (полное название факультета) | | | |
| Тема | *Разработка приложения "Расписание"* | | |
|  | (полное название темы выпускной квалификационной работы бакалавра) | | |
|  | | | |
| Исходные данные (или цель работы) | | | *Целью работы является разработка* |
| *веб-приложения "Расписание" для генерации и визуализации расписания уроков* | | | |
| *в школе* | | | |
| *ПО: Python3, PyCharm, PostgreSQL, Docker, Django* | | | |
| Структурные части работы | | |  |
| *Введение* | | | |
| 1. *Анализ существующих решений* | | | |
| 1. *Особенности разработки приложения* | | | |
| 1. *Особенности реализации* | | | |
| *Заключение* | | | |

Задание согласовано и принято к исполнению.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель**  **от НГТУ** |  |  |  | **Студент** |
| *Осипенко И.В.* |  |  |  | *Еремеева В.В.* |
| (фамилия, имя, отчество) |  |  |  | (фамилия, имя, отчество) |
| *Старший преподаватель* |  |  |  | *АВТФ, АВТ-009* |
| (ученая степень, ученое звание) |  |  |  | (факультет, группа) |
|  |  |  |  |  |
| (подпись, дата) |  |  |  | (подпись, дата) |

|  |  |  |  |
| --- | --- | --- | --- |
| Тема утверждена | приказом по НГТУ № | ㅤㅤㅤㅤ | от «ㅤㅤ» ㅤㅤㅤㅤㅤㅤ 2024 г. |
| изменена | приказом по НГТУ № | ㅤㅤㅤㅤ | от «ㅤㅤ» ㅤㅤㅤㅤㅤㅤ 2024 г. |

| ВКР сдана в ГЭК №ㅤㅤㅤㅤ, тема сверена с данными приказа | |
| --- | --- |
|  |  |
|  | (подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата) |
|  |  |
|  | (фамилия, имя, отчество секретаря государственной  экзаменационной комиссии по защите ВКР) |

**Аннотация**

В пояснительной записке к выпускной квалификационной работе бакалавра содержится 63 страницы, 29 иллюстраций, 4 таблицы и 24 использованных литературных источников.

Целью данной работы является разработка веб-приложения "Расписание" для генерации и визуализации расписания уроков в школе

Проект является веб-приложением, написанным на Python с использованием фреймворка Django. В качестве базы данных используется postgres.

В данной работе будут представлены теоретические материалы о существующих алгоритмах и системах и их применении. Во время практики предполагается:

* рассмотреть конкретные примеры средств для составления расписания в учреждениях образования,
* провести исследование существующих алгоритмов и методов формирования расписания,
* выполнить сравнительный анализ эффективности и выбор возможных проектных решений.

В практической части будут описаны возможные технологии реализации, апробированы отдельные технические средства.

В заключение отчёта по практике будут вынесены выводы о проделанной работе и перспективы использования её результатов в предстоящей ВКР по соответствующей теме. В приложении к отчёту по практике будет проложен проект технического задания по выбранной теме и кандидатуры руководителя бакалаврской ВКР.

Содержание

[Глоссарий 7](#_Toc168573872)

[Введение 9](#_Toc168573873)

[1 Анализ существующих решений 11](#_Toc168573874)

[1.1 Существующие программные аналоги 11](#_Toc168573875)

[1.2 Раскраска графов 14](#_Toc168573876)

[1.3 Генетические алгоритмы 17](#_Toc168573877)

[2 Особенности разработки 20](#_Toc168573878)

[2.1 Общие архитектурные положения 21](#_Toc168573879)

[2.2 Структура проекта 22](#_Toc168573880)

[2.3 Схема взаимодействия 23](#_Toc168573881)

[2.4 Модель прецедентов 24](#_Toc168573882)

[2.5 Диаграмма классов 24](#_Toc168573883)

[3 Особенности реализации 26](#_Toc168573884)

[3.1 Реализация проекта 26](#_Toc168573885)

[3.2 Развёртка в контейнере Docker 42](#_Toc168573886)

[3.3 Руководство пользователя 44](#_Toc168573887)

[Заключение 52](#_Toc168573888)

[Список литературы 53](#_Toc168573889)

[Приложение А. Сводная таблица аналогов 56](#_Toc168573890)

[Приложение Б. Схема Rest API 57](#_Toc168573891)

[Приложение В. Код создания модели прецедентов 59](#_Toc168573892)

[Приложение Г. Диаграмма прецедентов 61](#_Toc168573893)

[Приложение Д. Шкала трудности учебных предметов 62](#_Toc168573894)

[Приложение Е. Диаграмма сущность-связь автоматически сформированная 63](#_Toc168573895)

[Приложение Ж. Код начального создания объектов 64](#_Toc168573896)

# Глоссарий

желательно таблицей и нумерация

обрезать пустоту главную страницу

размер на плашках одинаковый

картинку Жени убрать

наполнить home

Декартово произведение

Кроссинговер

Окна

Фреймворк

Мутация

Django

MVC

MTV

CMS

API

UI

UX

JSON

XML

AJAX

Middleware

OpenAPI

Планарные графы

Петли

Хроматический многочлен

dnf

Декоратор

SQL-инъекции

Транзакции

Миксин

# Введение

В современном мире информация играет ключевую роль в эффективности любой организации. Растёт количество информации. 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.

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** работы является разработка веб-приложения "Расписание" для генерации и визуализации расписания уроков в школе. Для этого я поставила перед собой следующие **задачи**:

* Ознакомиться с организационной структурой МБОУ СОШ №8, ознакомиться с организацией образовательного и технологического процесса.
* Провести анализ существующих решений.
* Разработать архитектуру приложения; создать модели; написать шаблоны страниц; оформить стили проекта.
* Определить функционала и бизнес-логику.
* Произвести тестирование интерфейса и функциональное тестирование.
* Разработать руководство пользователя.
* Создать Dockerfile для развёртки приложения в контейнере.

# Анализ существующих решений

## Существующие программные аналоги

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. Сводную таблицу можно посмотреть в приложении.

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.

Для генерации расписаний обычно используются алгоритмы полного перебора, которые перебирают все возможные варианты расписаний и выбирают оптимальный вариант. Однако, при большом количестве вариантов расписаний, полный перебор становится вычислительно затратным и неэффективным.

Трудно сказать, какие алгоритмы используются в аналогах моего приложения генерации расписания из-за коммерческой тайны и высокой цены на продукты.

## Раскраска графов

Раскраску графов применяют для многих типов задач, таких как раскраска географических карт, оптимизация каких-то процессов, имеющие общие ресурсы, в том числе формирование расписания. При раскраске графов важно, чтобы никакие две смежные вершины не были раскрашены в один и тот же цвет. При этом количество цветов должно быть минимальным.

Рассмотрим применение теории графов для решения задачи расписания на примере. Пусть нам нужно составить расписание для двух классов на понедельник. Для этого способа нужно распределить предметы по дням по классам. Приведём такой пример в Таблице 1:

Таблица 1.1 – Набор предметов для составления расписания

|  |  |  |  |
| --- | --- | --- | --- |
| № п.п |  | 5А | 5Б |
| 1 | Русский | + | + |
| 2 | Математика | + | + |
| 3 | Английский |  | + |
| 4 | Физкультура | + | + |
| 5 | ОБЖ | + | + |
| 6 | Биология | + |  |
| 7 | Литература |  | + |
| 8 | Обществознание | + |  |

Предметы должны распределяться исходя из пожеланий учителей по дням работы. Также нужно учитывать время для выполнения домашнего задания в классе. Можно поставить в один день целую параллель классов, например для учителя география в целях экономии времени учителя для подготовки к урокам на следующий день.

Далее необходимо составить граф. В целях экономии места на Рисунке 1.1будем отмечать предметы порядковым номером из Таблицы 1.

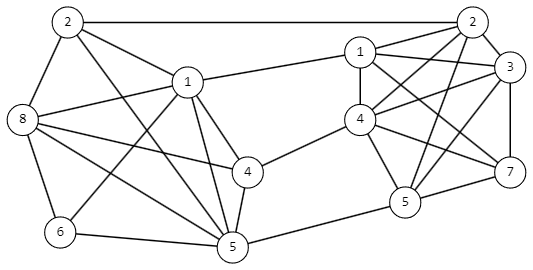


Рисунок 1.1 — Граф, составленный по таблице 1

На этом рисунке мы можем заметить 2 многоугольника вершин, расположенные в левой и правой части картинки. Они обозначают 2 разных класса: 5А и 5Б. Цифры в многоугольниках обозначают предметы в классах в понедельник.

Далее мы соединяем вершины рёбрами в местах пересечений. Внутри многоугольников все цифры соединены друг с другом. Это означает, что мы не можем вести две дисциплины в классе одновременно. Это невозможно для детей. Но мы можем, не соединять предметы, на которых класс делится на подгруппы. Например, технология, английский, физкультура.

Далее соединяются вершины между классами. Это означает, что мы не хотим, чтобы учитель преподавал в двух классах одновременно. Можно не соединять вершины, если их ведут разные учителя. Также дополнительно можно соединить русский и литературу, например, в разных классах, если их должен вести один и тот же учитель. Но мы этого делать не будем, потому что примем, что в русский и литературу в школе ведут разные учителя.

Затем происходит раскраска полученного графа. Она представлена на Рисунке 1.2:



Рисунок 1.2 —Раскрашенный граф

Граф можно раскрасить разными способами, причём с помощью всего четырёх цветов. Но на данном примере целесообразнее пользоваться шестью цветами по числу уроков в день. Введём обозначения, красный – первый урок, оранжевый – второй и далее по цветам радуги. Таким образом покажем получившееся расписание в Таблице 2:

Таблица 1.2 – Получившееся расписание

|  |  |  |
| --- | --- | --- |
|  | 5А | 5Б |
| 1 | Математика | Физкультура |
| 2 | Русский | ОБЖ |
| 3 | ОБЖ | Математика |
| 4 | Физкультура | Английский |
| 5 | Биология | Литература |
| 6 | Обществознание | Русский |

Полученное расписание не имеет наложений уроков у классов и классов у учителей, что соответствует требованиям. Рекомендации о желательном порядковом номере урока также следует учитывать. Для этого выбираются те варианты, где используются соответствующие цвета для урока. Либо же можно выставить цвета для уроков, имеющих константные номера.

Можно воспользоваться онлайн-сервисом по раскраске графов <https://graphonline.ru/> .

При составлении настоящего расписания придётся строить гораздо большие графы, что является очень трудоёмким и долгим процессом.

## Генетические алгоритмы

Также для составления расписания можно воспользоваться генетическими алгоритмами. Генетические алгоритмы используются для решения задач оптимизации. Они являются так называемой оптимизацией полного перебора. Алгоритм идентичен настоящему эволюционному алгоритму выбора клеток в природе.

Его суть заключается в комбинации элементов множества. Это называется Декартовым произведением. В нашем случае умножать мы будем множества учителей на множество классов, дней недели и уроков.

Но с тем условием, что участвуют не все варианты, а с корректировкой на каждом шаге. Пройдёмся по шагам:

* выбор начальных наборов;
* оценивание на жизнеспособность;
* селекция;
* скрещивание;
* мутация.

На первом этапе случайным образом составляется набросок расписания. Наброски должны быть в нескольких экземплярах и их количество зависит от размера школы. Но чем их будет больше и чем сильнее они будут отличаться друг от друга, тем лучше.

На следующем шаге нужно выбрать функцию качества. Она должна быть в численном виде, чтобы можно было оценить её значение. Каждый критерий должен оцениваться. Например, можно подсчитать число окон у учеников. Также можно подсчитать число наложений у учителей и учеников. Соответствия рекомендуемым нормам СанПин о динамике трудоспособности в день и неделю.

При селекции мы выбираем какой-то начальный набор, пусть даже он нам не совсем подходит. Таких наборов должно быть несколько. Далее выясняется их пригодность в численном виде. Для этого используется функция качества.

Функция качества (fitness function) в генетических алгоритмах определяет, насколько хорошо решение соответствует поставленной задаче. Она оценивает каждое индивидуальное решение в популяции и помогает определить, какие решения следует сохранить и использовать для создания новых поколений. Чем выше значение функции качества, тем более пригодным считается решение. Функция качества играет ключевую роль в оптимизации и выборе лучших расписаний при использовании генетических алгоритмов.

Каждый полученный набор подставляется в функцию и вычисляется значение. Затем выбираются наиболее подходящие элементы и начинается уже скрещивание их.

Пусть, например, функция качества задаётся уравнением типа

f­­(a,b,c,d)=5a+2b+7c+d.

Пусть для идеального расписания сумма этих элементов должна быть равна 30.

Теперь представим, что у потомков значения являются следующими:

Таблица 1.3 – Начальные наборы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | f­­(a,b,c,d) |
| Н1 | 1 | 5 | 3 | 7 | 43 |
| Н2 | 3 | 2 | 6 | 1 | 62 |
| Н3 | 4 | 6 | 2 | 2 | 48 |
| Н4 | 2 | 4 | 5 | 6 | 59 |
| Н5 | 5 | 1 | 1 | 3 | 37 |

Теперь вычислим пригодность каждого из этих наборов (9):

Таблица 1.4 – Расчёт пригодности

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | a | b | c | d | f­­(a,b,c,d) | 1/|f-30| | (7)/сумму(7) | (8)×100% |
| Н1 | 1 | 5 | 3 | 4 | 40 | 0,1 | 0,27 | 27,46155 |
| Н2 | 3 | 2 | 6 | 1 | 62 | 0,03125 | 0,09 | 8,581736 |
| Н3 | 4 | 6 | 2 | 2 | 48 | 0,055556 | 0,15 | 15,25642 |
| Н4 | 2 | 4 | 5 | 6 | 59 | 0,034483 | 0,09 | 9,469501 |
| Н5 | 5 | 1 | 1 | 3 | 37 | 0,142857 | 0,39 | 39,23079 |
|  |  |  |  |  |  | 0,364145 |  |  |

Для этого мы будем выполнять действия в заголовках таблицы. Стоит отметить, что внизу столбца 7 указана сумма значений.

Далее мы произвольным образом скрещиваем полученные наборы. Имеет смысл перебрать больше вариантов со скрещиванием набора 1 и 5.

Но нельзя забывать и про другие наборы, которые также должны участвовать, чтобы процесс не остановился в точке локального максимума. Потомки не будут участвовать в селекции только если их выживаемость (9) станет равна 0.

Чем больше изначальных наборов, тем быстрее мы будем приближаться к желаемому значению.

Не обязательно целиком брать значения каждого из двух родителей. В процессе можно производить мутации. Например, мы можем поменять один предмет на другой, которого нет ни в одном из родителей. Или поменять классы кабинетами при возможности.

# Особенности разработки

Составление расписания является мультиоптимизационной задачей. Требуется учитывать следующие требования:

* отсутствие окон у учеников;
* минимизация числа окон у учителей;
* отсутствие «наслоений» предметов у детей и классов у учителей;
* пожелания учителей;
* соответствие графику трудоспособности детей для дня и недели: учёт времени начала учёбы, окончания, отдыха;
* недопущение ситуаций, когда предметы из одной группы (например, технические, математика и физика) следуют друг за другом;
* закрепление кабинетов;
* равномерное распределение предметов в течение дня и недели.

Конкретные условия качества составления прописаны в СанПин. В Приложении E приведён пример шкалы трудности предметов для 5-9 классов. В неё по строкам размещены предметы, а по столбцам трудность этой дисциплины в каждом классе.

На рисунке ниже (Рисунок 2.1) приведён пример диаграммы работоспособности в баллах по дням для шестидневной недели.

Рисунок 2.1 — Шкала трудности предметов

## Общие архитектурные положения

Проект будет разрабатываться с помощью фреймворка Django. Хотя этот фреймворк и имеет свои минусы (синхронность, встроенные запросы в БД), ими можно пренебречь. В Django используется шаблон программирования MTV, что очень похоже на модель MVC.

Модуль модели часто представлен в виде таблиц БД. Код моделей пишется в файлы models. Django использует ORM в качестве абстракции и переноса из классов моделей в БД. ORM в Django позволяет работать с базой данных, с помощью синтаксиса Python и параметризованных запросов, что делает взаимодействие с базой данных быстрее и удобнее, защищает от инъекций SQL.

Модуль templates представлен в виде html-файлов. Часто в них находится специфический код, похожий на язык Jinja. В Django этот язык называется Django template language. Шаблоны можно вкладывать друг в друга.

Модуль View отвечают за логику маршрутизации, заголовки, post, get. Являются связующим звеном между model и templates, отображают ответы и обрабатывают запросы.

## Структура проекта

Поговорим о стандартной файловой структуре. Она представлена на рисунке 2.2:

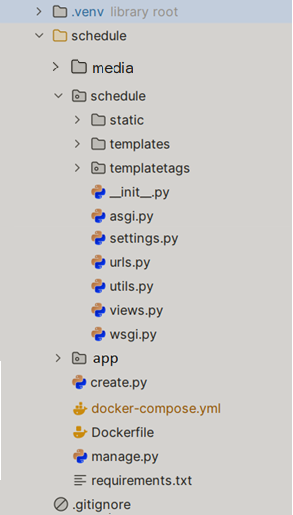


Рисунок 2.2 — Файловая структура проекта

Папка **.venv** необходима для создания виртуального окружения. В неё устанавливаются библиотеки pip, чтобы не занимать основное пространство компьютера.

Далее раскрывается главное приложение проекта **schedule**. В нём содержится файл \_\_**init\_\_.py**. Это служебный файл, который нужен для инициализации пакета. Далее идёт файл **asgi.py**. В нём указаны стандартные настройки для возможности асинхронной работы. Файл **settings.py** содержит настройки конфигурации приложения. Например, там указываются параметры подключения к БД. В файле **urls.py** заданы соответствия html-запросов и вызываемых функций Django. **utils.py** не является обязательным, но в нём принято хранить миксины. В файле **views** описаны функции обработки запросов. **Wsgi** является стандартом взаимодействия, промежуточным слоем между запросом в браузере и нашим приложением. Подробнее мы можем рассмотреть это в следующем разделе на Рисунке 2.3.

Далее рассмотри папки внутри главного приложения. В templates лежат html-шаблоны. В папке static хранятся подпапки со стилями, картинками и js-скриптами. Templatetags является необязательной папкой и содержит файл работы с меню.

На уровень выше лежит папка media, куда загружаются файлы от пользователей. Файлы **Docker** служат для развёртки приложения в контейнере. Внутри **requirements** находится список необходимых библиотек и их версии. **Manage.py** является главным файлом приложения. Он распознаёт команды терминала и выполняет необходимые функции администрирования проекта.

## Схема взаимодействия

На рисунке ниже (Рисунок 2.3) представлена стандартная схема взаимодействия django-приложения. Наше приложение целиком соответствует этой диаграмме.

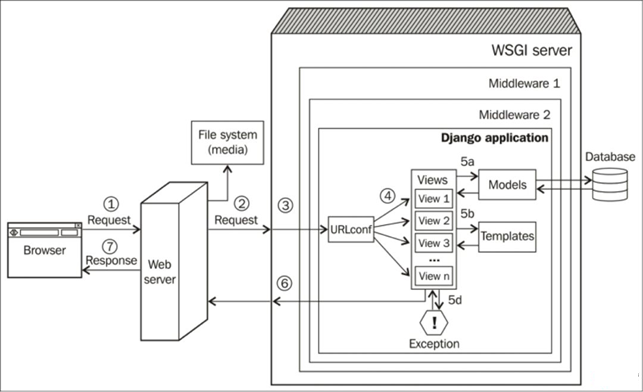


Рисунок 2.3 — Схема взаимодействия

При запросе в веб-браузере происходит запрос на сервер **wsgi**, затем в **Django**, которое обёрнуто в **middleware**. Файл **urls** ищет функции в файле **views**. Адрес сопоставляется с известными адресами, которые наше приложение умеет обрабатывать. В случае, если приложение не знает такого адреса, то отправит ответ вида **exeption** 404. **Views** оперирует **БД** (при необходимости), отрисовывает шаблоны из файлов **templates**, передаёт туда данные. **Templates** могут обращаться к статическим файлам типа стилей и скриптов. Таким образом ответ из views отправляется обратно в браузер через **middleware** и **wsgi**.

## Модель прецедентов

Код для модели прецедентов был написан на языке PlantUML и представлен в Приложении C. Он является очень простым, и я считаю, что не составит труда в нём разобраться самостоятельно. Диаграмма приведена на Рисунке 2.3.

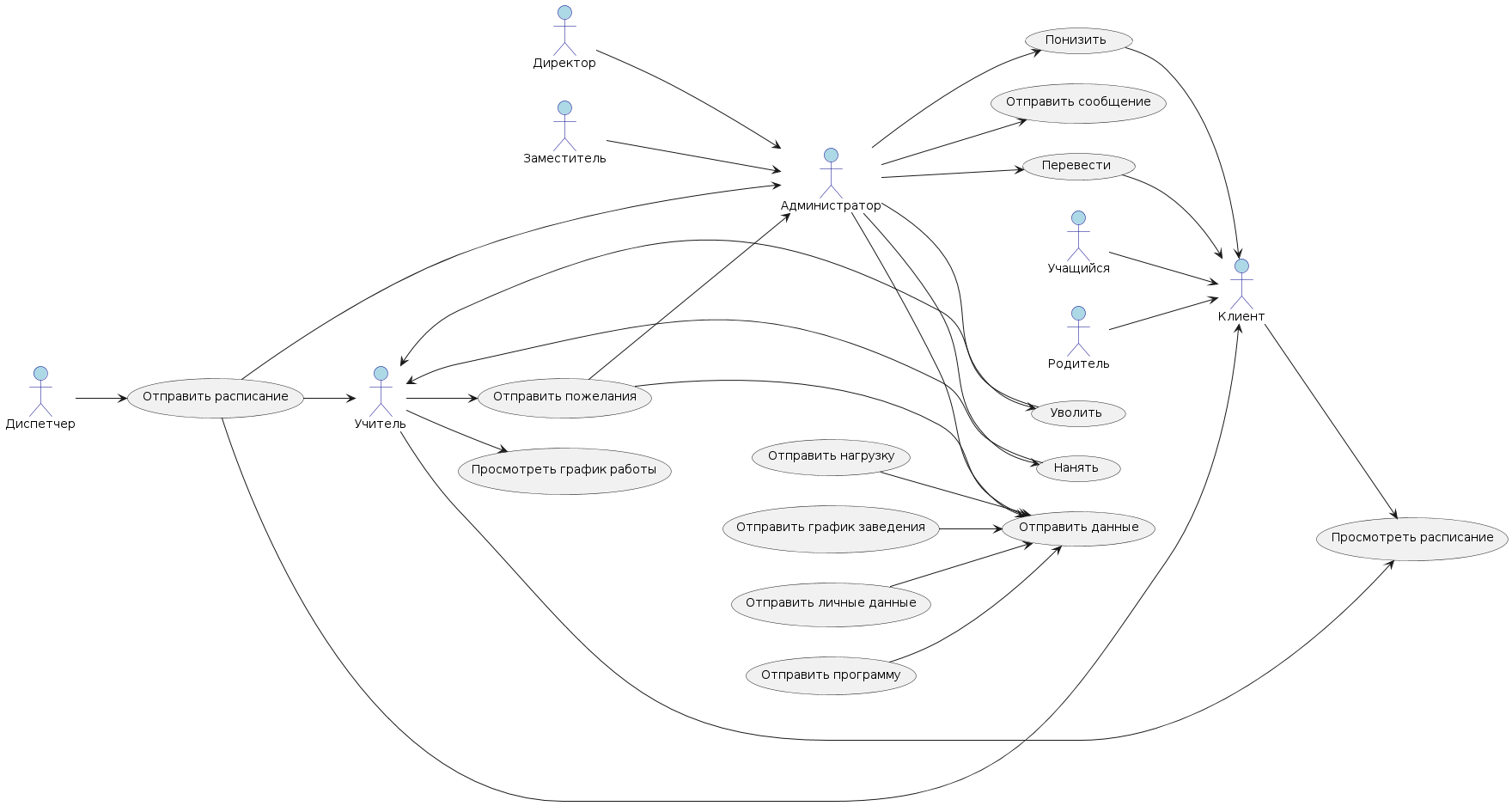


Рисунок 2.4 — Диаграмма прецедентов

## Диаграмма классов

Проект будет состоять из 5 модулей: дисциплины, программы, классы, учителя. Модули перечислены в том порядке, в котором их рекомендуется разрабатывать и в котором удобно работать на сайте.

В качестве дополнительных таблиц нам понадобится таблица **teacher\_discipline**, в которой будет связываться учитель и его специализация в дисциплинах. Ещё понадобится таблица **teacher\_discipline\_class**, в которой будут указываться фактически преподаваемая дисциплина учителя и класс, в котором он её ведёт.

Листинг 2.1

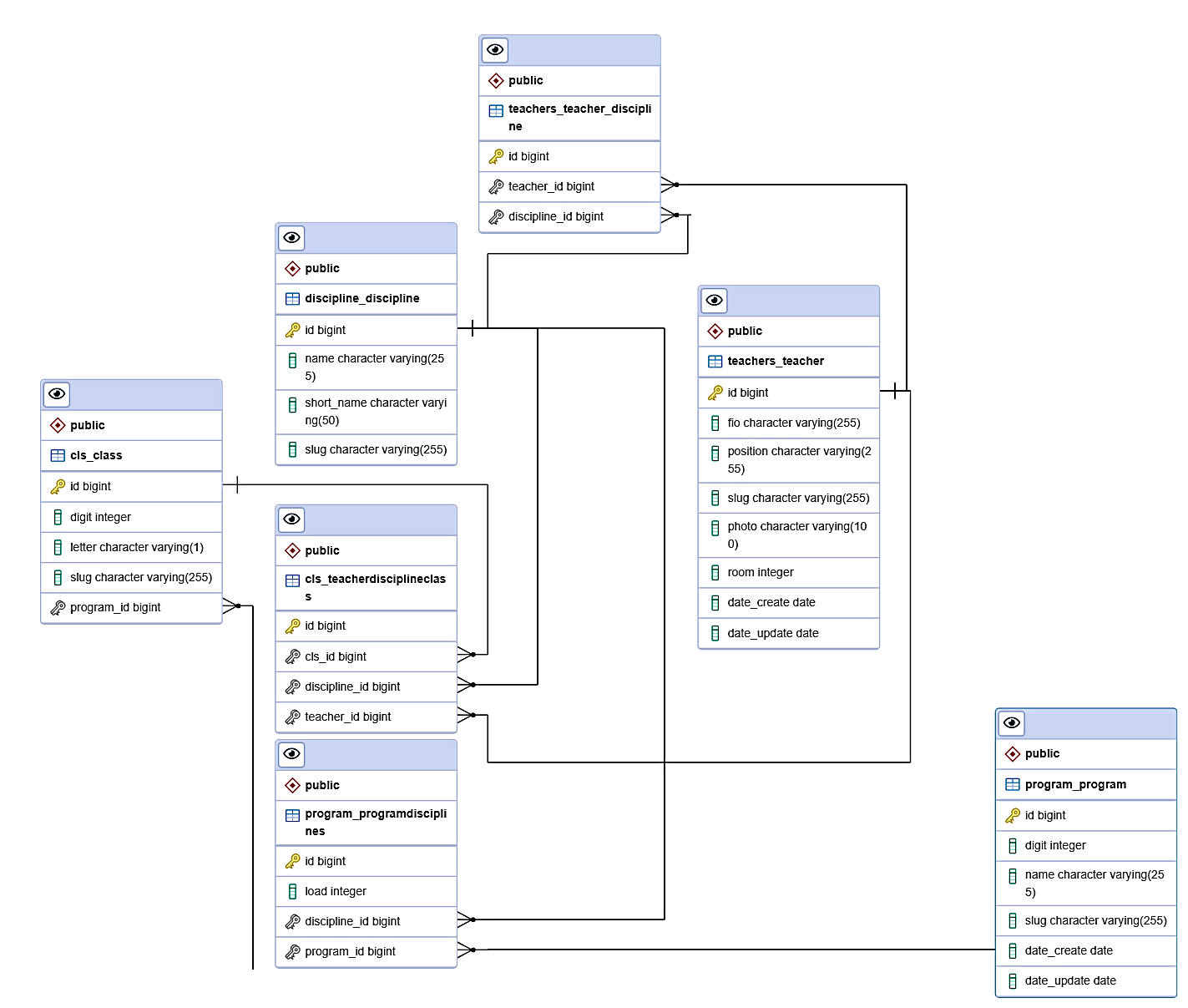


Рисунок 2.5 — Диаграмма классов

# Особенности реализации

## Реализация проекта

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

Листинг 3.1 — Настройка разрешённых хостов

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

Затем вписываем нашу БД:

Листинг 3.2 — Конфигурирование 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.

Листинг 3.3 — Настройка статических файлов

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.

Листинг 3.4 — Настройка папки увшф

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

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

Листинг 3.5 — Подключение дебаггера

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:

Листинг 3.6 — Создание меню

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:

Листинг 3.7 — Вставка блоков

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):

Листинг 3.8 — Вызов функции и передача аргументов

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**:

Листинг 3.9 — Файл base\_tags

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 **menus.html**:

Листинг 3.10 — Отрисовка меню

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:

Листинг 3.11 — Соединение путей

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].

Листинг 3.12 — Настройки Media при отладке

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:

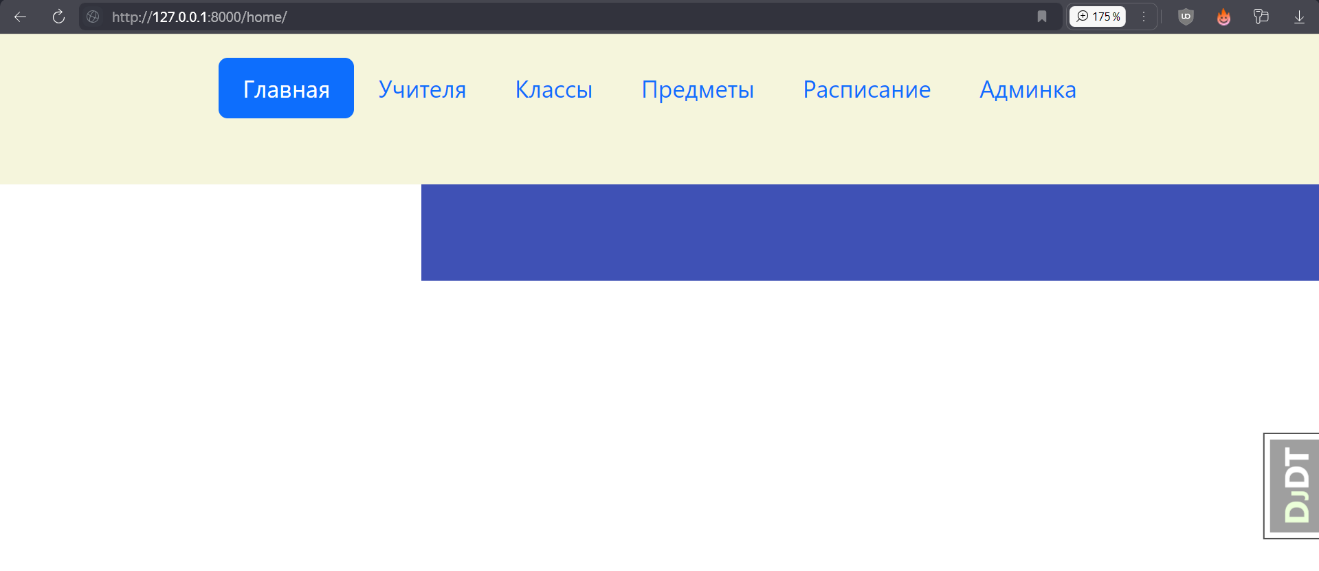


Рисунок 3.1 — Главная страница приложения

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

Далее нужно посмотреть имя приложения в файле apps.py. Обычно оно выглядит так:

Листинг 3.13 — apps.py

1. class TeacherConfig(AppConfig):
2. verbose\_name = 'Учителя'
3. default\_auto\_field = 'django.db.models.BigAutoField'
4. name = 'teachers'

Далее добавить это приложение в INSTALLED\_APPS в файле настроек:

Листинг 3.14 — Добавление приложения

1. INSTALLED\_APPS = […
2. 'teachers.apps.TeacherConfig', …
3. ]

Покажем написание класса модели в файле models:

Листинг 3.15 — Создание 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, максимальный 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>.

Вложенный класс **Meta** (10) содержит информацию о классе. В классе Meta можно устанавливать имена, которые будут устанавливаться для таблицы при создании, если они отличны от имени класса, порядок объектов, автоматические имена на сайте администратора.

Метод **save** вызывается при попытке сохранить объект. Там можно переопределять поля, выполнять пред- и последействия. При этом желательно выполнять операции с БД с помощью строки

with transaction.atomic():

…

Это нужно чтобы все операции при сохранении одного объекта выполнялись внутри одной транзакции.

Можно задать метод **serializable** для правильного сохранения объекта в объект json. Он определяется при помощи декоратора property. Это нужно, чтобы обращаться к методу как к полю (без скобок). Такой метод называется свойство.

Потом нужно сформировать файлы миграций (1), которые будут автоматически сохраняться в папке migrations, и применить их, то есть создать таблицы (2):

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

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:

Листинг 3.16 — Класс ListView

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]:

Листинг 3.17 — Конфигурирование БД

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 интересно посмотреть на функцию url (4), которая будет формировать url по имени шаблона и передавать туда параметр:

Листинг 3.18 — Цикл в html

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:

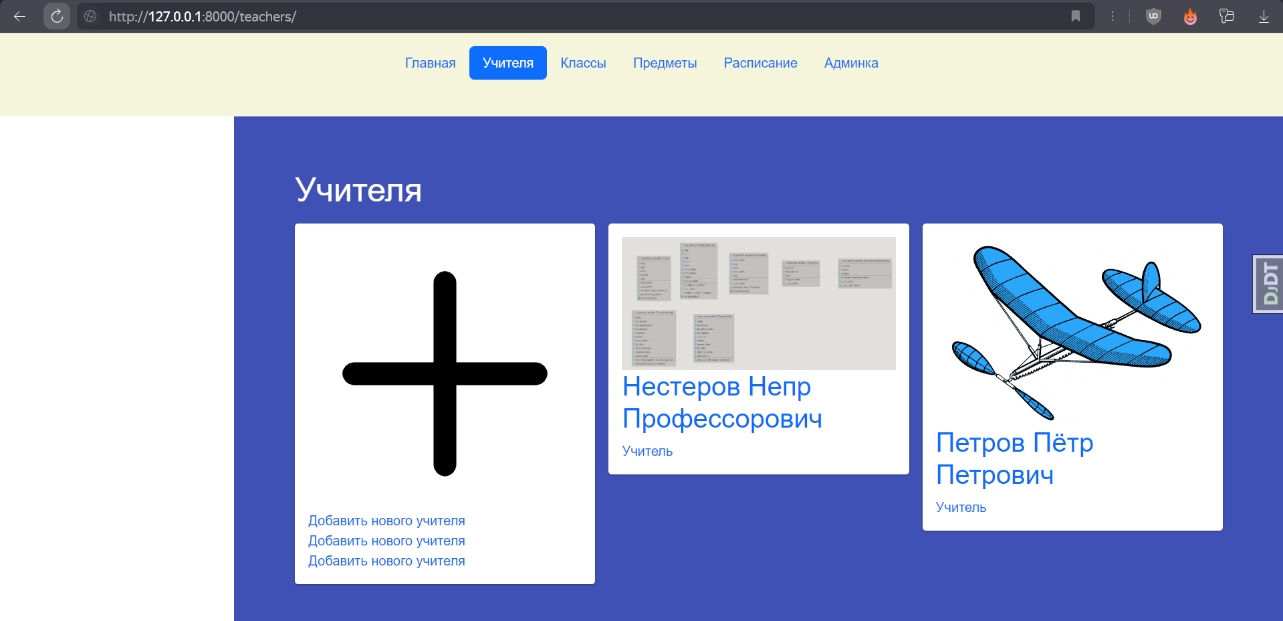


Рисунок 3.2 — Страница учителей

Плюс сформирован тем же образом, что и остальные карточки.

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

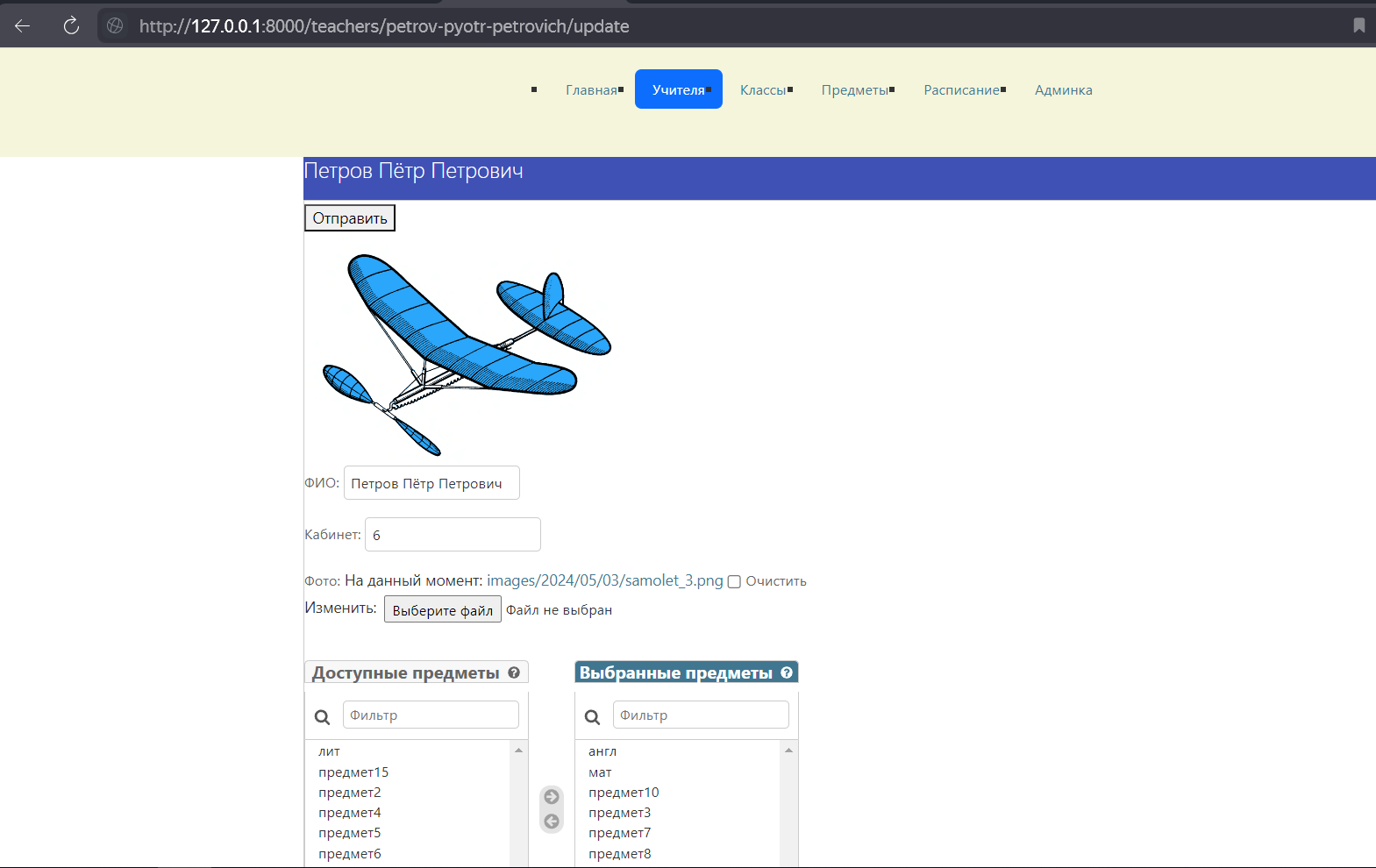


Рисунок 3.3 — Страница обновления учителя

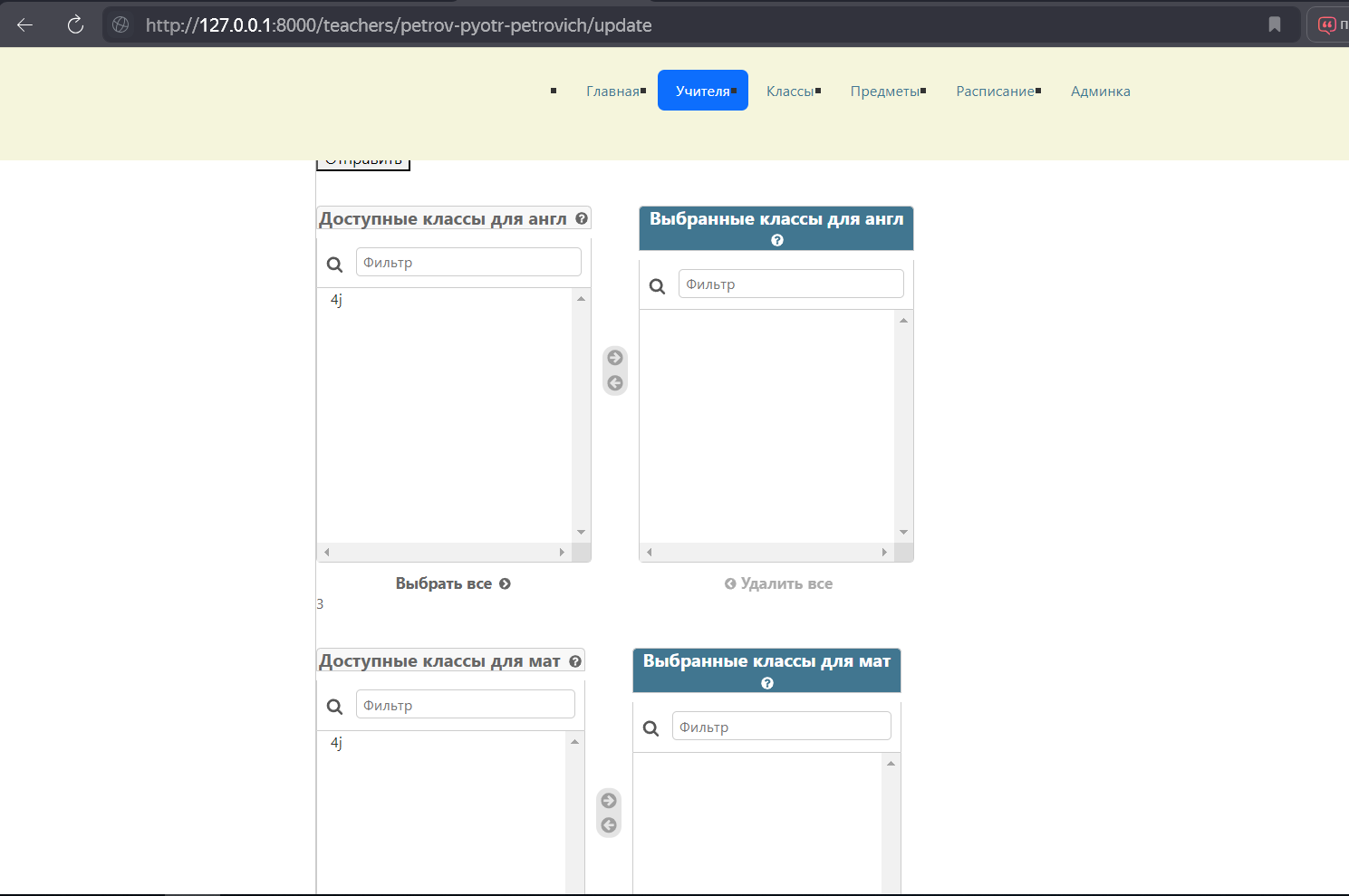


Рисунок 3.4 — Выпадение списка классов

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):

Листинг 3.19 — Подключение файла в блоке

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:

Листинг 3.20 — Форматирование списка

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:

Листинг 3.21 — Код формы

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:

Листинг 3.22 — Observer

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:

Листинг 3.23 — Установка слушателя на мультисписок

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:

Листинг 3.24 — Обработка выбора опций

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 исходного кода 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:

Листинг 3.25 — Создание динамических полей с помощью ajax

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/':

Листинг 3.26 — Получение данных о классах из БД

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:

Листинг 3.27 — Обработка отправки формы

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:

Листинг 3.28 — Сохранение динамических полей

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.

Произведём вставку заранее полученных данных в проект и БД. Для этого удобнее открыть shell:

python manage.py shell

Она используется для запуска интерактивной оболочки для взаимодействия с БД.

Далее вставим скрипт из create.py. Он представлен в Приложении Ж.

Для выхода из интерактивной оболочки используется команда

exit()

## Развёртка в контейнере Docker

Теперь наше приложение нужно уметь разворачивать на сервере, потому что в дальнейших планах его следует поместить на платный сервер. Развёртка будет происходить в контейнере docker с помощью одноимённого приложения. Представим код Dockerfile:

Листинг 3.29 — Dockerfile

1. FROM quay.io/fedora/python-311
2. WORKDIR /app
3. RUN curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py && \
4. python get-pip.py && \
5. rm get-pip.py
6. COPY . /app
7. RUN pip install -r requirements.txt
8. USER root
9. RUN dnf install -y postgresql postgresql-server
10. CMD ["sh", "-c", "python manage.py migrate && \
11. python manage.py createsuperuser --noinput --username 1234 --email vikavladim@mail.ru --password '12345678' && \
12. python manage.py runserver 0.0.0.0:8000"]

Распишем код построчно. В первой строке происходит скачивание образа Python 3.11 с docker hub quay репозитория. Ранее загрузка производилась с официального хаба Docker, но 30.05.24 Docker ушёл из России и приходится искать обходные пути.

Далее с помощью установленного Python скачиваем pip по инструкции, которую можно найти на github (2). Здесь происходит обращение к странице из терминала, скачивание и именование файла get-pip.py. Потом этот файл, который является скриптом требуется запустить, а после удалить.

В строке 8 мы копируем все файлы проекта в директорию app.

В строке 10 мы запускаем установку пакетов из файла requirements, который ранее сформировали с помощью команды

pip freeze > reqirements.txt

Потом в строке 12 мы получаем права администратора и в строке 14 скачиваем postgres с помощью dnf. Dnf - это менеджер пакетов для, который заменяет yum.

В строках 16-18 происходит запуск миграций, запуск файла, создание суперпользователя.

Также для конфигурирования нам понадобится файл docker-compose.yml. Приведём его содержимое:

Листинг 3.30 — docker-compose.yml

1. version: '3'
2. services:
3. db:
4. image: quay.io/sclorg/postgresql-15-c9s
5. environment:
6. POSTGRESQL\_USER: postgres
7. POSTGRESQL\_PASSWORD: 32767
8. POSTGRESQL\_DATABASE: schedule
9. volumes:
10. - postgres\_data:/var/lib/pgsql/data
11. web:
12. build: .
13. ports:
14. - "8000:8000"
15. depends\_on:
16. - db
17. volumes:
18. - .:/app
19. environment:
20. - DJANGO\_SUPERUSER\_USERNAME=1234
21. - DJANGO\_SUPERUSER\_PASSWORD=12345678
22. - DJANGO\_SUPERUSER\_EMAIL=12345678
23. volumes:
24. postgres\_data:

В первой строке указывается версия языка docker-compose. Далее в строках 3 и 12 указываются имена служб (db и web).

В строках 5-8 и 20-23 определяются переменные среды, которые нужно использовать.

В строках 9-10 и 25-26 используется место для сохранения томов, в том числе между запусками контейнера.

В строке 16-17 указывается последовательность запуска служб. То есть сервис web установится только после db.

## Руководство пользователя

Покажем на Рисунке 3.5 иерархию интерфейсов для пользователя:

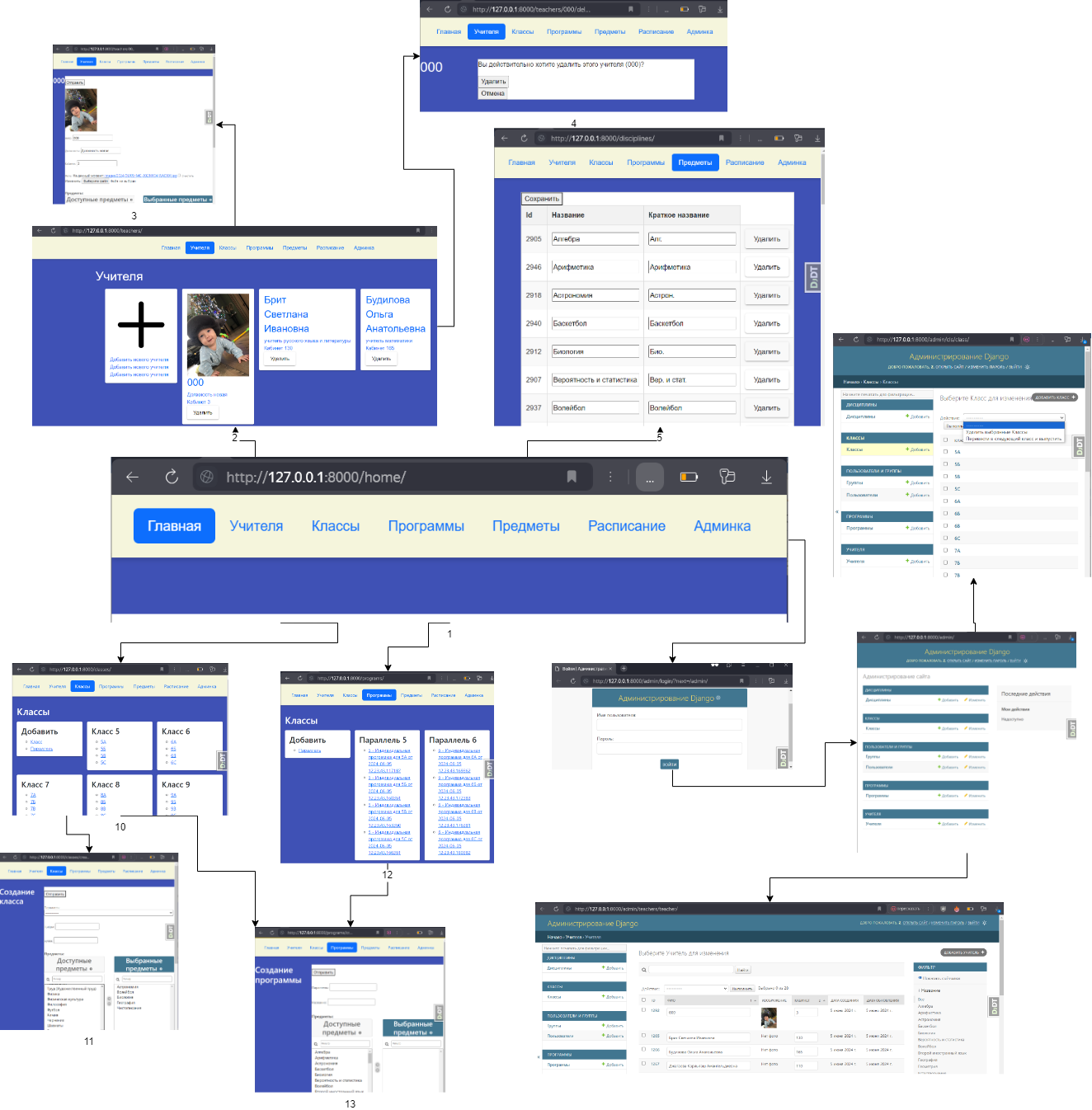


Рисунок 3.5 — Иерархия интерфейсов

При входе на главную страницу пользователь видит интерфейс 1. Покажем его подробнее на Рисунке 3.6:

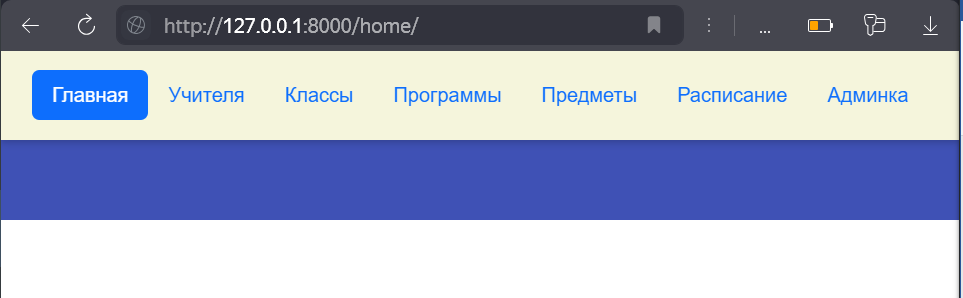


Рисунок 3.6 — Главная страница приложения

На главной странице, как и на всех остальных, находится главное меню. Отсюда можно перейти в интерфейс учителей (2) (Рисунок 3.7):

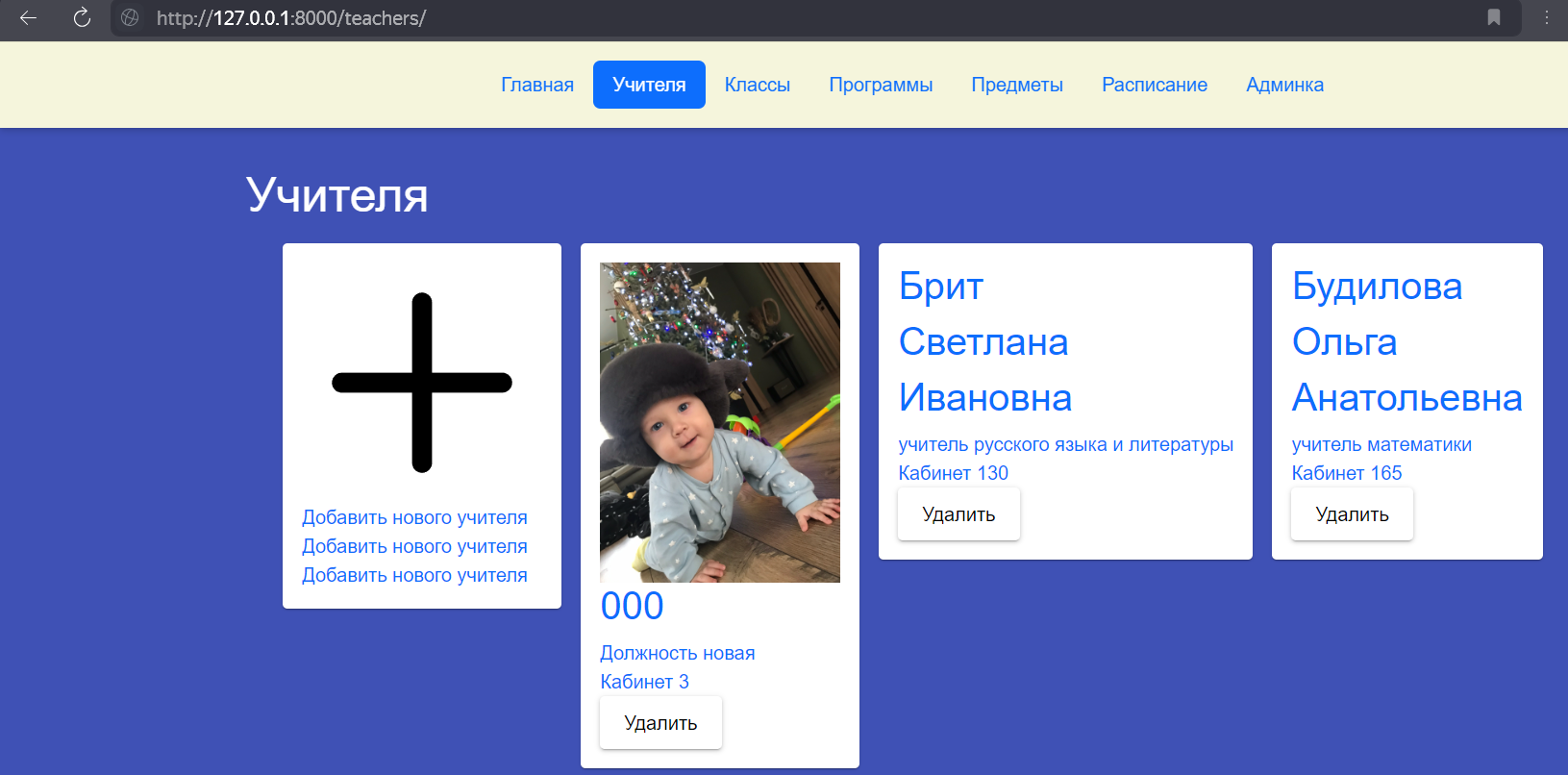


Рисунок 3.7 — Экран списка учителей

Здесь можно отредактировать учителя (Рисунок 3.8), нажав по любому месту плашки, или добавить нового (Рисунок 3.9), нажав на изображение с плюсом, удалить учителя (Рисунок 3.10). Стоит отметить, что при изменении предметов, которые ведёт учитель, под списком выпадают динамическим образом новые списки для выбора классов, в которых учитель преподаёт этот предмет (Рисунок 3.8).: добавить номера экранов

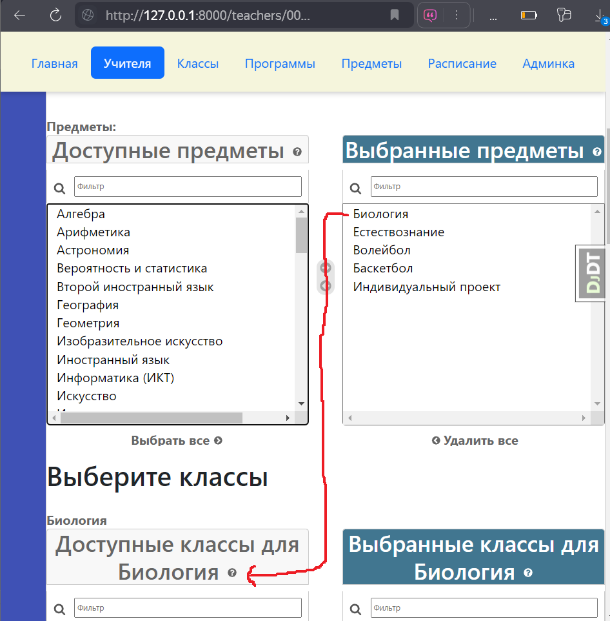
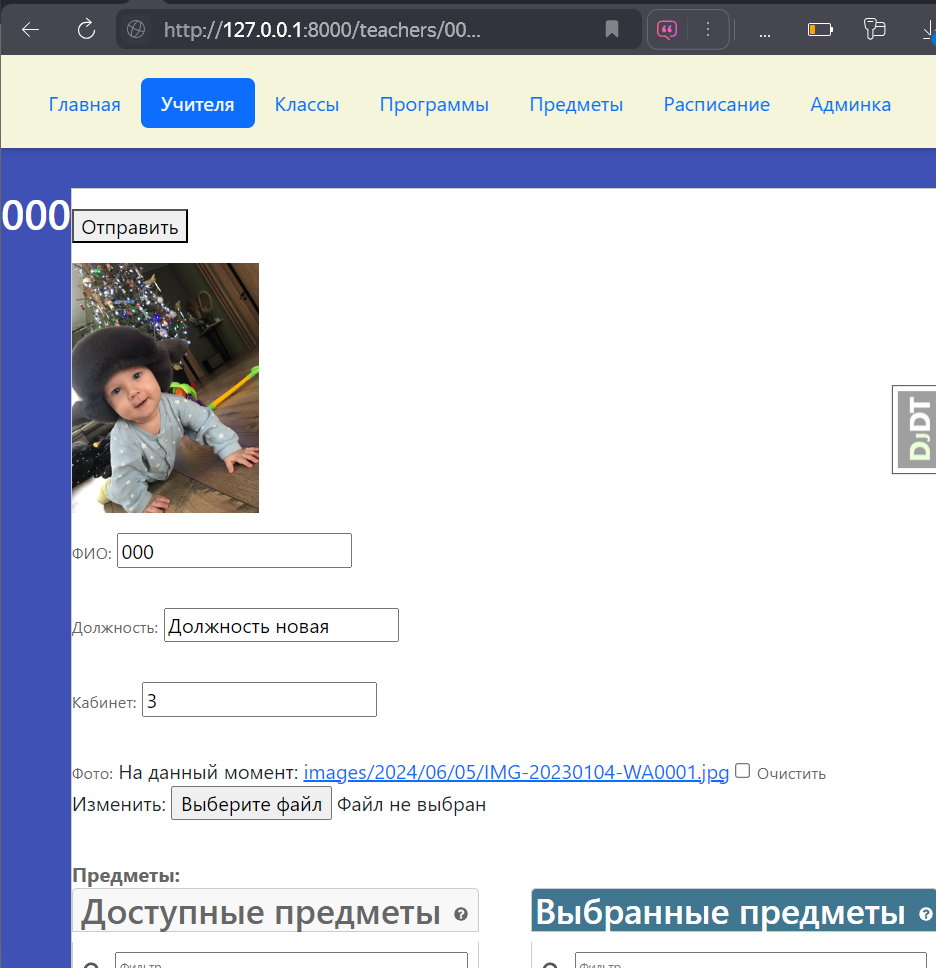


Рисунок 3.8 — Экран обновления учителей

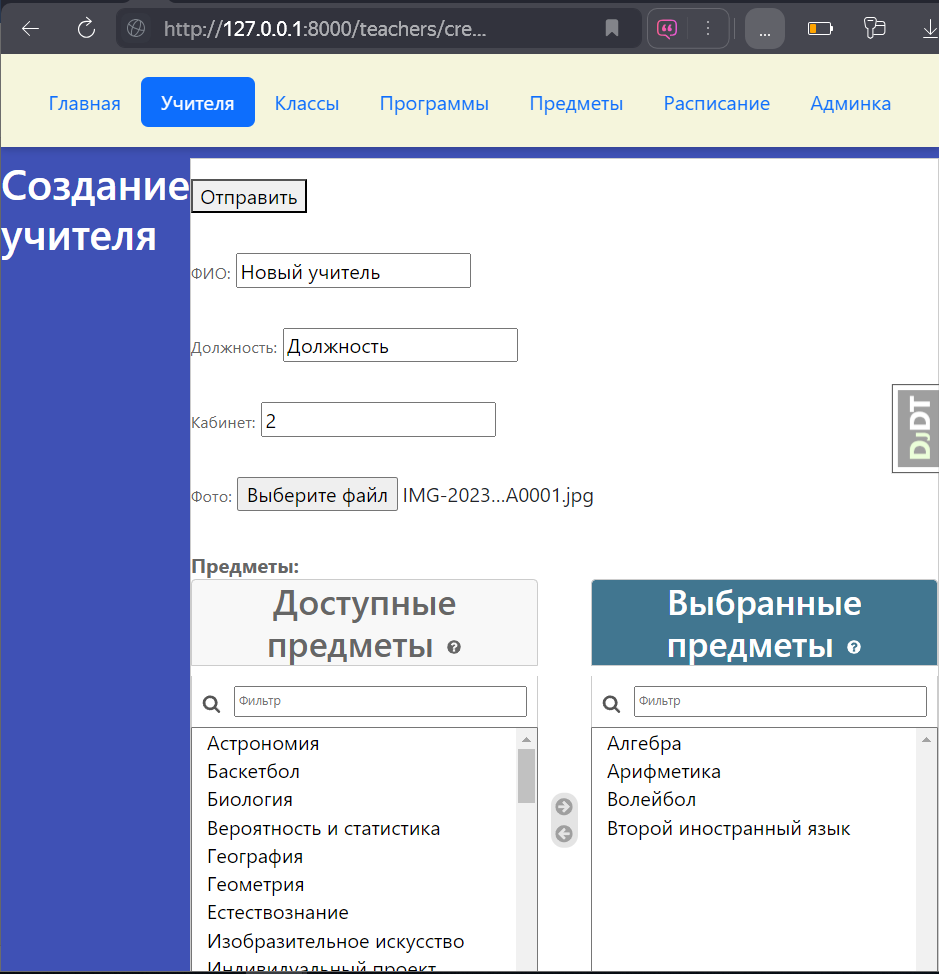


Рисунок 3.9 — Экран создания учителя



Рисунок 3.10 — Экран удаления учителя

Покажем экран создания и редактирования дисциплин (5) (Рисунок 3.11). Удаление происходит на всех экранах одинаково по кнопке удалить как на Рисунке 3.10.

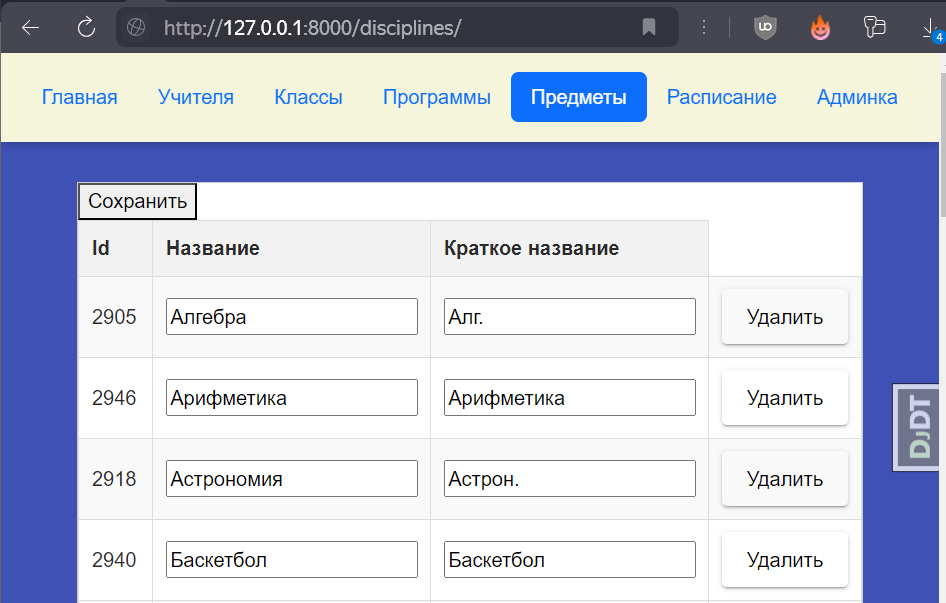


Рисунок 3.11 — Экран предметов

Рассмотрим просмотр и редактирование классов на экранах 10-11. Они приведены на Рисунке 3.12 и 3.13. Обратим внимание, что после выбора предмета или программы нужно выбрать учителя и количество часов ниже (Рисунок 3.13).



Рисунок 3.12 — Экран всех классов

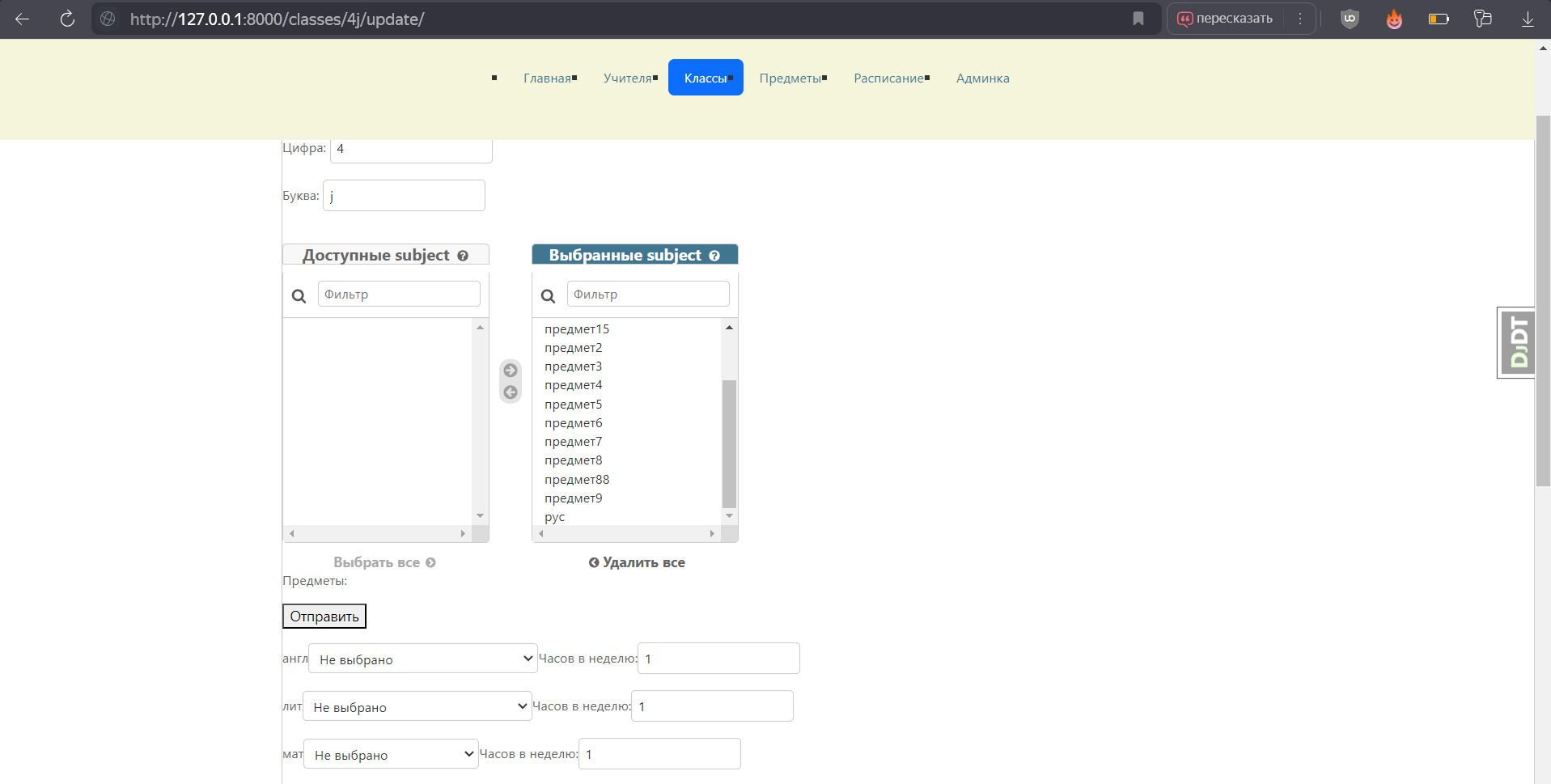
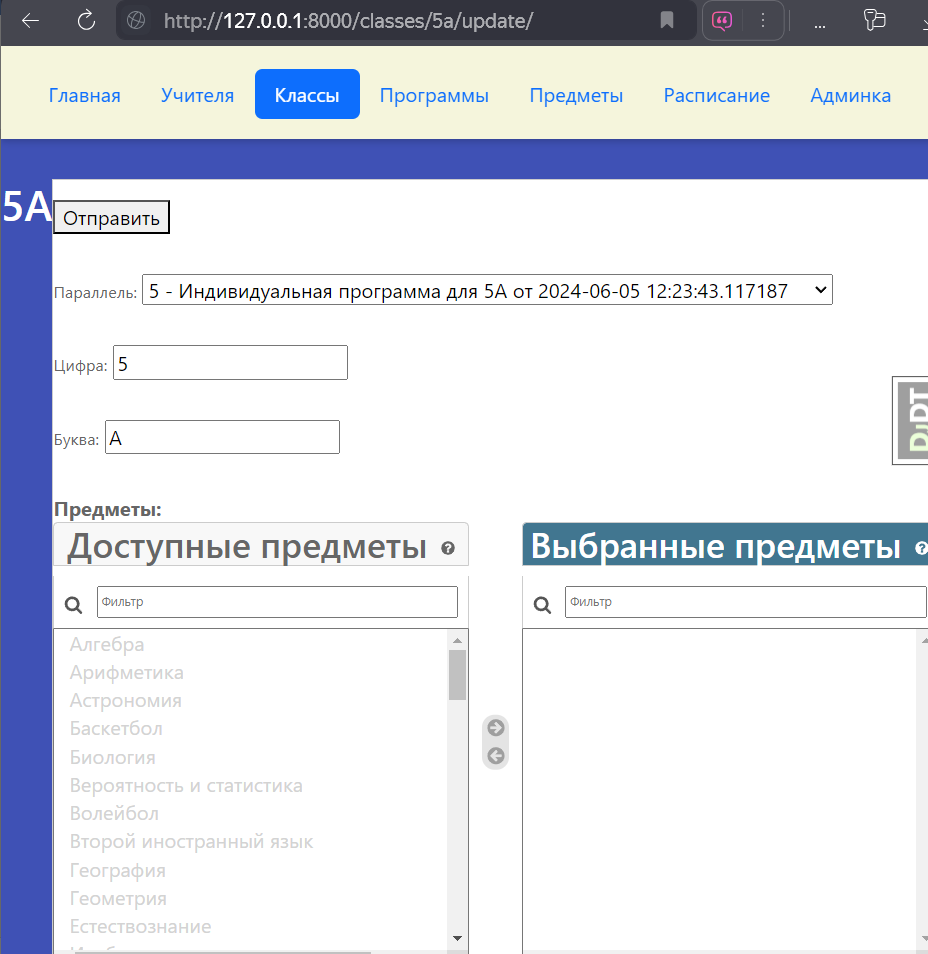


Рисунок 3.13 — Экран редактирования 5А класса

Покажем действия с параллелями на Рисунке 3.14:

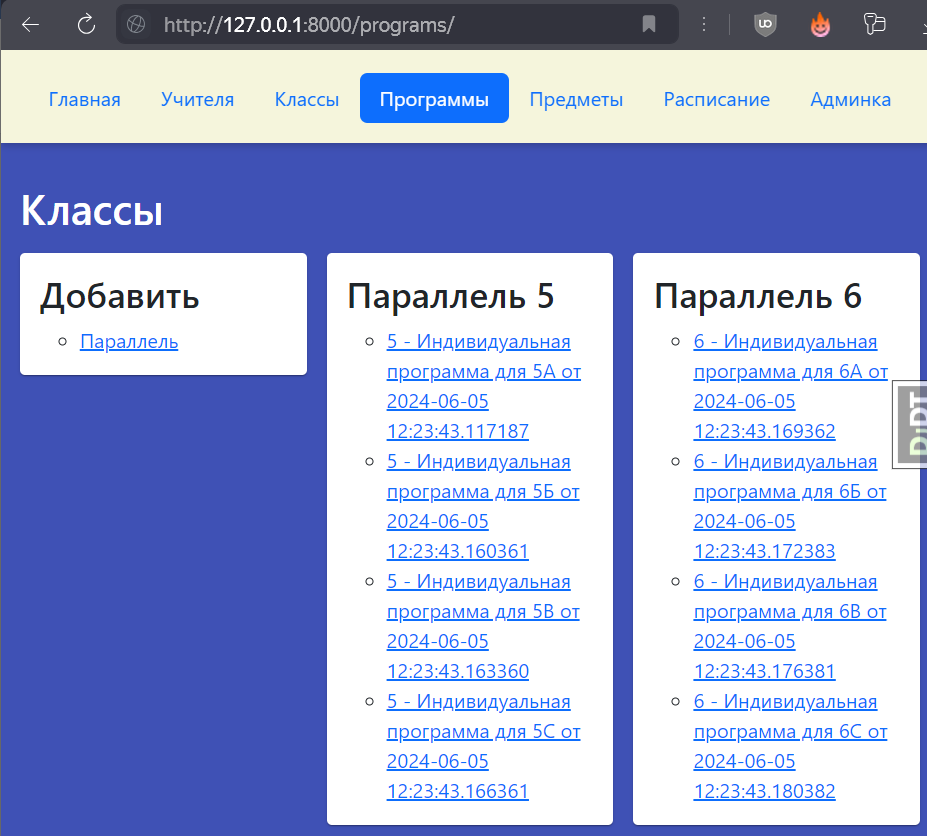


Рисунок 3.14 — Экран всех программ

Программу можно организовывать на несколько классов одновременно или без классов вообще, чтобы составлять программу для будущих классов. Программа не запрещает использование программы 5-ого класса для, например, 6-ого. Добавление параллели (экран 13) очень напоминает Рисунок 3.13, поэтому он здесь не приводится.

Продемонстрируем экраны администратора. Сначала нужно войти с паролем администратора (Рисунок 3.15):

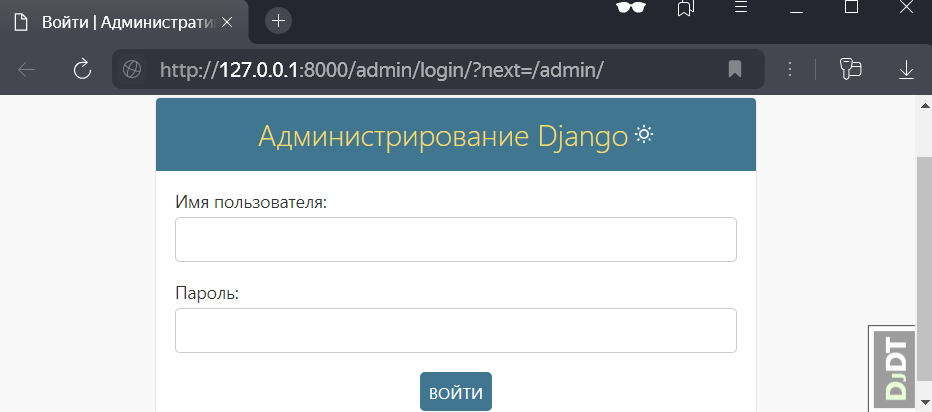


Рисунок 3.15 — Окно входа

Далее появится сайт администратора с такой главной страницей (Рисунок 3.16):

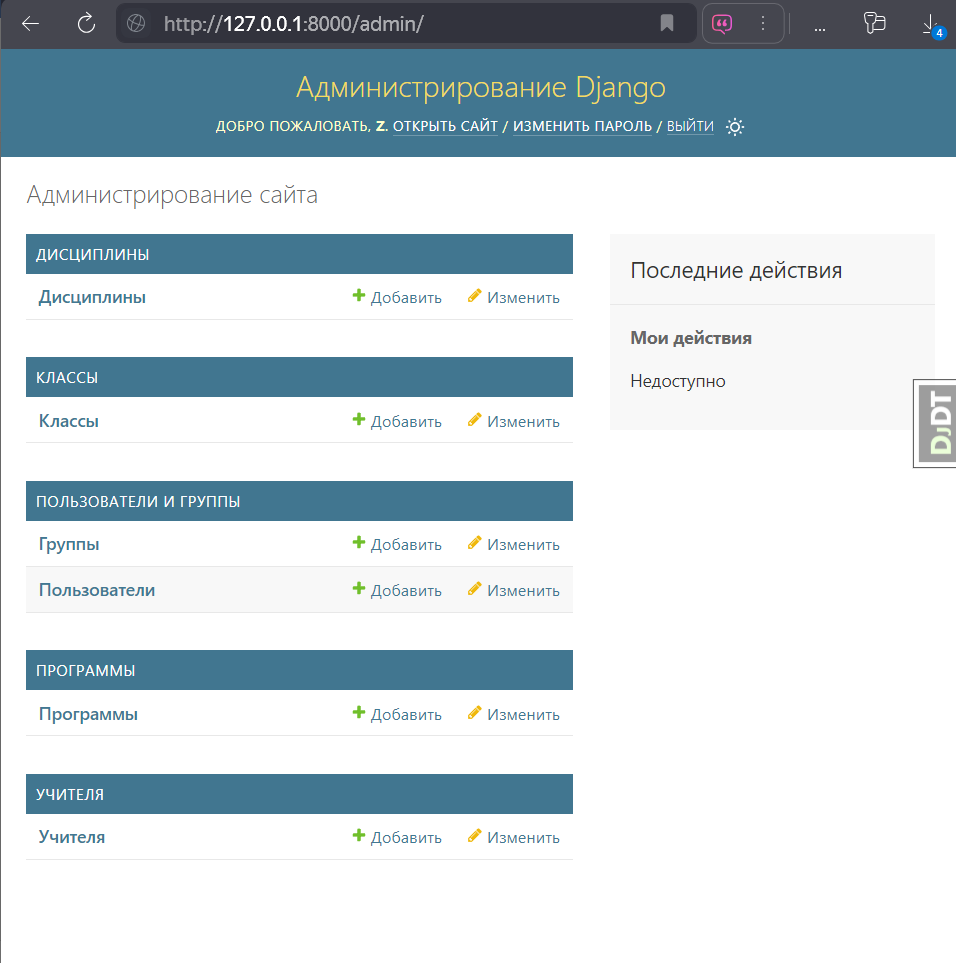


Рисунок 3.16 — Главная страница администратора

Перейдём, например, по ссылке классов (Рисунок 3.17). Классы можно также редактировать, можно выбрать несколько или все и перевести или выпустить.

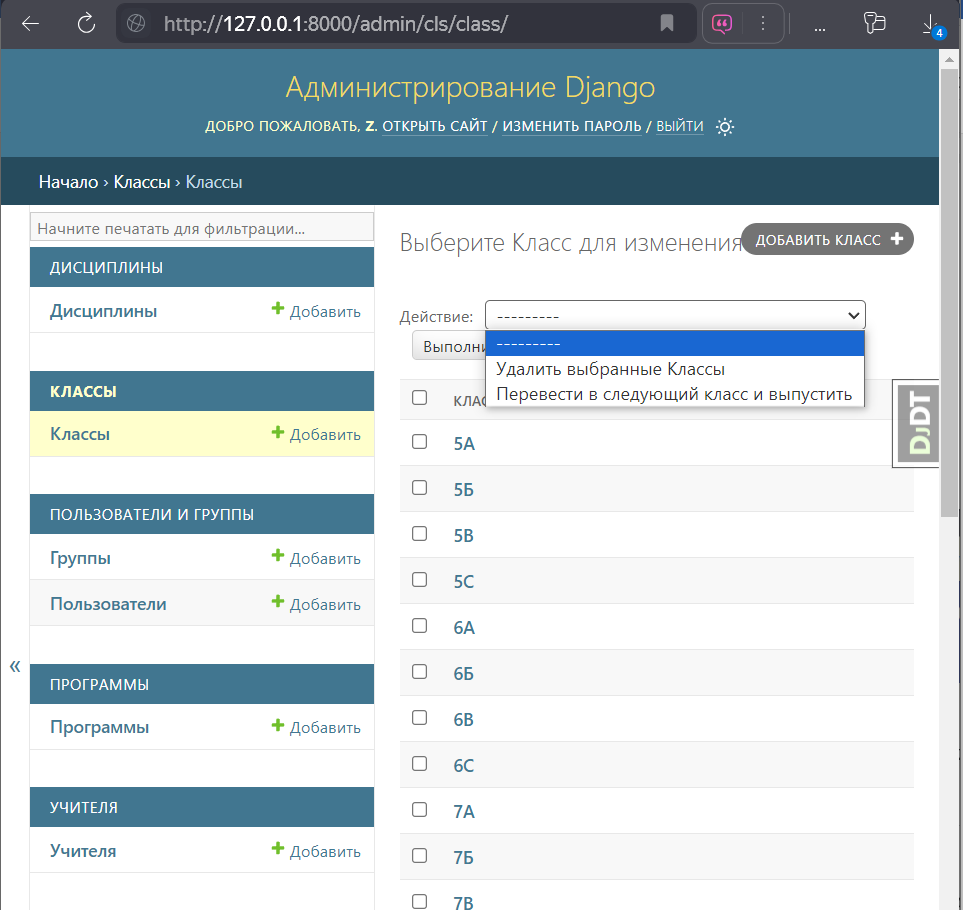


Рисунок 3.17 — Страница администратора для классов

Покажем аналогичную страницу для учителей (Рисунок 3.18). Здесь представлены фильтры, сортировка, поиск по полям, изменения прямо на странице в таблице.

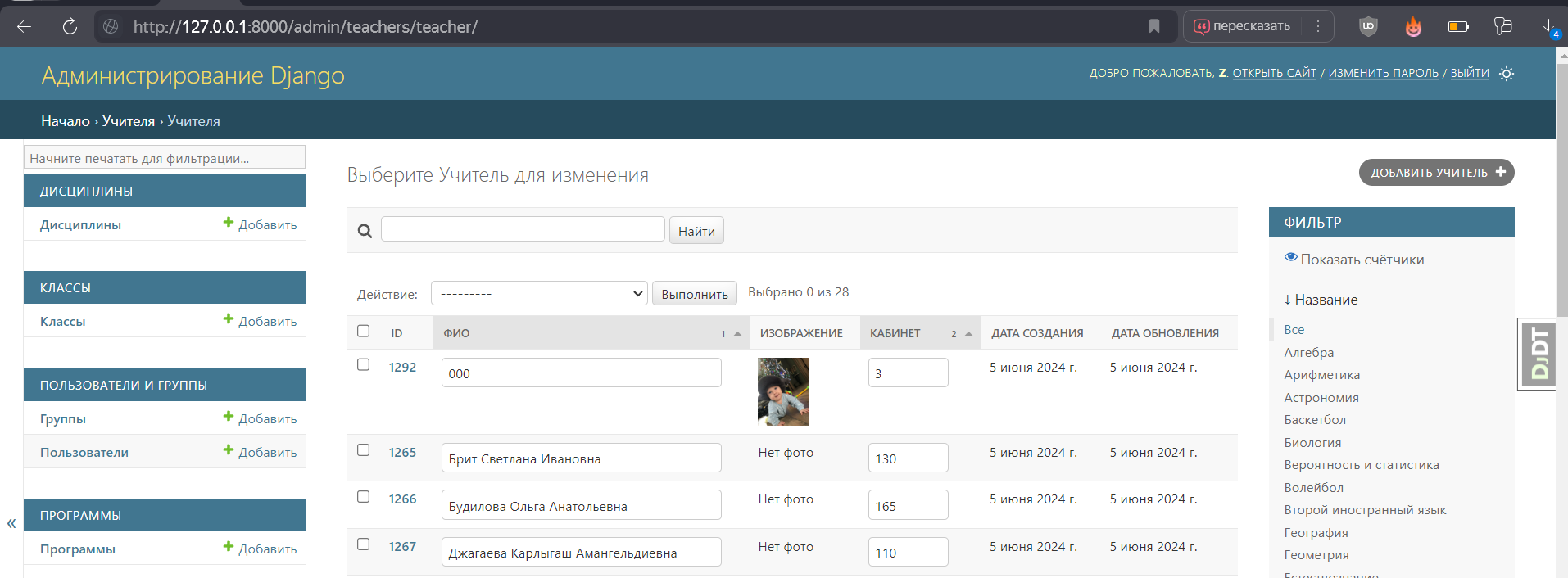


Рисунок 3.18 — Страница администратора для учителей

В ходе разработки приложения получилась схема Rest API, представленная в Приложении Б.

# Заключение

В теоретической части были рассмотрены аналоги разработки, разобраны два наиболее популярны алгоритма составления расписаний: метод раскраски графа и генетический алгоритм.

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.

Для развёртки приложения на сервере был написан Dockerfile и docker-compose.yml файлы, в которых приведена установка необходимых модулей и их конфигурирование.

Разработана соответствующая документация в виде руководства пользователя и схемы Rest API, была построена иерархия интерфейсов.

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 литературы

1. 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) .
2. Экспресс-расписание. URL: <https://pbprog.ru/catalog/timetable> (дата обращения: 25 апреля 2024).
3. Хронограф для расписания уроков. URL: <https://uogornogo.obr.sakha.gov.ru/ikt-v-obrazovanii/hronograf-dlja-raspisanija-urokov> (дата обращения: 22 апреля 2024).
4. Ника Софт - Ника Колледж. URL: <https://nikasoft.ru/nika-college/> (дата обращения: 03 мая 2024).
5. ASC Timetables. URL: <https://www.asctimetables.com/> (дата обращения: 01 мая 2024).
6. Уроки по Django. URL: <https://proproprogs.ru/django4> (дата обращения: 28 апреля 2024).
7. Документация по Django. URL: <https://docs.djangoproject.com/en/5.0/> (дата обращения: 20 апреля 2024).
8. Django | Отношения Многие ко многим. URL: <https://metanit.com/python/django/5.7.php> (дата обращения: 5 мая 2024)
9. Приложение «АВТОРасписание» | Ютел. URL: <http://utelksp.ru/programs/prilozhenie-avtoraspisanie> (дата обращения: 7 мая 2024)
10. JavaScript | Основы XMLHttpRequest. URL: <https://learn.javascript.ru/ajax-xmlhttprequest> (дата обращения: 20 мая 2024)
11. JavaScript. URL: <https://www.webpupil.ru/javascript_view.php?id=243> (дата обращения: 21 мая 2024)
12. JavaScript. | Fetch. URL: <https://learn.javascript.ru/fetch> (дата обращения: 22 мая 2024)
13. METANIT.COM Сайт о программировании. values и values\_list и сортировка. URL: <https://metanit.com/python/django/5.14.php> (дата обращения: 22 мая 2024)
14. JavaScript | FormData. URL: <https://learn.javascript.ru/formdata> (дата обращения: 23 мая 2024)
15. Генетический алгоритм. Просто о сложном. Рассказ Марка Андреева. Хабр. URL: <https://habr.com/ru/articles/128704/> (дата обращения: 23 мая 2024)
16. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.
17. Ковалева Е.А Генетический алгоритм в задаче составления расписания // Старт в науке. – 2018. – № 5-1.
18. Comparison of FastAPI with Django and Flask. URL: <https://www.geeksforgeeks.org/comparison-of-fastapi-with-django-and-flask/> (дата обращения: 29 мая 2024)
19. Автоматическая документация API Django Rest Framework с помощью DRF Spectacular. Хабр. URL: <https://habr.com/ru/articles/733942/> (дата обращения: 27 мая 2024)
20. Запускаем PostgreSQL в Docker: от простого к сложному. Хабр. URL: <https://habr.com/ru/articles/578744/> (дата обращения: 28 мая 2024)
21. Django | Язык шаблонов. URL: <https://docs.djangoproject.com/en/5.0/ref/templates/language/> (дата обращения: 28 мая 2024)
22. Оптимизация перебора. Хабр. URL: <https://habr.com/ru/articles/190850/> (дата обращения: 29 мая 2024)
23. Министерство просвещения Российской Федерации Федеральное государственное бюджетное научное учреждение «Институт возрастной физиологии Российской академии образования». Рекомендации по составлению расписания уроков для обучающихся начального общего образования., М., 2021
24. Ковалева Е.А. Визуализация задачи составления расписания при помощи графов // Старт в науке. – 2020. – № 1.
25. <https://www.tune-it.ru/web/leksa/blog/-/blogs/1996258>
26. Сводная таблица аналогов

Таблица А.1 — Сводная таблица аналогов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование программы | Ссылка | Цена, руб. | Качество по 5-балльной шкале | Особенности |
| Экспресс-расписание Колледж | [https://pbprog.ru/products/programs.php?ELEMENT\_ID=376](https://vk.com/away.php?to=https%3A%2F%2Fpbprog.ru%2Fproducts%2Fprograms.php%3FELEMENT_ID%3D376&cc_key=) | 2 500 - 300 000 руб. | 3 | Есть проверка аудиторий на вместимость. |
| 1С: Автоматизированное составление расписания. Колледж | [https://solutions.1c.ru/catalog/asp\_spo](https://vk.com/away.php?to=https%3A%2F%2Fsolutions.1c.ru%2Fcatalog%2Fasp_spo&cc_key=) | 22 000 – 26 000 | 4 | Решение прорабатывается под отдельную организацию. |
| Система «АВТОРасписание» | [https://www.mmis.ru/programs/avtor](https://vk.com/away.php?to=https%3A%2F%2Fwww.mmis.ru%2Fprograms%2Favtor&cc_key=) | 20 000 - 150 000 | 5 | Развивается с 1996 года.  Наличие подробной статистики и объективной оценки качества любого варианта расписания.  Возможность поддержки любого национального языка (по желанию клиента). |
| Хронограф | [http://softrare.ru/windows/hronograf-30-master](https://vk.com/away.php?to=http%3A%2F%2Fsoftrare.ru%2Fwindows%2Fhronograf-30-master&cc_key=) | бесплатно | 1 | Разрешается указывать данные о кабинетах, инвентаре, расписание звонков. |
| «Ника-Колледж» | [http://nikasoft.ru/nika-college/](https://vk.com/away.php?to=http%3A%2F%2Fnikasoft.ru%2Fnika-college%2F&cc_key=) | 11 000 – 25 000 на 5 лет. Демо – 2500. | 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. |
| aSc | [https://www.asctimetables.com/](https://vk.com/away.php?to=https%3A%2F%2Fwww.asctimetables.com%2F%23%21%2Fhome&cc_key=) | 150 000 – 200 000 | 3 | Зарубежная программа с удобным интерфейсом имеет хорошие отзывы. |

1. Схема Rest API

Таблица Б.1 — Схема Rest API

|  |  |  |
| --- | --- | --- |
| Метод | URL | Действие |
| Классы | | |
| get | /classes/create\_class/class\_list/ | Список всех классов |
| get | /classes/create\_class/ | Переход на страницу создания класса |
| post | /classes/create\_class/ | Отправка формы, создание класса |
| get | /classes/<slug:slug>/update/ | Переход на страницу обновления класса |
| post | /classes/<slug:slug>/update/ | Отправка формы, обновление класса |
| post | /classes/getTeachersFromDB/ | Вызывает одноимённую функцию, пердаёт список доступных и закреплённых за классом учителей |
| post | /classes/teachers\_field\_form/ | Сохраняет в БД несвязанные с формой поля |
| get | /classes/<slug:slug>/delete/ | Удаляет класс |
| post | /classes/change\_disciplines/ | В зависимости от выбираемой программы изменяет отображения дисциплин и нагрузки |
| Дисциплины | | |
| get | /disciplines/ | Отображает мультиформу создания и изменения дисциплин |
| post | /disciplines/ | Сохраняет мультиформу |
| get | /disciplines/<slug:slug>/delete/ | Удаляет дисциплины |
| Программы | | |
| get | /programs/ | Отображает список всех программ |
| get | /programs/create\_program/ | Форма создания программы |
| post | /programs/create\_program/ | Сохранение данных формы |
| get | /programs/<slug:slug>/update/ | Отображение формы изменения |
| post | /programs/<slug:slug>/update/ | Сохранение формы |
| post | /programs/getHoursFromDB/ | Отправляет данные выбранных дисциплин |
| post | /programs/load\_field\_form/ | Отправка дополнительных полей и создание несвязанных с моделью записей |
| Учителя | | |
| get | /teachers/ | Список всех учителей |
| get | /teachers/<slug:slug>/update/ | Отображение формы обновления учителя |
| post | /teachers/<slug:slug>/update/ | Отправка формы и её сохранение |
| get | /teachers/create/ | Форма создания учителя |
| post | /teachers/create/ | Сохранение учителя |
| get | /teachers/<slug:slug>/delete/ | Удаление учителя |
| post | /teachers/getDataFromDB/ | Отображает дисциплины и классы учителя |
| post | /teachers/classes\_field\_form/ | Отправляет и сохраняет классы учителя |
|  | | |
| get | /admin/\* | Страницы управления администратора |
| get | / | Перенаправление на главную страницу |
| get | /\_\_debug\_\_/ | Вкладка отладки проекта |
| get | /api/schema/ | Статическая схема проекта для формирования документации |
| get | /api/schema/swagger-ui/ | Представление UI интерфейса для просмотра документации |

1. Код создания модели прецедентов

Листинг В.1 — Создание модели прецедентов

1. @startuml
2. left to right direction
3. skinparam actor {
4. BackgroundColor LightBlue
5. BorderColor DarkBlue
6. }
7. actor Учитель
8. actor Клиент
9. actor Учащийся
10. actor Родитель
11. actor Диспетчер
12. actor Администратор
13. actor Директор
14. actor Заместитель
15. Директор-->Администратор
16. Заместитель-->Администратор
17. Родитель-->Клиент
18. Учащийся-->Клиент
19. Учитель --> (Отправить пожелания)
20. (Отправить пожелания) --> Администратор
21. Учитель --> (Просмотреть график работы)
22. Администратор --> (Отправить сообщение)
23. Администратор --> (Отправить данные)
24. Администратор --> (Перевести)
25. Администратор --> (Понизить)
26. Перевести --> Клиент
27. Понизить --> Клиент
28. Администратор --> (Уволить)
29. Администратор --> (Нанять)
30. Уволить --> Учитель
31. Нанять --> Учитель
32. (Отправить нагрузку) --> (Отправить данные) <<extends>>
33. (Отправить график заведения) --> (Отправить данные) <<extends>>
34. (Отправить личные данные) --> (Отправить данные) <<extends>>
35. (Отправить пожелания) --> (Отправить данные) <<extends>>
36. (Отправить программу) --> (Отправить данные) <<extends>>
37. Диспетчер --> (Отправить расписание)
38. (Отправить расписание)--> Учитель
39. (Отправить расписание)--> Клиент
40. (Отправить расписание)--> Администратор
41. Клиент -->(Просмотреть расписание)
42. Учитель -->(Просмотреть расписание)
43. @enduml
44. Шкала трудности учебных предметов

Таблица Г.1 — Шкала трудности учебных предметов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Учебные предметы | Количество баллов (по классам) | | | | |
| 5 класс | 6 класс | 7 класс | 8 класс | 9 класс |
| Физика | - | - | 8 | 9 | 13 |
| Химия | - | - | - | 10 | 12 |
| История | 5 | 8 | 6 | 8 | 10 |
| Иностранный язык | 9 | 11 | 10 | 8 | 9 |
| Математика | 10 | 13 | - | - | - |
| Геометрия | - | - | 12 | 10 | 8 |
| Алгебра | - | - | 10 | 9 | 7 |
| Природоведение | 7 | 8 | - | - | - |
| Биология | 10 | 8 | 7 | 7 | 7 |
| Литература | 4 | 6 | 4 | 4 | 7 |
| Информатика и ИКТ | 4 | 10 | 4 | 7 | 7 |
| Русский язык/Родной язык | 8 | 12 | 11 | 7 | 6 |
| География | - | 7 | 6 | 6 | 5 |
| Изобразительное искусство | 3 | 3 | 1 | - | - |
| Мировая художественная культура (МХК) | - | - | 8 | 5 | 5 |
| Музыка | 2 | 1 | 1 | 1 | - |
| Обществознание (включая экономику и право) | 6 | 9 | 9 | 5 | 5 |
| Технология | 4 | 3 | 2 | 1 | 4 |
| Основы безопасности жизнедеятельности | 1 | 2 | 3 | 3 | 3 |
| Физическая культура | 3 | 4 | 2 | 2 | 2 |

1. Диаграмма сущность-связь автоматически сформированная

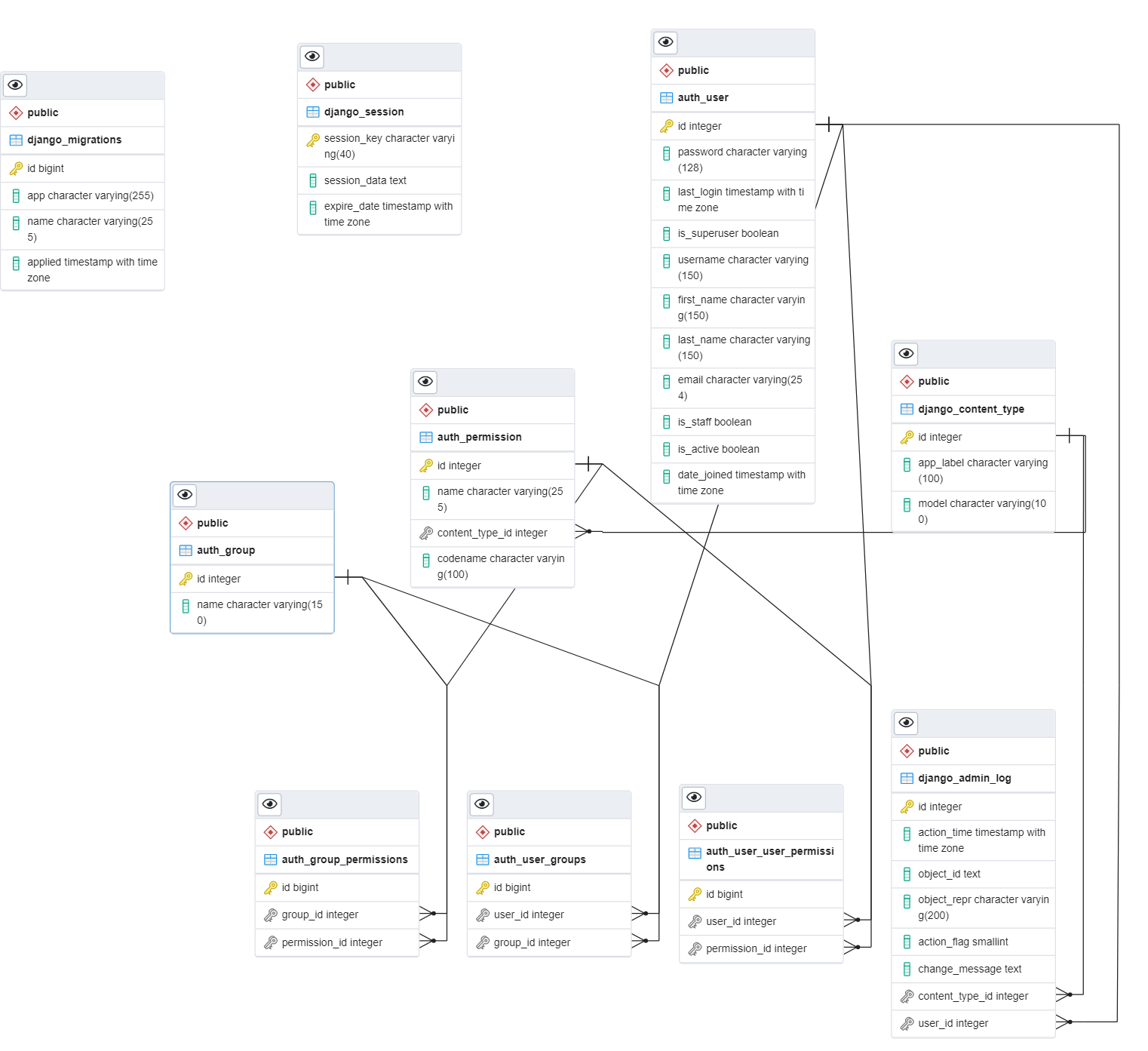


Рисунок Д.1 —Диаграмма сущность-связь автоматически сформированная

1. Код начального создания объектов

Листинг Е.1 — Создание объектов

1. from cls.models import Class
2. from discipline.models import Discipline
3. from teachers.models import \*
4. teachers\_data = [ …
5. {'fio': 'Орлов Георгий Андреевич', 'position': 'учитель физической культуры', 'room': 145},
6. {'fio': 'Петрова Наталья Геннадьевна', 'position': 'учитель-логопед', 'room': 155},
7. {'fio': 'Печенкина Светлана Анатольевна', 'position': 'учитель информатики', 'room': 165}, …
8. ]
9. for data in teachers\_data:
10. try:
11. Teacher.objects.create(fio=data['fio'], position=data['position'], room=data['room'])
12. except Exception as e:
13. print(f"Не удалось создать объект для данных: {data}. Ошибка: {e}")
14. # предметы
15. disciplines\_data = [
16. {'name': 'Математика', 'short\_name': 'Матем.'},
17. {'name': 'Русский язык', 'short\_name': 'Рус.'}, …
18. ]
19. for data in disciplines\_data:
20. try:
21. Discipline.objects.create(\*\*data)
22. except Exception as e:
23. print(f"Не удалось создать объект для данных: {data}. Ошибка: {e}")
24. # классы
25. class\_data = []
26. for digit in range(5, 12):
27. for letter in ['А', 'Б', 'В', 'С', ]:
28. class\_data.append({
29. 'digit': digit,
30. 'letter': letter
31. })
32. for data in class\_data:
33. try:
34. Class.objects.create(\*\*data)
35. except Exception as e:
36. print(f"Не удалось создать объект для данных: {data}. Ошибка: {e}")