Кыргыз Республикасынын Эл аралык Университети

International University of Kyrgyz Republic

Международный Университет Кыргызской Республики

Кафедра Программная инженерия

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на соискание академической степени бакалавра

по направлению: 710400 Программная инженерия.
Специализация: Программная инженерия

Тема: Создание Web приложения "Электронный журнал" для МУКР

Исполнитель **Дуйшеналиев А.Б**

Руководитель **Красниченко Л.С**

Зав. кафедрой **Миркин Е.Л**

Бишкек 2020

Кыргызстан Эл аралык Университети

International University of Kyrghyzstan

Международный Университет Кыргызстана

Кафедра Программная инженерия

«Утверждаю»

заведующий кафедрой

_____

д.т.н., проф. Миркин Е.Л

« » 2020г

ЗАДАНИЕ НА ВЫПУСКНУЮ РАБОТУ

СТУДЕНТУ: Дуйшеналиеву Акиму Басылбековичу

Тема работы: Создание Web приложения "Электронный журнал" для МУКР

Утверждена приказом по МУКР от « 9 » октября 2019г №200

Срок сдачи студентом законченной работы «9» июня 2020г

ИСХОДНЫЕ ДАННЫЕ К РАБОТЕ

1.Литература по веб-ориентированным языкам программирования

2.Интерет источники по Python, Django, HTML, CSS

СОДЕРЖАНИЕ РАБОТЫ (перечень вопросов, подлежащих разработке)

1. **Введение**
2. **Актуальность разработки**
3. **Теоретическая часть**
4. **Практическая часть**

Дата выдачи задания «6» октября 2019г

Руководитель к.ф.м.н., и.о. доцента Красниченко Л.С

Задание принял к исполнению «6» октября 2019

АННОТАЦИЯ

Выпускная работа на тему «Создание Web приложения "Электронный журнал" для МУКР»., посвящена актуальной теме созданию информационного сайта. В ходе выполнения выпускной работы был получен полнофункциональный web-сайт, полностью готовый к применению.

Объём выпускной работы 83 страницs, содержит 2 рисунка, 1 таблица и 6 источников литературы, включая ссылки.

АННОТАЦИЯСЫ

"Кыргызстандын Эл аралык университети үчүн" Электрондук журнал "веб тиркемесин түзүү" темасындагы дипломдук иш, маалыматтык сайтты түзүүнүн актуалдуу темасына арналган. Акыркы жумуштун жүрүшүндө толугу менен иштей турган веб-сайт алынды, ал толугу менен колдонууга даяр.

Акыркы чыгарманын көлөмү 83 беттен турат, 2 сүрөт, 1 таблица жана адабият булактары, ошондой эле шилтемелерди камтыган 6 булактан турат.

ANNOTATION

Graduation work on the topic "Creating a Web Application" Electronic Journal "for the International University of Kyrgyzstan.", Is devoted to the urgent topic of creating an information site. In the course of the final work, a fully functional website was obtained, which is completely ready for use.

The volume of the final work is 83 pages, contains 2 figures, 1 table and 6 sources of literature, including references

Содержание

Введение

Данная WEB система предназначена для облегчения доступа учеников, преподавателей к оценкам по их предметам путем создания user-friendly интерфейса позволяющего выставлять оценки всем ученикам группы сразу.

В данное время WEB приложения занимают все большую и большую часть повседневной жизни и все больше и больше компаний и предприятий отказываются от бумажного хранения информации, в связи с этим переход на «Электронный журнал» является натуральным шагом в эволюции.

Это **оперативность** и **актуальность**. На данный момент в Кыргызстане система электронного журнала является чем то не освоенным, некоторые университеты уже имеют подобную систему но по большей части ей не удобно и не приятно пользоваться, она не представляет интуитивного интерфейса, высокой производительности и даже не приятна на глаз, система выставления оценок сложна и репетативна, таким образом данный проект является довольно актуальным как минимум в пределах Кыргызстана.

Это **объем информации**. Обычно процесс отслеживания оценок это тяжкий и репетативный процесс который не только утомляет ум но и портит зрение т.к человеку приходиться просматривать тонну бумаг, электронный журнал решает эту проблема если не целиком то по большей части, возможность поиска исключает необходимость вручную пролистывать бумаги и напрягать зрение чтобы не пропустить какую деталь таким образом не смотря на объем хранимой информации в базе данных поиск, сравнение, изменение информации никогда не является проблемой.

Для достижения цели необходимо выполнить следующие задачи:

1) Углубленно изучить средства разработки сайта.
2) Разработать структуру сайта с максимально информативным и интуитивно понятным интерфейсом.
3) Создать базу данных содержащую информацию специфичную для рекламного сайта.
4) Разработать панель администратора для коррекции информации базы данных.

Глава 1. Разработка структуры web-системы

1.1 Актуальность работы

Актуальность проекта построена на разработке и внедрении автоматизированной системы работы с оценками, учениками (студентами) к которым они относятся, предметам к которым относятся ученики и преподавателям, к которым относятся предметы.

Чем объясняется актуальность сайта:

- Оперативностью и масштабностью подачи информации широкому кругу пользователей
- Обратной онлайн связью
- Общедоступностью информации

Актуальность создания сайтов очевидна: с появлением глобальной сети каждый человек получил интерактивный инструмент, позволяющий сообщить миру об услугах и товарах компании, привлечь единомышленников и покупателей. Расходы на содержание сайта незначительны и сводятся лишь к платежам за поддержание ресурса в достойном виде.

- Содержимое данного сайта хранится не в виде статичных HTML страничек, находится в базе данных, и отображается «на ленту», по запросу пользователя.
- Достоинство данного web-сайта — это возможность быстрого внесения изменений сразу во все страницы сайта. Одним из преимуществ данного сайта является минимальный переход по страницам что придает максимальную экономию трафика пользователя.
- С помощью HTML, CSS, Django и базы данных PostgreSQL была реализована гибкость и функциональность сайта.

- На сегодняшний день применение баз данных приобрело весьма большое значение для многих организаций, которые для упрощения своей работы применяют компьютерные технологии.

1.2. Цель и задача работы

Цель данной работы – разработать web-систему "Электронный журнал" для МУКР

Главными задачами являлось:

1. Простой и удобный интерфейс позволяющий выставить оценки по указанному предмету всем студентам выбранной группы на одной странице без необходимости скакать по всему сайту

2. Разработать метод связки этого самого “удобного” интерфейса с базой данных так чтобы данные не путались при вводе и выводе

Обеспечить быстродействие системы

3. Организовать базу данных таким образом чтобы данные не путались даже если обрабатываются не алгоритмом, а человеком (в случае если администратор хочет проверить сырую (не обработанную) информацию

Любой информационный-сайт должен создаваться именно для пользователя, но в данном случае информация предоставляется не только пользователю, но и админам коими в данном случае будут является представители деканата и ректората. По этой причине необходимо провести анализ целей и задач при разработке сайта.

С этой целью в первую очередь необходимо определиться не только с составом потенциальных пользователей, но и с **принципами организации работы сайта.** Чёткие ответы на эти вопросы позволят получить реальное представление о том, каким должен быть будущий сайт.

С этой целью необходимо:

- представить себе потребности будущих пользователей;
- Определение информационных тем и их организация.

Сделав анализ состава аудитории сайта, а также техпроцессов, которые будут организованы, следует определиться с его информационной моделью. С этой целью необходимо определить именно те информационные темы, которые могут использоваться для всеобщей группы пользователей, разбив их после этого на отдельные категории.

Следующий этап связан с созданием информационной иерархии сайта и определением не только горизонтальных, но и вертикальных связей, которые должны существовать между некоторыми информационными темами. Полученная структура должна быть проверена на соответствие процессам, которые планируется реализовать впоследствии, чтобы при этом каждый конкретный процесс мог точно вписываться в информационную структуру.

Определение задач создания сайта очень важный этап работ.

- Разработать дизайн;
- Комментарии и обратная связь
- Административная панель;
- Полное управление панелями;

Цель выпускной квалификационной работы – закрепления знаний, полученных при изучении дисциплины, а также получение практических навыков проектирования информационных систем с использованием современных технологий и инструментальных средств.

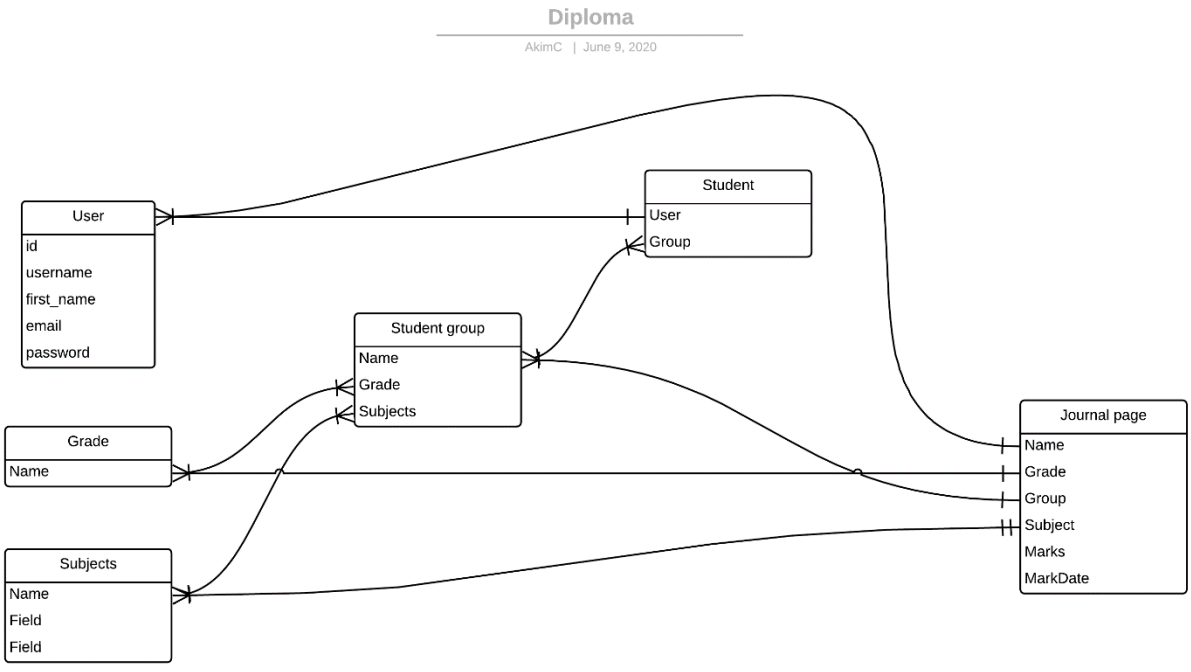
Задачей ВКР является проектирование информационной системы. А также, приобретение практических навыков обследования предметной области, концептуального, логического и физического проектирования базы данных, освоение средств поддержания целостности базы данных, запросов, отчётов.

Глава 2. Выбор инструментальных средств

2.1 Разработка базы данных

Django (что есть фреймфорк для разработки сайтов и WEB приложений написанный на языке Python) предоставляет невероятно простой и интуитивный метод создания базы данных, все что нужно это написать модели и “программировать” их для того чтобы создать валидную структуру базы данных, помимо этого данный метод исключает необходимость учить SQL что уменьшает потенциальное количество человек необходимое для создание сайта/Web приложения и время затраченное на создание сайта.

ED электронного журнала выглядит следующим образом:



2.2 Обработка данных

Обработка всей получаемой информации происходит в файле “views.py” в данном файле происходит обработка представлений так же известных как “вьюшки”, по сути вся работа бэкэнда происходит именно там, во вьюшках происходит сохранение информации в базу данных, извлечение информации из базы данных и передача в HTML шаблон а так же передача информации из HTML форм в базу данных.

2.3 Пользовательский интерфейс

Пользовательский интерфейс целиком и полностью состоит из HTML/CSS, JavaScript не используется за отсутствием необходимости для поставленной цели.

2.4 Python

Python (МФА: [ˈpʌɪθ(ə)n]; в русском языке распространено название питóн) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, обобщенное, объектно-ориентированное, функциональное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Есть реализация интерпретатора для JVM с возможностью компиляции, CLR, LLVM, другие независимые реализации. Проект PyPy использует JIT-компиляцию, которая значительно увеличивает скорость выполнения Python-программ.

Python — активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года. Язык не подвергался официальной стандартизации, роль стандарта де-факто выполняет CPython, разрабатываемый под контролем автора языка. В настоящий момент Python занимает третье место в рейтинге TIOBE с показателем 10,2 %. Аналитики отмечают, что это самый высокий балл Python за все время его присутствия в рейтинге.

Python портирован и работает почти на всех известных платформах — от КПК до мейнфреймов. Существуют порты под Microsoft Windows, практически все варианты UNIX (включая FreeBSD и Linux), Plan 9, Mac OS и macOS, iPhone OS (iOS) 2.0 и выше, iPadOS, Palm OS, OS/2, Amiga, HaikuOS, AS/400 и даже OS/390, Windows Mobile, Symbian и Android.

По мере устаревания платформы её поддержка в основной ветви языка прекращается. Например, с версии 2.6 прекращена поддержка Windows 95, Windows 98 и Windows ME. В версии 3.5 перестала поддерживаться Windows XP, а минимальным требованием является Windows Vista. Однако на устаревших платформах можно использовать предыдущие версии Python — на данный момент сообщество активно поддерживает версии Python начиная от 2.3 (для них выходят исправления).

При этом, в отличие от многих портируемых систем, для всех основных платформ Python имеет поддержку характерных для данной платформы технологий (например, Microsoft COM/DCOM). Более того, существует специальная версия Python для виртуальной машины Java — Jython, что позволяет интерпретатору выполняться на любой системе, поддерживающей Java, при этом классы Java могут непосредственно использоваться из Python и даже быть написанными на Python. Также несколько проектов обеспечивают интеграцию с платформой Microsoft.NET, основные из которых — IronPython и Python.Net.

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем». В Python имеются встроенные типы: булевый, строка, Unicode-строка, целое число произвольной точности, число с плавающей запятой, комплексное число и некоторые другие. Из коллекций в Python встроены: список, кортеж (неизменяемый список), словарь, множество и другие. Все значения являются объектами, в том числе функции, методы, модули, классы.

Добавить новый тип можно либо написав класс (class), либо определив новый тип в модуле расширения (например, написанном на языке C). Система классов поддерживает наследование (одиночное и множественное) и метапрограммирование. Возможно наследование от большинства встроенных типов и типов расширений.

Все объекты делятся на изменяемые и неизменяемые: списки, словари и множества являются изменяемыми, а все остальные — неизменяемыми (например, при изменении строки фактически создаётся новая, а при изменении списка — только меняются ссылки в нём).

Кортеж в Python является, по сути, неизменяемым списком. Во многих случаях кортежи работают быстрее списков, поэтому если вы не планируете

изменять последовательность, то лучше использовать именно их. Неизменяемые объекты (и все объекты в них, если это, например, кортеж) могут быть ключами словаря (должны иметь метод hash).

Появившись сравнительно поздно, Python создавался под влиянием множества языков программирования:

ABC — отступы для группировки операторов, высокоуровневые структуры данных (map) (Python фактически создавался как попытка исправить ошибки, допущенные при проектировании ABC);

Modula-3 — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял Common Lisp);

C, C++ — некоторые синтаксические конструкции (как пишет сам Гвидо ван Россум — он использовал наиболее непротиворечивые конструкции из C, чтобы не вызвать неприязнь у C-программистов к Python[66]);

Smalltalk — объектно-ориентированное программирование;

Lisp, в частности, Scheme — отдельные черты функционального программирования (lambda, map, reduce, filter и другие);

Fortran — срезы массивов, комплексная арифметика;

Miranda — списочные выражения;

Java — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для декораторов;

Icon — генераторы.

Большая часть других возможностей Python (например, байт-компиляция исходного кода) также была реализована ранее в других языках.

Наиболее часто Python сравнивают с Perl и Ruby. Эти языки также являются интерпретируемыми и обладают примерно одинаковой скоростью выполнения программ. Как и Perl, Python может успешно применяться для написания скриптов (сценариев).

Как и Ruby, Python является хорошо продуманной системой для ООП. При этом реализация ООП в Python отличается от многих других объектно-ориентированных языков. В частности:

В отличие от Ruby, Python не придерживается идеологии «всё — объект», и поддерживает встроенные примитивные типы, не входящие в иерархию классов. Такое решение упрощает и делает более технически эффективным межъязыковое взаимодействие, хотя может быть сочтено неудобным фанатами объектного подхода.

В отличие от некоторых ООЯП (Java, Object Pascal, Ruby, …) в Python нет реального общего базового класса, от которого все объекты наследуют общие методы. Хотя формально новый класс в Python наследует (прямо или косвенно) тип object, это является только синтаксическим приёмом, так как методы, которые являются общими для всех объектов — id, type, isinstance, issubclass, str, repr, getattr, … не наследуются от object, а реализованы в виде глобальных функций. Такое решение приводит к тому, что изменение поведения этих методов производится не перегрузкой, а определением специальных методов класса.

В среде коммерческих приложений скорость выполнения программ на Python часто сравнивают с Java-приложениями.

Классический Python имеет общий со многими другими интерпретируемыми языками недостаток — сравнительно невысокую скорость выполнения программ. В некоторой степени ситуацию улучшает сохранение байт-кода (расширения .pyc и, до версии 3.5, .pyo), которое позволяет

интерпретатору не тратить время на синтаксический разбор текста модулей при каждом запуске.

Существуют реализации языка Python, вводящие высокопроизводительные виртуальные машины (ВМ) в качестве бэк-энда компилятора. Примерами таких реализаций может служить PyPy, базирующийся на RPython; более ранней инициативой является проект Parrot. Ожидается, что использование ВМ типа LLVM приведёт к тем же результатам, что и использование аналогичных подходов для реализаций языка Java, где низкая вычислительная производительность в основном преодолена. Однако нельзя забывать, что динамический характер Python делает неизбежными дополнительные накладные расходы при исполнении программ, что ограничивает производительность Python-систем независимо от применяемых технологий. Вследствие этого для написания критических участков кода используются низкоуровневые языки, интеграция с которыми обеспечивается множеством программ и библиотек (см. выше).

В самой популярной реализации языка Python интерпретатор довольно велик и более требователен к ресурсам, чем в аналогичных популярных реализациях Tcl, Forth, LISP или Lua, что ограничивает его применение во встроенных системах. Тем не менее, Python нашёл применение в КПК и некоторых моделях мобильных телефонов.

2.5 Django

Django — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других

(например, Ruby on Rails). Один из основных принципов фреймворка — DRY (англ. Don't repeat yourself)

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Веб-фреймворк Django используется в таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, Pinterest, YouTube, Google и др.

Также Django используется в качестве веб-компонента в различных проектах, таких как Graphite — система построения графиков и наблюдения, FreeNAS — свободная реализация системы хранения и обмена файлами и др.

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (англ. View), а презентационная логика Представления реализуется в Django уровнем Шаблонов (англ. Template). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Первоначальная разработка Django как средства для работы новостных ресурсов достаточно сильно отразилась на его архитектуре: он предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Так, например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать,

изменять и удалять любые объекты наполнения сайта, протоколируя все совершённые действия, и предоставляет интерфейс для управления пользователями и группами (с пообъектным назначением прав).

В дистрибутив Django также включены приложения для системы комментариев, синдикации RSS и Atom, «статических страниц» (которыми можно управлять без необходимости писать контроллеры и представления), перенаправления URL и другое.

Некоторые возможности Django:

- ORM, API доступа к БД с поддержкой транзакций
- встроенный интерфейс администратора, с уже имеющимися переводами на многие языки
- диспетчер URL на основе регулярных выражений
- расширяемая система шаблонов с тегами и наследованием
- система кеширования
- интернационализация
- подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты «generic views» — шаблоны функций контроллеров авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и проч.
- Система фильтров («middleware») для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий
- библиотека для работы с формами (наследование, построение форм по существующей модели БД)

- встроенная автоматическая документация по тегам шаблонов и моделям данных, доступная через административное приложение

Некоторые компоненты фреймворка между собой связаны слабо, поэтому их можно достаточно просто заменять на аналогичные. Например, вместо встроенных шаблонов можно использовать Mako или Jinja.

В то же время заменять ряд компонентов (например, ORM) довольно сложно.

Помимо возможностей, встроенных в ядро фреймворка, существуют пакеты, расширяющие его возможности. Возможности, предоставляемые пакетами, а также полный перечень пакетов удобно отслеживать через специальный ресурс — www.djangopackages.com.

Django проектировался для работы под управлением Apache с модулем mod python и с использованием PostgreSQL в качестве базы данных.

С включением поддержки WSGI, Django может работать под управлением FastCGI, mod wsgi, или SCGI на Apache и других серверах (lighttpd, nginx,…), сервера uWSGI.

В настоящее время, помимо базы данных PostgreSQL, Django может работать с другими СУБД: MySQL, SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere и Oracle.

2.6 Python Decouple

Python Decouple – это библиотека Python, которая позволяет разработчикам отделять параметры конфигурации от кода. Первоначально разработанная для Django, теперь она является универсальным инструментом Python для хранения параметров и определения постоянных значений отдельно от кода приложения.

Глава 3. Конструкторская часть

3.1 Этапы разработки проекта

На сегодняшний день существуют несколько этапов разработки веб-сайта:

Проектирование сайта или веб-приложения (сбор и анализ требований, разработка технического задания, проектирование интерфейсов);

1. Разработка креативной концепции сайта;
2. Создание дизайн-концепции сайта;
3. Создание макетов страниц;
4. Создание мультимедиа и FLASH-элементов;
5. Вёрстка страниц и шаблонов;
6. Программирование (разработка функциональных инструментов) или интеграция в систему управления содержимым (CMS);
7. Оптимизация и размещение[уточнить] материалов сайта;
8. Тестирование и внесение корректировок;
9. Публикация проекта на хостинге;
10. Обслуживание работающего сайта или его программной основы.
11. В зависимости от текущей задачи, какие-то из этапов могут отсутствовать.

3.1.1 Создание технического задания

Составлением технического задания могут заниматься проектировщик, аналитик, веб-архитектор, менеджер проекта вместе или по отдельности.:В случае, когда сайт разрабатывается фрилансером, техническое задание может быть составлено со стороны компании заказчика).

Работа с заказчиком начинается с заполнения брифа, в котором заказчик излагает свои пожелания относительно визуального представления и структуры сайта, указывает на ошибки в старой версии сайта, приводит примеры сайтов конкурентов. Исходя из брифа, менеджер составляет техническое задание, учитывая возможности программных и дизайнерских средств. Этап заканчивается после утверждения технического задания заказчиком. Важно сразу отметить, что этапы проектирования веб-сайтов зависят от многих факторов, таких как объём сайта, функциональность, задачи, которые должен выполнять будущий ресурс и многое другое. Однако, есть несколько этапов, которые в обязательном порядке присутствуют в планировании любого проекта.:В результате в документе, где описано техническое задание, могут быть следующие основные разделы:

- Цели и назначение сайта.
- Аудитория сайта.
- Технические характеристики.
- Содержание сайта (структура сайта с подробным описанием элементов и функций каждой страницы).
- Интерактивные элементы и сервисы (формы обратной связи, поиск на сайте, форум на сайте).
- Формы (отправки на почту, подписки на рассылку, обратной связи).
- Система управления содержимым (контентом).
- Требования к материалам.
- Перенос на хостинг.
- Дизайн основной и типовых страниц сайта

Начинается работа с создания дизайна, обычно в графическом редакторе. Дизайнер создаёт один или несколько вариантов дизайна, в соответствии с техническим заданием. При этом отдельно создаётся дизайн главной страницы,

и дизайны типовых страниц (например: статьи, новости, каталог продукции). Собственно «дизайн страницы» представляет собой графический файл, слоеный рисунок, состоящий из наиболее мелких картинок-слоев элементов общего рисунка.

При этом дизайнер должен учитывать ограничения стандартов HTML (не создавать дизайн, который затем не сможет быть реализован стандартными средствами HTML). Исключение составляет Flash-дизайн.

3.1.2 HTML-верстка

Утверждённый дизайн передаётся HTML-верстальщику, который «нарезает» графическую картинку на отдельные рисунки, из которых впоследствии складывает HTML-страницу. В результате создаётся код, который можно просматривать с помощью браузера. А типовые страницы впоследствии будут использоваться как шаблоны.

3.1.3 Программирование

Далее готовые HTML-файлы передают программисту. Программирование сайта может осуществляться как «с нуля», так и на основе CMS — системы управления сайтом. Веб-разработчики часто называют CMS «движком».

В случае с CMS надо сказать, что сама «CMS» в некотором смысле это готовый сайт, состоящий из заменяемых частей. «Программист» — в данном случае правильно будет назвать его просто специалистом по CMS — должен заменить стандартный шаблон, поставлявшийся с CMS, на оригинальный

шаблон. Этот оригинальный шаблон он и должен создать на основе исходного «веб-дизайна».

При программировании сайта специалисту назначаются контрольные точки сроков.

3.1.4 Тестирование

Процесс тестирования может включать в себя самые разнообразные проверки: вид страницы с увеличенными шрифтами, при разных размерах окна браузера, при отсутствии флэш-плеера и многие другие. Также — юзабилити-тестирование.

Обнаруженные ошибки отправляются на исправление до тех пор, пока не будут устранены. Сроки контролирует менеджер проекта. Также, на этом этапе привлекают к работе дизайнера, чтобы он провёл авторский надзор.

3.1.5 Размещение сайта в Интернет

Файлы сайта размещают на сервере провайдера (хостинга) и производят нужные настройки. На этом этапе сайт пока закрыт для посетителей.

Деплой — процесс выкладки новой версии сайта на сервер (или сервера). Этот процесс может быть довольно сложным и сильно зависит от используемых технологий. Во время деплоя выполняются следующие задачи (ниже всего лишь один из возможных вариантов, причём довольно примитивный):

- Код проекта скачивается на сервер (обычно через клонирование Git)
- Ставятся все необходимые зависимости
- Выполняется процесс сборки, например собирается фронтенд-часть
- Выполняются миграции. Миграции — SQL-скрипты, которые изменяют структуру базы данных

- Запускается новая версия кода

Как это ни странно, но во многих компаниях прямо сейчас весь этот процесс выполняется руками. Программист заходит на сервер, запускает git pull и далее проходится по списку выше. Это худший способ деплоить. Деплой относится к тем задачам, которые должны быть автоматизированы от и до.

Несмотря на разнообразие способов деплоя, есть одно важное правило общее для всех — деплоить можно только вперёд! Деплой нельзя «откатывать» (в первую очередь это касается миграций, но про базы мы пока не говорим). Если после или во время деплоя что-то пошло не так, то правильно деплоить снова, но предыдущую версию.

Кроме того, деплои можно классифицировать по способу обновления и отката:

Последовательное обновление — сервера обновляются по очереди

Сине-Зелёный деплой — полное дублирование инфраструктуры с подменой

3.1.6 Наполнение контентом и публикация

Сайт наполняют содержимым (контентом) — текстами, изображениями, файлами для скачивания и так далее. Иногда тексты составляются специалистом студии, иногда контентом занимается ответственное лицо со стороны заказчика. Это решается на этапе составления технического задания. В случае, если контент составляется представителем студии, то это происходит и утверждается параллельно с другими этапами проекта. На каждой странице находятся текстовые блоки, они могут быть типовыми (стандартные) и не типовыми. Как правило нетиповой текстовый блок расположен на странице 404.

К стандартным текстовым блокам относятся:

1. header сайта;
2. footer сайта;
3. навигационная цепочка, или "хлебные крошки".
4. Основные элементы текстового блока:
5. заголовки 1, 2 и 3 уровней;
6. изображения;
7. изображения в тексте;
8. галереи;
9. текст;
10. раскрывающийся блок текста, который содержит заголовок;
11. нумерованные и ненумерованные списки;
12. таблицы;
13. файлы для скачивания;
14. видео.

3.1.7 Внутренняя SEO-оптимизация

Связана с некоторыми изменениями самого сайта. SEO-оптимизация начинается с определения семантического ядра. Здесь определяются такие ключевые слова, которые привлекут наиболее заинтересованных посетителей, по которым выиграть конкуренцию проще. Затем эти слова вносятся на сайт. Тексты, ссылки, другие теги адаптируются так, чтобы поисковые системы могли их успешно находить по ключевым словам.

3.1.8 Внешняя SEO-оптимизация

Сводится, как правило, к построению структуры входящих ссылок. Это, собственно, и есть раскрутка сайта. К разработке сайта внешняя SEO-оптимизация не имеет отношения. SEO-оптимизация классифицируется на

«белую» и «черную» (такую, после которой сайт за две недели попадает в топ, а потом в бан поисковиков). Настоящая, «белая» SEO-оптимизация, это трудоёмкий и долгий процесс, стоимость которого может в несколько раз превышать расходы на создание сайта.

3.1.9 Сдача проекта

Заказчик или его доверенное лицо просматривают готовый проект и в случае, если все устраивает, то подписывают документы о сдаче проекта.

Также, на этом этапе производится обучение представителя заказчика навыкам работы в администраторской зоне сайта.

3.2. Разработка пользовательского интерфейса

3.2.1 Интерфейс сайта

Пользовательский интерфейс – комплекс мер, который подводит пользователей к совершению определенного действия.

Интерфейс сайта — это то, с чем взаимодействует пользователь. Это не просто графическое оформление в определенном стиле, а в первую очередь инструмент и порядок работы посетителя с той или иной функцией на сайте. Порядок расположения функциональных блоков сайта, способствующий совершению определенных действий пользователем.

Проектирование интерфейса — очень важный этап создания сайта. Без интерфейса невозможно начинать верстку и программирование. Необходимо проектировать интерфейс, имея все необходимые исходные данные по проекту: портрет пользователей, их роли, представление о функциях проекта.

Разные страницы содержат контент разного типа. Созданы элементы управления, которые упрощают пользователю работу с сайтом.

Данный интерфейс простой в использовании, эффективный и оперативный, он отлично справляться со своей задачей.

Пользовательский интерфейс — это инструмент управления. Он предоставляет доступ к различным функциям web-сайта. Данный интерфейс дает возможность пользователю с наименьшими усилиями выполнить интересующее его действие.

Дизайн сайта, расположение функциональных блоков, содержание и расположение контента производится таким образом, что пользователь подталкивается к совершению необходимого действия: звонок, написание комментария, совершение покупки, заказ и т.д. Стоит понимать, что поведение пользователей никак не корректируется и не изменяется. Трансформации подвергается сам сайт.

Чтобы разработка интерфейса, необходимо четко поставить цели и определить результат, который необходимо достигнуть. Специалисты не просто работают над тем, чтобы оформление сайта нравилось пользователям. Каждый конкретный проект должен подводить пользователей к совершению требуемых действий: регистрация на сайте, звонок, совершение покупки. Поэтому проектирование всегда осуществляется с учетом специфики бизнеса заказчика и тех целей, которые необходимо достигнуть.

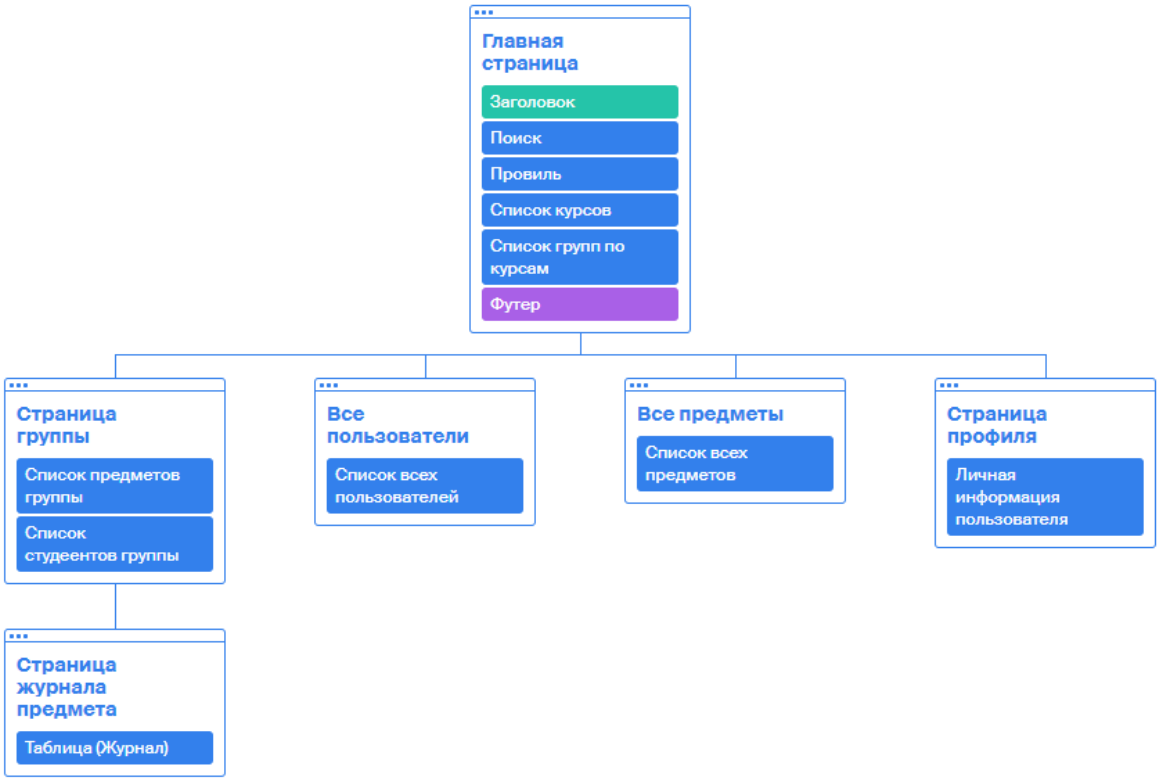
Какие особенности имеет пользовательский интерфейс?

Поскольку пользователь при попадании на сайт должен совершить определенные действия, то все, и дизайн, и контент сайта, работают только для достижения определенной цели.

- На сайте нет никакой лишней информации, которая способна отвлечь пользователя от его действий. Нет лишних баннеров, рубрик, разделов.
- Дизайн отличается высокой информативностью.

- Расположение элементов имеет свой определенный порядок.
- Логичность и простота расположения функциональных блоков.

3.2.2 Структура сайта

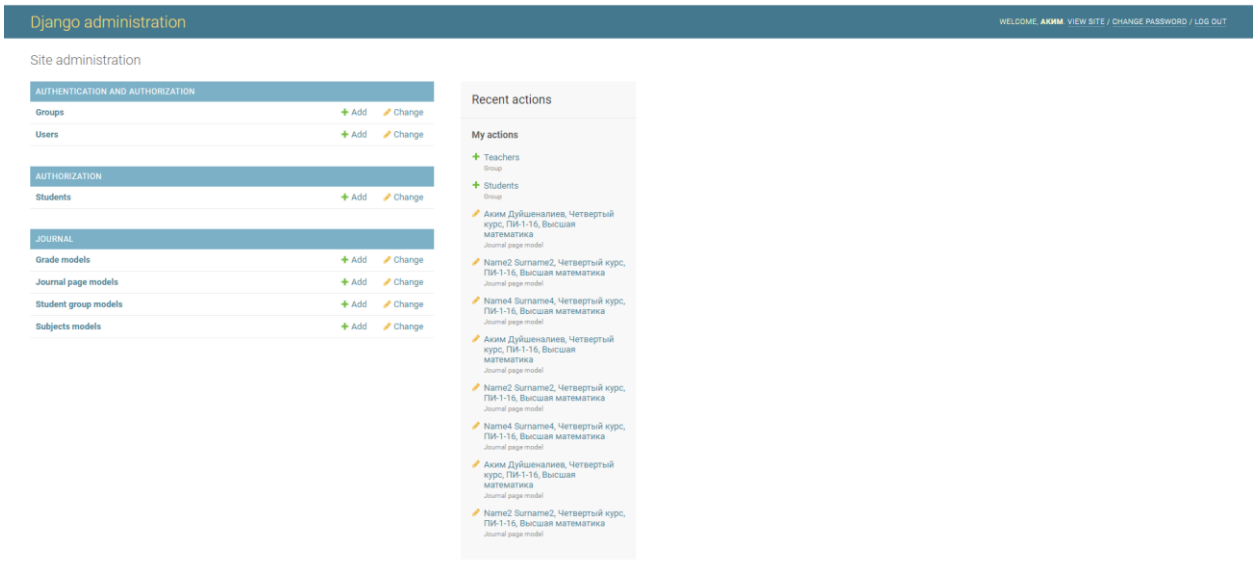


3.3. Разработка административной части

Django предоставляет широкий спектр возможностей касательно админ панели, одной из которых является автоматическая генерация самой панели, таким образом программист способен вывести в админ панель всю базу данных в организованном и понятном виде

На данный момент в админ панели данного проекта присутствуют разделы Пользователей, оценок, групп студенов, предметов, главной страницы журнала, курса, должности, которую занимает пользователь (админ, учитель, студент)

3.2.1 Интерфейс административной панели



3.3.2 Изменение и расширение информационной системы

В ходе разработки база данных много кратно меняла свои очертания пока не достигла оптимального состояния, база данных началась с одной модели пользователя и расширилась на модель групп, оценок, студентов и т.

Заключение

В данном проекте уже было изучено много новых концептов и применено много новых вариантов использования фреймворка для достижения желаемого результата

В ходе выполнения были исследованы пути создания процедурно генерируемой таблицы, отправление информации из таблицы во фреймворк, обработка полученной информации, сохранение обработанной информации в базу данных в необходимом порядке и вытаскивание информации из базы данных в необходимом порядке а так же отправка информации в таблицу.

Список используемых литератур

1. https://docs.djangoproject.com/en/3.0/
2. https://stackoverflow.com/
3. https://www.w3schools.com/python/default.asp
4. https://ru.wikipedia.org/wiki/Python
5. https://ru.wikipedia.org/wiki/Django
6. https://ru.wikipedia.org/wiki/Веб-разработка

Приложения

Система технически имеет 2 приложения (2 apps) “Authorization” и “Journal”, первая отвечает за регистрацию и авторизацию пользователей, вторая за главный функционал коим является все что связано с выставлением оценок и отображением их на сайте.

Authorization

templates

allSubjects.html

Данная страница показывает список всех предметов сохраненных в базе данных.

```
{% extends "Authorization/base.html" %}
{% load crispy_forms_tags %}

{% block content %}
<div class="container content-section bg-secondary shadow-sm boxshadow px-0">

   <h1 class="text-center my-3">Список всех предметов</h1>

   <table class="table table-bordered">
      {% for subject in subjects %}
         <tr>
            <th>{{ subject }}</th>
         </tr>
      {% endfor %}
   </table>
</div>
{% endblock content %}
```

allUsers.html

Данная страница отображает список всех пользователей сохраненных в базе данных независимо от их статуса.

```
{% extends "Authorization/base.html" %}
{% load crispy_forms_tags %}

{% block content %}
<div class="container table-responsive content-section bg-secondary shadow-sm boxshadow">

   <h1 class="text-center my-3">Список всех пользователей</h1>

   <table class="table table-bordered">
      <tr>
         <th>ФИО</th>
         <th>Группа</th>
         <th>Курс</th>
      </tr>
      {% for user, group, grade in usersInfo %}
         <tr>
            <th>{{ user }}</th>
            <th><a href="{% url 'Journal:GroupPage' group_slug=group %}">{{ group }}</a></th>
            <th>{{ grade }}</th>
         </tr>
      {% endfor %}
   </table>
</div>
{% endblock content %}
```

base.html

Это «базовая» страница, она никогда не отображается сама по себе но всегда присутствует на любой другой странице, это сделано для того чтобы не писать один и тот же код на каждой странице, плюс к этому таким образом все стили что были подключены на этой странице можно использовать на любой другой странице которая использует эту как базу.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>{{ title }}</title>

<!-- Bootstrap styles -->
   <link rel="stylesheet" href="{% static 'Authorization/css/bootstrap.min.css' %}" type="text/css">
   <link rel="stylesheet" href="{% static 'Authorization/css/custom-classes.css' %}" type="text/css">

</head>
<body>

<!-- Start navbar -->
<nav class="navbar navbar-light bg-faded justify-content-between flex-nowrap flex-row bottomshadow">
   <div class="container w-100">
      <a href="/" class="navbar-brand float-left">Главная</a>
```

```
        <ul class="nav navbar-nav flex-row float-left">
            <li class="nav-item"><a class="nav-link pr-3" href="{% url 'allUsers' %}">Все пользователи</a></li>
            <li class="nav-item"><a class="nav-link" href="{% url 'allSubjects' %}">Все предметы</a></li>
            <li class="nav-item pl-5">
            <form method="GET" class="form-inline my-2 my-lg-0" action="{% url 'search' %}">
                <input class="form-control mr-sm-2" type="search" name="search" aria-label="Search">
                <button class="btn btn-outline-body my-2 my-sm-0" type="submit">Найти</button>
            </form>
            </li>

        </ul>
        <ul class="nav navbar-nav flex-row float-right">
            {% if user.is_authenticated %}
                <li class="nav-item"><a class="nav-link pr-3" href="{% url 'profile' %}">{{ user.first_name }} {{ user.last_name }}</a></li>
                <li class="nav-item"><a class="nav-link pr-3" href="{% url 'logout' %}">Выйти</a></li>
            {% else %}
                <li class="nav-item"><a class="nav-link pr-3" href="{% url 'login' %}">Войти</a></li>
                <li class="nav-item"><a class="nav-link" href="{% url 'register' %}">Зарегистрироваться</a></li>
            {% endif %}
        </ul>
    </div>
</nav>
```

```
<!-- End navbar -->

    {% if messages %}
        {% for message in messages %}
            <div class="alert alert-{{ message.tags }}">
                {{ message }}
            </div>
        {% endfor %}
    {% endif %}

<br>

{% if search %}
    {% if result %}
        <div class="container">
            <ul>
                {% for item in result %}
                    <li><a href="#">{{ item }}</a></li>
                {% endfor %}
            </ul>
        </div>
    {% else %}
        <div class="container text-center">
            <p>Не удалось ничего найти по запросу "{{ search }}"</p>
        </div>
    {% endif %}
{% else %}
```

```
{% endif %}


<!-- Start block -->
   {% block content %}     {% endblock %}
<!-- End block -->


<!-- Footer -->
<link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.3/css/font-awesome.min.css" rel="stylesheet"
     integrity="sha384-T8Gy5hrqNKT+hzMclPo118YTQO6cYprQmhrYwIiQ/3axmI1hQomh7Ud2hPOy8SP1" crossorigin="anonymous">
<section class="footers bg-light pt-1 pb-3">
<hr>
  <div class="container pt-5">
     <div class="row">
        <div class="col-xs-12 col-sm-6 col-md-4 footers-one">
                 <div class="footers-logo">
               <h4>Здесь будет лого</h4>
                    <!-- <img src="" alt="Logo" style="width:120px;"> -->
                 </div>
                 <div class="footers-info mt-3">
                    <p>Cras sociis natoque penatibus et magnis Lorem Ipsum tells about the compmany right now the best.</p>
                 </div>
                 <div class="social-icons">
           <a href="https://www.facebook.com/"><i id="social-fb" class="fa fa-facebook-square fa-2x social"></i></a>
           <a href="https://twitter.com/"><i id="social-tw" class="fa fa-twitter-square fa-2x social"></i></a>
```

```
                <a href="https://plus.google.com/"><i id="social-gp" class="fa fa-
google-plus-square fa-2x social"></i></a>
                <a href="mailto:bootsnipp@gmail.com"><i id="social-em" class="fa
fa-envelope-square fa-2x social"></i></a>
            </div>
             </div>
          <div class="col-xs-12 col-sm-6 col-md-2 footers-two">
              <h5>Главное</h5>
              <ul class="list-unstyled">
            <li><a href="#">Искать</a></li>
                  </ul>
            </div>
          <div class="col-xs-12 col-sm-6 col-md-2 footers-three">
              <h5>Информация</h5>
              <ul class="list-unstyled">
            {% if user.is_authenticated %}
            {% else %}
              <li><a href="{% url 'register' %}">Зарегистрироваться</a></li>
            {% endif %}
            <li><a href="#">Инструкция</a></li>
            <li><a href="#">О нас</a></li>
          </ul>
              </div>
          <div class="col-xs-12 col-sm-6 col-md-2 footers-four">
              <h5>Исследуйте</h5>
              <ul class="list-unstyled">
            <li><a href="#">Обратная связь</a></li>
            <li><a href="#">Пользовательское соглашение</a></li>
                  </ul>
```

```
                </div>
            <div class="col-xs-12 col-sm-6 col-md-2 footers-five">
                    <h5>Компания</h5>
                    <ul class="list-unstyled">
                <li><a href="#">Условия пользования</a></li>
                <li><a href="#">Соглашение</a></li>
                        </ul>
                </div>
      </div>
  </div>
</section>
<section class="copyright border">
    <div class="container">
        <div class="row text-center">
            <div class="col-md-12 pt-3">
                <p class="text-muted">© 2020 Что-то что-то, я один человек но тут может что то быть.</p>
            </div>
        </div>
    </div>
</section>
<!-- Footer end -->
<!-- Bootstrap scripts -->
        <script src="{% static 'Authorization/js/jquery-3.4.1.min.js' %}"></script>
        <script src="{% static 'Authorization/js/bootstrap.min.js' %}"></script>

</body>
</html>
```

login.html

Это страница, на которой пользователь может войти в аккаунт.

```
{% extends "Authorization/base.html" %}
{% load crispy_forms_tags %}

{% block content %}
  <div class="container content-section bg-secondary boxshadow">
    <form method="POST">
      {% csrf_token %}
      <fieldset class="form-group">
        <legend class="border-bottom mb-4">Войти в аккаунт.</legend>
        {{ form|crispy }}
        <div class="form-group">
          <button class="btn btn-outline-info" type="submit">Войти</button>
        </div>
      </fieldset>
    </form>
    <div class="border-top pt-3">
      <small class="text-muted">
      Еще нет аккаунта? <a class="ml-2" href="{% url 'register' %}">Зарегистрироваться</a>
      </small>
    </div>
  </div>
{% endblock content %}
```

logout.html

Это простая страница, которая показывается, когда пользователь вышел из аккаунта.

```
{% extends "Authorization/base.html" %}


{% block content %}
   <div class="container content-section bg-secondary">
      <h1>Вы вышли из аккаунта!</h1>
      <div class="border-top pt-3">
         <small class="text-muted">
         Войти обратно в аккаунт? <a href="{% url 'login' %}">Войти</a>
         </small>
      </div>
   </div>
{% endblock content %}
```

profile.html

На этой странице отображаются все необходимые данные о пользователе, который на данные момент залогинен.

```
{% extends "Authorization/base.html" %}
{% load crispy_forms_tags %}

{% block content %}
  <div class="container boxshadow p-0">
    <h1 class="text-center py-3">{{ user.first_name }} {{ user.last_name }}</h1>

    <div class="container p-0">
      <table class="table table-bordered">
        <tr>
          <td>Имя</td>
          <td>{{ user.first_name }}</td>
        </tr>
        <tr>
          <td>Фамилия</td>
          <td>{{ user.last_name }}</td>
        </tr>
        <tr>
          <td>Группа</td>
          <td><a href="{% url 'Journal:GroupPage' group_slug=userGroup %}">{{ userGroup }}</a></td>
        </tr>
        <tr>
          <td>Курс</td>
          <td>{{ userGrade }}</td>
```

```
            </tr>
            {% for subject in userSubjectList %}
              <tr>
                {% if forloop.counter == 1 %}
                  <td class="align-middle" rowspan="{{ userSubjectList|length
}}">Предметы</td>
                  <td>{{ subject }}</td>
                {% else %}
                  <td>{{ subject }}</td>
                {% endif %}
              </tr>
            {% endfor %}
          </table>


      </div>
    </div>
{% endblock content %}
```

register.html

Это страница регистрации, где новый ученик может создать аккаунт (учетные записи учителей и администраторов создаются лично другими администраторами в целях безопасности.)

```
{% extends "Authorization/base.html" %}
{% load crispy_forms_tags %}

{% block content %}
   <div class="container content-section bg-secondary boxshadow">
      <form method="POST">
         {% csrf_token %}
         <fieldset class="form-group">
            <legend class="border-bottom mb-4">Создать новый
аккаунт.</legend>
            {{ form|crispy }}
            <div class="form-group">
               <button class="btn btn-outline-info"
type="submit">Зарегистрироваться</button>
            </div>
         </fieldset>
      </form>
      <div class="border-top pt-3">
         <small class="text-muted">
         Уже есть аккаунт? <a class="ml-2" href="{% url 'login' %}">Войти</a>
         </small>
      </div>
   </div>
{% endblock content %}
```

admin.py

Данный файл проводит авторизацию моделей в админ панель, без этого созданная модель не будет отображена в панели администратора.

```
from django.contrib import admin

from .models import *

@admin.register(Student)
class StudentAdminForm(admin.ModelAdmin):
    list_display = ('full_name_display', 'group_name_display', 'cours_display')
    list_filter = ('group',)

    def full_name_display(self, obj):
        return '{} {}'.format(obj.user.first_name, obj.user.last_name)

    def group_name_display(self, obj):
        return '{}'.format(obj.group.name)

    def cours_display(self, obj):
        return '{}'.format(obj.group.grade.name)
```

forms.py

Данный файл содержит формы используемые на страницах регистрации и авторизации, Django далее конвертирует эти формы в HTML формы и автоматически создает необходимую структуру, таким образом программисту не нужно париться с созданием HTML формы и можно сразу приступить к сохранению информации в базу данных.

```
from django import forms
from django.contrib import admin
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm

from .models import *



class UserRegisterForm(UserCreationForm):
    email = forms.EmailField(required=False)

    class Meta:
        model = User
        fields = ['username', 'first_name', 'last_name', 'email', 'password1',
'password2']


class UserAuthenticationForm(forms.Form):
    username = forms.CharField(max_length=25)
    password = forms.CharField(max_length=32, widget=forms.PasswordInput)
```

models.py

В данном файле создаются «модели», которые далее мигрируются в базу данных что создает в ней таблицы необходимого вида.

```
from django.db import models, transaction
from django.contrib.auth.base_user import BaseUserManager

from django.contrib.auth.models import User
from django.contrib.auth.models import Group

from Journal.models import *



class Student(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    group = models.ForeignKey(StudentGroupModel,
on_delete=models.CASCADE)


    def __str__(self):
        name = self.user.first_name
        surname = self.user.last_name
        grade = self.group.grade.name
        group = self.group.name
        fullinfo = name + ' ' + surname + ', ' + grade + ', ' + group
        return fullinfo
```

urls.py

Данная страница отвечает за рутирование, здесь можно создать все “урлы” (URL адреса) для данного приложения путем указания вьюшки, к которой надо создать урл и дальше указать само имя по которому к ней нужно обращаться в строке электронного адреса

```
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

from .views import *

urlpatterns = [
    path(r'search-result/$', search, name='search'),
    path('register', register, name='register'),
    path('login',
auth_views.LoginView.as_view(template_name='Authorization/login.html'),
name='login'),
    path('logout',
auth_views.LogoutView.as_view(template_name='Authorization/logout.html'),
name='logout'),
    path('profile', profile, name='profile'),
    path('allUsers',
AllUsersView.as_view(template_name='Authorization/allUsers.html'),
name='allUsers'),
    path('allSubjects',
AllSubjectsView.as_view(template_name='Authorization/allSubjects.html'),
name='allSubjects'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

views.py

Этот файл отвечает за “вьюшки”, вьюшка говорит фреймворку с какой страницей она работает и что нужно делать в этой странице, например функция “register” здесь работает с “ register.html” и сохраняет информацию, полученную из формы в базу данных.

```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.views.generic import TemplateView
from django.db.models.functions import Concat
from django.db.models import Q, Value

from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User

from Journal.models import *
from .forms import *


def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, f'Аккаунт зарегистрирован!')
            return redirect('login')
    else:
        form = UserRegisterForm()
```

```
    return render(request, 'Authorization/register.html', {'form': form, 'title':
'Регистрация'})



@login_required
def profile(request):
    profileTemplateName = 'Authorization/profile.html'
    currentUser = request.user.pk


    currentUserGroup = Student.objects.filter(user=currentUser).values('group')
    currentUserGroup = currentUserGroup[0]['group']


    currentUserGroupName =
StudentGroupModel.objects.filter(pk=currentUserGroup).values('name')
    currentUserGroupName = currentUserGroupName[0]['name']


    currentUserGrade =
StudentGroupModel.objects.filter(pk=currentUserGroup).values('grade')
    currentUserGrade =
GradeModel.objects.filter(pk=currentUserGrade[0]['grade']).values('name')
    currentUserGrade = currentUserGrade[0]['name']


    currentUserSubjects =
StudentGroupModel.objects.filter(pk=currentUserGroup).values('subjects')



    currentUserSubjectsList = [None] * len(currentUserSubjects)


    for _ in range(len(currentUserSubjects)):
```

```
        temp =
SubjectsModel.objects.filter(pk=currentUserSubjects[_]['subjects']).values('name')
        temp = temp[0]['name']
        currentUserSubjectsList[_] = temp


    context = {
        'userGroup': currentUserGroupName,
        'userGrade': currentUserGrade,
        'userSubjectList': currentUserSubjectsList,
    }

    return render(request, profileTemplateName, context)

class AllUsersView(TemplateView):
    template_name = 'Authorization/allUsers.html'

    def get(self, request):

        ### Variables
        usernames = User.objects.all()
        usersId = User.objects.filter().values('id')  # Get all user id
        userGroups = [None] * len(usernames)
        usersGrade = [None] * len(usernames)
        user_list = [None] * len(usernames)



        for _ in range(len(usernames)):
            num = _ + 1
```

```
        ### Get first and last name, combine them and put into a list
        temp =
User.objects.filter(username__icontains=usernames[_]).values('first_name',
                                                'last_name')  # Get first name and last
name
        temp = temp.__getitem__(0)  # Rewrite to make it regular dict


        firstName = temp.get('first_name')
        lastName = temp.get('last_name')
        fullName = firstName + ' ' + lastName  # Combine to make a single string
with first and last names


        user_list[_] = fullName  # Write full name into list on a position



        temp = Student.objects.filter(user=num).values('group')
        temp = temp.__getitem__(0)
        temp = temp.get('group')


        temp = StudentGroupModel.objects.filter(pk=temp).values_list('name',
'grade')
        tempName = temp[0][0]
        userGroups[_] = tempName


        tempGrade = temp[0][1]
        tempGrade = GradeModel.objects.filter(pk=tempGrade).values_list('name')
        tempGrade = tempGrade[0][0]
        usersGrade[_] = tempGrade
```

```
        context = {
            'title': 'Пользователи',
            'usersInfo': zip(
                user_list,
                userGroups,
                usersGrade,
            ),
        }

        return render(request, self.template_name, context)

class AllSubjectsView(TemplateView):
    template_name = 'Authorization/allSubjects.html'

    def get(self, request):

        subjectsRaw = SubjectsModel.objects.filter().values_list('name')
        subjects = [None] * len(subjectsRaw)

        for _ in range(len(subjectsRaw)):
            temp = subjectsRaw[_][0]
            subjects[_] = temp

        context = {
            'title': 'Предметы',
            'subjects': subjects,
```

```
        }

        return render(request, self.template_name, context)


def search(request):
    template_name = 'Authorization/base.html'

    query = request.GET.get('search')
    result = []

    groupResultRaw =
StudentGroupModel.objects.filter(Q(name__icontains=query)).values_list('name')
    nameResultRaw = User.objects.annotate(fullname=Concat('first_name', Value('
'), 'last_name'))
    nameResultRaw = nameResultRaw.filter(fullname__icontains=query)



    if len(groupResultRaw) > 0 and len(nameResultRaw) == 0:
        result = [None] * len(groupResultRaw)
        for _ in range(len(groupResultRaw)):
            temp = groupResultRaw.__getitem__(_)  # rewrite them to make a regular
dict
            result[_] = temp[0]  # Write all id's into the list as regular strings
    elif len(groupResultRaw) == 0 and len(nameResultRaw) > 0:
        result = [None] * len(nameResultRaw)
        for _ in range(len(nameResultRaw)):
            temp = nameResultRaw.__getitem__(_)  # rewrite them to make a regular
dict
```

```
        temp = User.objects.filter(username=temp).values('first_name', 'last_name')
        temp = temp.__getitem__(0)
        firstName = temp.get('first_name')
        lastName = temp.get('last_name')
        fullName = firstName + ' ' + lastName
        result[_] = fullName  # Write all id's into the list as regular strings


context = {
    'search': query,
    'result': result,
}
return render(request, template_name, context)
```

Journal

templates

groupPage.html

Данная страница отображает информацию о выбранной группе, т.е она показывает предметы этой группы и список студентов этой группы.

```
{% extends 'Authorization/base.html' %}
{% load crispy_forms_tags %}

{% block content %}

<div class="container w-75 boxshadow px-0 pb-3">
   <h1 class="text-center">Это страница группы {{ title }}</h1>

<hr>

   <div class="row">
      <div class="col ml-4">
         <h3>Список предметов группы {{ title }}</h3>
         {% for subject, subjectNamesSlugged in subjects %}
            <a href="{% url 'Journal:JournalPage' group_slug=title
subject=subjectNamesSlugged %}">{{ subject }}</a><br/>
         {% endfor %}
      </div>
      <div class="col">
         <h3>Здесь список студентов этой группы</h3>
         {% for user in userNames %}
            <a href="#">{{ user }}</a><br/>
         {% endfor %}
```

```
        </div>
    </div>

</div>


{% endblock %}
```

journal.html

Это страница журнала, самая важная часть проекта сразу после моделей и вьюшки этой страницы, на ней отображаются все студенты выбранной группы этого предмета, все оценки всех студентов выбранной группы выбранного предмета, может сохранять новые оценки для каждого студента на странице так же может изменять любую из показанных оценок, так же показывает даны для каждого ряда оценок.

```
{% extends 'Authorization/base.html' %}
{% load crispy_forms_tags %}

{% block content %}
    {% if user.is_authenticated %}
        <div class="container table-responsive boxshadow px-0">
            <div class="container">
                <h1 class="text-center mt-3"><a href="{% url 'Journal:GroupPage' group_slug=group %}">Группа: {{ group }}</a></h1>
                <h1 class="text-center mb-3">Предмет: {{ subject }}</h1>
            </div>
```

```
<table class="table table-bordered">
  <tr>
    <th class="w-0 text-center align-middle" rowspan="2">№</th>
    <th class="w-25 text-center align-middle h-100" rowspan="2">ФИО</th>
    <th class="w-75 text-center align-middle" colspan="100">
      Рабочая область
      <button class="btn btn-outline-dark float-right mr-5" type="submit" form="gradebook" name="submit">Сохранить</button>
    </th>
  </tr>
  <form id="gradebook" method="post" enctype="multipart/form-data">
  {% csrf_token %}
    <tr class="text-center align-middle">
      {% for date in userMarkDates %}
        <td><input class="form-control" id="date" type="date" name="date_{{ forloop.counter }}" value="{{ date }}"/></td>
      {% endfor %}
      <td><input class="form-control" id="date" type="date" name="date" value="{{ today }}"/></td>
    </tr>
  {% for id, name, marks in student %}
    <tr class="text-center align-middle">
      <td>{{ forloop.counter }}</td>
      <td class="text-left">{{ name }}</td>
      {% for mark in marks %}
        <td><input class="form-control" id="{{ id }}" type="text" name="User_{{ name }}_loopcounter_{{ forloop.counter }}_mark_{{ mark }}" value="{{ mark }}" autocomplete="off"/></td>
      {% endfor %}
```

```
                <td><input class="form-control" id="{{ id }}" type="text"
name="{{ name }}" autocomplete="off"/></td>
            </tr>
          {% endfor %}
          </form>
        </table>

      </div>
    {% else %}
      <div class="container table-responsive boxshadow px-0">
        <div class="container">
          <h1 class="text-center mt-3"><a href="{% url 'Journal:GroupPage'
group_slug=group %}">Группа: {{ group }}</a></h1>
          <h1 class="text-center mb-3">Предмет: {{ subject }}</h1>
        </div>

        <table class="table table-bordered">
          <tr>
            <th class="w-0 text-center align-middle" rowspan="2">№</th>
            <th class="w-25 text-center align-middle h-100"
rowspan="2">ФИО</th>
            <th class="w-75 text-center align-middle" colspan="100">
              Рабочая область
            </th>
          </tr>
          <form id="gradebook" method="post" enctype="multipart/form-data">
          {% csrf_token %}
            <tr class="text-center align-middle">
              {% for date in userMarkDates %}
```

```
                    <td><input class="form-control" id="date" type="date"
name="date_{{ forloop.counter }}" value="{{ date }}"/></td>
                {% endfor %}
                <td><input class="form-control" id="date" type="date"
name="date" value="{{ today }}"/></td>
            </tr>
          {% for id, name, marks in student %}
            <tr class="text-center align-middle">
              <td>{{ forloop.counter }}</td>
              <td class="text-left">{{ name }}</td>
              {% for mark in marks %}
                <td><input class="form-control" id="{{ id }}" type="text"
name="User_{{ name }}_loopcounter_{{ forloop.counter }}_mark_{{ mark }}"
value="{{ mark }}" autocomplete="off" disabled/></td>
              {% endfor %}
              <td><input class="form-control" id="{{ id }}" type="text"
name="{{ name }}" autocomplete="off" disabled/></td>
            </tr>
          {% endfor %}
          </form>
        </table>

    </div>
  {% endif %}

{% endblock %}
```

mainPage.html

Это главная страница на ней есть доступ ко всем группам сохранным в базу данных на данный момент, и она является первой страницей, которую человек заходящий на сайт видит.

```
{% extends 'Authorization/base.html' %}
{% load crispy_forms_tags %}

{% block content %}

<div class="container w-75 boxshadow px-0 pb-3">
   <h1 class="text-center">Это главная страница</h1>
   <hr>
   <div class="row">
   {% for grade, groups, groupsSlugifyed in mainPageInfo %}
      <div class="col">
         <h3><a class="ml-4" href="#">{{ grade }}</a><br></h3>
         {% for group in groupsSlugifyed %}
            <a class="ml-4 text-uppercase" href="{% url 'Journal:GroupPage' group_slug=group %}">{{ group }}</a><br>
         {% endfor %}
      </div>
   {% endfor %}

</div>


{% endblock %}
```

admin.py

Это тот же фейл что и для “Authorization” только регистрирует модели этого приложения.

```
from django.contrib import admin

from .models import *



@admin.register(JournalPageModel)
class JournalPageModelAdmin(admin.ModelAdmin):
    list_display = ('full_name_display', 'group_name_display', 'grade', 'subject')
    list_filter = ('group', 'grade')

    def full_name_display(self, obj):
        return '{} {}'.format(obj.name.first_name, obj.name.last_name)

    def group_name_display(self, obj):
        return '{}'.format(obj.group.name)



admin.site.register(GradeModel)
admin.site.register(SubjectsModel)
admin.site.register(StudentGroupModel)
```

forms.py

Это формы для приложения “journal” но т.к таблица для страницы журнала пришлось писать кастомную форму этот файл в этом приложении почти не был использован

```
from django import forms
from django.contrib import admin
from django.forms import ModelForm

from .models import *


### Admin panel forms

class JournalPageAdmin(admin.ModelAdmin):
    fields='__all__'
    list_display = ('first_name', 'last_name')
    form = JournalPageModel
```

methods.py

Это файл специально созданный для хранения функций которые часто использовались в ходе разработки, т.к эти функции часто использовались было решено записать их в отдельный файл и вызывать при необходимости, таким образом была решена проблема повторяющегося кода что уменьшило его количество, ускорило производительность сайта, уменьшило размер файла и ускорило процесс разработки.

```
import re
```

```
from datetime import date
from django.contrib.auth.models import User

from .models import *

def get_ids_marks_and_markdates(usernames, usersId, subjectID):
    today = date.today()

    today = today.strftime("%Y/%m/%d")
    today = re.sub('/', '-', today)

    userIdList = [None] * len(usernames)
    user_list = [None] * len(usernames)
    user_marks = [None] * len(usernames)
    user_markdates = [None] * len(usernames)



    ### Write all user ID's into a list
    for _ in range(len(usernames)):
        tempUserId = usersId.__getitem__(_)  # rewrite them to make a regular dict
        userIdList[_] = tempUserId.get('id')  # Write all id's into the list as regular
strings


    for _ in range(len(usernames)):
        ### Get first and last name, combine them and put into a list
        temp = User.objects.filter(id=userIdList[_]).values('first_name',
                                                            'last_name')  # Get first name and last
name
```

```
        temp = temp.__getitem__(0)  # Rewrite to make it regular dict


        firstName = temp.get('first_name')
        lastName = temp.get('last_name')
        fullName = firstName + ' ' + lastName  # Combine to make a single string with first and last names


        user_list[_] = fullName  # Write full name into list on a position


        ### Get marks by user ID and current subject
        tempUserMarks = JournalPageModel.objects.filter(name=userIdList[_], subject=subjectID).values('marks')  # Get all marks of a selected id
        temp = tempUserMarks[0]
        temp = temp.get('marks')


        if len(tempUserMarks) == 0:
            if len(temp) == 0:
                user_marks[_] = '-'
            elif re.search('\D', temp[0]):
                user_marks[_] = '-'
        else:
            tempUserMarks = tempUserMarks.__getitem__(0)  # Rewrite them to make a regular dict
            user_marks[_] = tempUserMarks.get('marks')  # Write user marks into the dict



    ### Get mark dates by user ID and current subject
    tempUserMarkDates = JournalPageModel.objects.filter(name=userIdList[0], subject=subjectID).values('marksDate') # Get all mark dates of a selected id
```

```
    temp = tempUserMarkDates[0]
    temp = temp.get('marksDate')

    if len(temp) == 0:
        user_markdates = []
    elif len(userIdList) == 1:
        tempUserMarkDates = tempUserMarkDates.__getitem__(0)  # Rewrite them to make a regular dict
        tempUserMarkDates = tempUserMarkDates.get('marksDate')  # Write user marks into the dict

        user_markdates = tempUserMarkDates

    else:
        tempUserMarkDates = tempUserMarkDates.__getitem__(0)  # Rewrite them to make a regular dict
        user_markdates[_] = tempUserMarkDates.get('marksDate')  # Write user marks into the dict

        if len(user_markdates) >= 2:
            user_markdates = user_markdates[_]

    return(userIdList, user_list, user_marks, user_markdates)


def compate_and_save_marks(usernames, userIdList, user_marks, receivedMarks, subjectID):
    for _ in range(len(usernames)):
        user = JournalPageModel.objects.get(name=userIdList[_], subject=subjectID)
        for j in range(len(user_marks[_])):
```

```
            if user_marks[_] == receivedMarks[_][j]:
                continue
            else:
                user.marks = receivedMarks[_]


                user.save()
                break



def compare_and_save_dates(usernames, userIdList, user_marks, user_markdates,
receivedMarkDates, subjectID):
    for _ in range(len(usernames)):
        user = JournalPageModel.objects.get(name=userIdList[_], subject=subjectID)
        for j in range(len(user_marks[_])):
            if len(user_markdates) == 1:
                if user_markdates[_] == receivedMarkDates[_]:
                    continue
                else:
                    user.marksDate = receivedMarkDates


                    user.save()
                    break
            else:
                if user_markdates[j] == receivedMarkDates[j]:
                    continue
                else:
                    user.marksDate = receivedMarkDates
```

```
user.save()
break
```

models.py

Данный файл содержит модели приложение “Journal”

```
import datetime

from django.db import models
from django.contrib.postgres.fields import ArrayField

from django.contrib.auth.models import User
from django.contrib.auth.models import Group


class GradeModel(models.Model):
    name = models.CharField(max_length=18)

    def __str__(self):
        return str(self.name)


class SubjectsModel(models.Model):
    name = models.CharField(max_length=50)

    def __str__(self):
        return str(self.name)


class StudentGroupModel(models.Model):
    name = models.CharField(max_length=20)
```

```
    grade = models.ForeignKey(GradeModel, on_delete=models.CASCADE)
    subjects = models.ManyToManyField(SubjectsModel)

    def __str__(self):
        name = self.name
        grade = self.grade.name
        nameAndGrade = name + ', ' + grade
        return nameAndGrade


class JournalPageModel(models.Model):
    name = models.ForeignKey(User, on_delete=models.CASCADE)
    grade = models.ForeignKey(GradeModel, blank=True,
on_delete=models.CASCADE)
    group = models.ForeignKey(StudentGroupModel, blank=True,
on_delete=models.CASCADE)
    subject = models.ForeignKey(SubjectsModel, blank=True,
on_delete=models.CASCADE)
    marks = ArrayField(models.CharField(max_length=2, blank=True),
blank=True)
    marksDate = ArrayField(models.CharField(max_length=20, blank=True),
blank=True)

    def __str__(self):
        firstName = self.name.first_name
        lastName = self.name.last_name
        grade = self.grade.name
        group = self.group.name
        subject = self.subject.name
        return str(firstName + ' ' + lastName + ', ' + grade + ', ' + group + ', ' + subject)
```

```
        # return str(firstName + ' ' + lastName)
```

urls.py

Данный файл содержит “урлы” (URL) для приложения “Journal”

```
from django.contrib import admin
from django.urls import include, path, re_path

from django.conf import settings
from django.conf.urls.static import static

from .views import *

app_name = 'Journal'
urlpatterns = [
    path('', MainPageView.as_view(), name='MainPage'),
    re_path(r'^group/(?P<group_slug>[\w-]+)/$', GroupPageView.as_view(), name='GroupPage'),
    re_path(r'^group/(?P<group_slug>[\w-]+)/(?P<subject>[\w-]+)/$', JournalPageView.as_view(), name='JournalPage'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

views.py

Данный файл содержит “вьюшки” для приложения “Journal” и отвечает за логику сохранения оценок в базу данных, изменения имеющихся оценок, сохранения дат оценок в базу данных и их изменение, так же отвечает за отправку информации на главную страницу, за отправку информации на страницу выбранной группы (фильтрует все группы чтобы показать именно ту что была выбрана.)

```
import re

from datetime import date

from django import template
from django.shortcuts import render, redirect
from django.views.generic import TemplateView
from django.utils.text import slugify
from django.db.models import Q, Value
from django.urls import reverse

from django.contrib.auth.models import User

from Authorization.models import *
from .models import *
from .forms import *
from .methods import (
    get_ids_marks_and_markdates,
    compate_and_save_marks,
    compare_and_save_dates,
```

```
)


class MainPageView(TemplateView):
    name = 'Journal/mainPage.html'

    def get(self, request):

        gradesIdsList = GradeModel.objects.all().values_list('id', flat=True)
        gradesRaw = GradeModel.objects.filter().values('name')

        groupsRaw =
StudentGroupModel.objects.all().order_by('grade').values_list('name',
                                                              'grade')  # By the order of
'grade' take fields 'name' and 'grade' and return them as a tuple in a list

        grades = [None] * len(gradesRaw)
        groups = [[] for _ in range(len(gradesIdsList))]
        groupsSlugifyed = [[] for _ in range(len(gradesIdsList))]

        for _ in range(len(grades)):
            tempGrades = gradesRaw.__getitem__(_)  # rewrite them to make a regular
dict
            grades[_] = tempGrades.get('name')  # Write all grade names into the list as
regular strings

        for _ in range(len(gradesIdsList)):
            for j in range(len(groupsRaw)):
                if groupsRaw[j][1] == (_ + 1):
```

```
                groups[_].append(groupsRaw[j][0])
                groupsSlugifyed[_].append(slugify(groupsRaw[j][0],
allow_unicode=True))

        context = {
            'title': 'Главная страница',
            'mainPageInfo': zip(
                grades,
                groups,
                groupsSlugifyed,
            ),
        }
        return render(request, self.name, context)



class GroupPageView(TemplateView):
    name = 'Journal/groupPage.html'


    def get(self, request, group_slug):
        chosenGroup =
StudentGroupModel.objects.filter(name__icontains=group_slug).values('subjects')
        groupID =
StudentGroupModel.objects.filter(name__icontains=group_slug).values_list('pk')
        groupStudentsIDs = Student.objects.filter(group=groupID[0][0]).values('user')


        subjectNames = [None] * len(chosenGroup)
        subjectNamesSlugged = [None] * len(chosenGroup)


        groupUserNames = [None] * len(groupStudentsIDs)
```

```
        for _ in range(len(groupStudentsIDs)):
            temp =
User.objects.filter(pk=groupStudentsIDs[_]['user']).values_list('first_name',
'last_name')
            temp = temp[0][0] + ' ' + temp[0][1]
            groupUserNames[_] = temp


        for _ in range(len(chosenGroup)):
            temp = chosenGroup.__getitem__(_)
            temp = temp.get('subjects')


            temp = SubjectsModel.objects.filter(pk=temp).values('name')
            temp = temp.__getitem__(0)
            temp = temp.get('name')


            subjectNames[_] = temp
            subjectNamesSlugged[_] = slugify(temp, allow_unicode=True)


        context = {
            'title': group_slug.upper(),
            'subjects': zip(
                subjectNames,
                subjectNamesSlugged,
            ),
            'userNames': groupUserNames,
        }


        return render(request, self.name, context)
```

```
class JournalPageView(TemplateView):
    name = 'Journal/journal.html'
    DATE_FORMAT = "%d-%m-%Y"

    def get(self, request, group_slug, subject):
        ### Variables

        group = group_slug.upper()
        groupID =
StudentGroupModel.objects.filter(name__icontains=group).values('pk')
        groupID = groupID[0]['pk']

        currentSubject = subject.capitalize()
        currentSubject = re.sub("-", " ", currentSubject)

        currentSubjectID =
SubjectsModel.objects.filter(Q(name__icontains=currentSubject)).values('id')
        currentSubjectID = currentSubjectID[0]['id']

        usersWithGroupAndSubjectRaw =
JournalPageModel.objects.filter(Q(group=groupID,
subject=currentSubjectID)).values('name')
        usersWithGroupAndSubject = [None] * len(usersWithGroupAndSubjectRaw)

        for _ in range(len(usersWithGroupAndSubjectRaw)):
            usersWithGroupAndSubject[_] =
usersWithGroupAndSubjectRaw[_]['name']
```

```
        usernames =
User.objects.filter(pk__in=usersWithGroupAndSubject).values('username')
        usersId = User.objects.filter(pk__in=usersWithGroupAndSubject).values('id')
# Get all user id


        print('\n',
        'Group ID', groupID, '\n',
        'Current subject ID', currentSubjectID, '\n',
        'Current group with subject', usersWithGroupAndSubject, '\n',
        'Usernames', usernames, '\n',
        'User ID', usersId, '\n',
        )

        ### Get today and parse it into a string with dd/mm/YY
        today = date.today()

        today = today.strftime("%Y/%m/%d")
        today = re.sub('/', '-', today)

        ### Call function to get user id list mark lists and mark date list
        func = get_ids_marks_and_markdates(usernames, usersId, currentSubjectID)

        ### Put function return into a variables
        userIdList = func[0]
        user_list = func[1]
        user_marks = func[2]
        user_markdates = func[3]

        print('\n', 'User id - ', userIdList, '\n',
```

```
        'User list - ', user_list, '\n',
        'User marks - ', user_marks, '\n',
        'User markdates - ', user_markdates,  '\n',)

        context = {
            'userMarkDates': user_markdates,
            'today': today,
            'student': zip(
                userIdList,
                user_list,
                user_marks,
            ),
            'group': group,
            'subject': currentSubject,
            'title': 'Страница журнала',
        }
        return render(request, self.name, context)

    def post(self, request, group_slug, subject):
        ### Variables
        group = group_slug.upper()
        groupID =
StudentGroupModel.objects.filter(name__icontains=group).values('pk')
        groupID = groupID[0]['pk']

        currentSubject = subject.capitalize()
        currentSubject = re.sub("-", " ", currentSubject)
```

```
        currentSubjectID =
SubjectsModel.objects.filter(Q(name__icontains=currentSubject)).values('id')
        currentSubjectID = currentSubjectID[0]['id']


        usersWithGroupAndSubjectRaw =
JournalPageModel.objects.filter(Q(group=groupID,
subject=currentSubjectID)).values('name')
        usersWithGroupAndSubject = [None] * len(usersWithGroupAndSubjectRaw)


        for _ in range(len(usersWithGroupAndSubjectRaw)):
            usersWithGroupAndSubject[_] =
usersWithGroupAndSubjectRaw[_]['name']


        usernames =
User.objects.filter(pk__in=usersWithGroupAndSubject).values('username')
        usersId = User.objects.filter(pk__in=usersWithGroupAndSubject).values('id')
# Get all user id


        newMarks = [None] * len(usernames)


        ### Call function to get user id list mark lists and mark date list
        func = get_ids_marks_and_markdates(usernames, usersId, currentSubjectID)


        ### Put function return into a variables
        userIdList = func[0]
        user_list = func[1]
        user_marks = func[2]
        user_markdates = func[3]


        i = 0
```

```
        j = 0

        ### Initialize oldMarks 2d array
        receivedMarks = []
        for i in range(len(usernames)):
            column = []
            for j in range(len(user_marks[i])):
                column.append(None)
            receivedMarks.append(column)

        receivedMarkDates = [None] * len(user_markdates)

        print('\n', 'Post request data', request.POST, '\n',
              'User id list', userIdList, '\n',
              'Existing marks', user_marks, '\n',
              'Existing mark dates', user_markdates, '\n')



        ### Get data from HTML form

        for _ in range(len(usernames)):
            newMarks[_] = request.POST.get(user_list[_])

            for j in range(len(user_marks[_])):
                receivedMarks[_][j] = request.POST.get(
                    'User_' + user_list[_] + '_loopcounter_' + str(j + 1) + '_mark_' +
str(user_marks[_][j]))
```

```
        for _ in range(len(user_markdates)):
            receivedMarkDates[_] = request.POST.get('date_' + str(_ + 1))


        newDate = request.POST.get('date')


        print('\n', 'New marks', newMarks, '\n',
              'Received marks', receivedMarks, '\n',
              'Received mark dates', receivedMarkDates, '\n',
              'New date', newDate, '\n')


        ### Check if new and old marks and dates are identical


        compate_and_save_marks(usernames, userIdList, user_marks,
receivedMarks, currentSubjectID)
        compare_and_save_dates(usernames, userIdList, user_marks,
user_markdates, receivedMarkDates, currentSubjectID)


        ### Save data from HTML form to database
        for _ in range(len(usernames)):
            user = JournalPageModel.objects.get(name=userIdList[_],
subject=currentSubjectID)
            print(user)
            if len(user.marks) == 1 and user.marks == [
                '-']:  ### If there is only one entry and that entry is default "-" then...
                ### Replace the default entry with the mark
                user.marks.pop()
                user.marks.append(newMarks[_])


                ### Replace the entry with the date
```

```
            user.marksDate.pop()
            user.marksDate.append(newDate)


            user.save()
        elif newMarks[_] == '':
            pass
        else:
            ### Append the mark
            user.marks.append(newMarks[_])


            ### Append the date
            user.marksDate.append(newDate)


            user.save()

    return redirect(reverse('Journal:JournalPage', kwargs={'group_slug':
group_slug, 'subject': subject}))
```