Друзяка

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc137721059)

[1 ИНФОРМАЦИОННЫЕ СИСТЕМЫ 5](#_Toc137721060)

[1.1 Описание информационных систем 5](#_Toc137721061)

[1.2 ИС «Рабочее пространство» 9](#_Toc137721062)

[1.3 Предприятие и информационные системы 10](#_Toc137721063)

[2 СРЕДА РАЗРАБОТКИ 12](#_Toc137721064)

[2.1 Программные средства для разработки информационных систем на примере сайта 12](#_Toc137721065)

[2.2 Среда разработки python django 14](#_Toc137721066)

[3 РАЗРАБОТКА САЙТА ДЛЯ ПРЕДПРИЯТИЯ 17](#_Toc137721067)

[3.1 Подготовка к работе и планирование сайта 17](#_Toc137721068)

[3.2 Начало создания сайта 17](#_Toc137721069)

[3.3 Настройка основных файлов 19](#_Toc137721070)

[3.4 Создание HTML шаблона 21](#_Toc137721071)

[3.5 Настройка моделей 26](#_Toc137721072)

[3.6 Создание тегов 30](#_Toc137721073)

[3.7 Создание основных страниц 32](#_Toc137721074)

[ЗАКЛЮЧЕНИЕ 50](#_Toc137721075)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 51](#_Toc137721076)

# ВВЕДЕНИЕ

Сайты играют огромную роль в современной жизни. Они позволяют осуществлять огромное разнообразие деятельности в сети «Интернет». Сайты позволяют легче взаимодействовать и организовать деятельность людей в самых различных сферах: как между самими предприятиями, так и между их клиентами. Для различных социальных групп они обеспечивают возможность удаленного общения, обмен и распространение информации, обеспечение доступа к ней большому кругу людей и т.д.

Распространение различных услуг и информации с помощью сайтов является одним из наиболее оптимальных в современных реалиях, так как это позволяет пользователям получать к ним доступ без предварительного скачивания, используя одно программное обеспечение (браузер) для доступа к огромному количеству различных информационных систем. Эти системы нередко также могут взаимодействовать друг с другом и взаимно интегрироваться.

Ввиду вышеперечисленных преимуществ разрабатываемую информационную систему было решено реализовать в виде сайта. Данная информационная система должна решать проблему организации деятельности и информации пользователя. Данная проблема является актуальной в современном мире благодаря постоянно возрастающему количеству потребляемой и обрабатываемой людьми информацией при ведении деятельности и повседневной жизни.

Целью дипломной работы является разработка информационной системы «Рабочее пространство». Данный ресурс необходим для создания записей, заметок, списков и добавления картинок с целью организации деятельности.

Задачи дипломной работы:

Рассмотреть типы информационных систем

* изучить предметную область, для которой разрабатывается ресурс,
* выбрать средства и технологии для разработки,
* спроектировать интерфейс информационной системы в виде веб-сайта и реализовать модель.

Дипломная работа состоит из введения, трех глав, подразделенных на параграфы, заключения, списка использованных источников и приложений.

В первой главе рассматривается общее понятие информационных систем, и в частности сайтов, функции ИС рабочих пространств а производится анализ программных средств для разработки информационной системы.

Вторая глава содержит описание среды разработки python django.

Третья глава содержит подробное описание процесса разработки информационной системы.

Источник информации для решения задач ВКР включают в себя опубликованные работы других авторов, на которые в тексте данной работы имеются ссылки.

# 1 ИНФОРМАЦИОННЫЕ СИСТЕМЫ

## 1.1 Описание информационных систем

Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию

ИС предназначена для своевременного обеспечения надлежащих людей надлежащей информацией, то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, базы данных и информационные услуги.

Понятие информационной системы интерпретируют по-разному, в зависимости от контекста.

Достаточно широкое понимание информационной системы подразумевает, что её неотъемлемыми компонентами являются данные, техническое и программное обеспечение, а также персонал и организационное обеспечение. Широко трактует понятие «информационной системы» федеральный закон Российской Федерации «Об информации, информационных технологиях и о защите информации», подразумевая под информационной системой совокупность содержащейся в базах данных информации и обеспечивающих её обработку информационных технологий и технических средств.

Среди российских ученых в области информатики наиболее широкое определение ИС дает М. Р. Когаловский, по мнению которого в понятие информационной системы помимо данных, программ, аппаратного обеспечения и людских ресурсов следует также включать коммуникационное оборудование, лингвистические средства и информационные ресурсы, которые в совокупности образуют систему, обеспечивающую «поддержку динамической информационной модели некоторой части реального мира для удовлетворения информационных потребностей пользователей».

Более узкое понимание информационной системы ограничивает её состав данными, программами и аппаратным обеспечением. Интеграция этих компонентов позволяет автоматизировать процессы управления информацией и целенаправленной деятельности конечных пользователей, направленной на получение, модификацию и хранение информации. Так, российский стандарт ГОСТ РВ 51987 подразумевает под ИС «автоматизированную систему, результатом функционирования которой является представление выходной информации для последующего использования». ГОСТ Р 53622-2009 использует термин информационно-вычислительная система для обозначения совокупности данных (или баз данных), систем управления базами данных и прикладных программ, функционирующих на вычислительных средствах как единое целое для решения определённых задач.

ИС в деятельности организации рассматривается как программное обеспечение, реализующее её деловую стратегию и бизнес-процессы. Желательной целью является создание и развертывание единой корпоративной информационной системы, удовлетворяющей информационные потребности всех сотрудников, служб и подразделений организации. На практике создание такой системы слишком затруднено или даже невозможно, а иногда и нецелесообразно, вследствие чего на предприятии обычно функционируют несколько различных систем, решающих отдельные группы задач: управление производством, финансово-хозяйственная деятельность, электронный документооборот и т. д. Часть задач бывает «покрыта» одновременно несколькими информационными системами, часть задач — вовсе не автоматизирована. Такая ситуация получила название «лоскутной автоматизации» и является довольно типичной для многих предприятий.

Классификация по архитектуре

По степени распределённости отличают:

настольные, или локальные ИС, в которых все компоненты (БД, СУБД, клиентские приложения) находятся на одном компьютере;

распределённые ИС, в которых компоненты распределены по нескольким компьютерам.

Распределённые ИС, в свою очередь, разделяют на:

файл-серверные ИС (ИС с архитектурой «файл-сервер»);

клиент-серверные ИС (ИС с архитектурой «клиент-сервер»).

В файл-серверных ИС база данных находится на файловом сервере, а СУБД и клиентские приложения находятся на рабочих станциях.

В клиент-серверных ИС база данных и СУБД находятся на сервере, а на рабочих станциях находятся только клиентские приложения.

В свою очередь, клиент-серверные ИС разделяют на двухзвенные и многозвенные.

В двухзвенных ИС всего два типа «звеньев»: сервер базы данных, на котором находятся БД и СУБД, и рабочие станции, на которых находятся клиентские приложения. Клиентские приложения обращаются к СУБД напрямую.

В многозвенных ИС добавляются промежуточные «звенья»: серверы приложений (application servers). Пользовательские клиентские приложения не обращаются к СУБД напрямую, они взаимодействуют с промежуточными звеньями. Типичный пример применения трёхзвенной архитектуры — современные веб-приложения, использующие базы данных. В таких приложениях помимо звена СУБД и клиентского звена, выполняющегося в веб-браузере, имеется как минимум одно промежуточное звено — веб-сервер с соответствующим серверным программным обеспечением.

Классификация по степени автоматизации

По степени автоматизации ИС делятся на:

автоматизированные: информационные системы, в которых автоматизация может быть неполной (то есть требуется постоянное вмешательство персонала);

автоматические: информационные системы, в которых автоматизация является полной, то есть вмешательство персонала не требуется или требуется только эпизодически.

«Ручные ИС» («без компьютера») существовать не могут, поскольку существующие определения предписывают обязательное наличие в составе ИС аппаратно-программных средств. Вследствие этого понятия «автоматизированная информационная система», «компьютерная информационная система» и просто «информационная система» являются синонимами.

Классификация по характеру обработки данных

По характеру обработки данных ИС делятся на:

информационно-справочные, или информационно-поисковые ИС, в которых нет сложных алгоритмов обработки данных, а целью системы является поиск и выдача информации в удобном виде;

ИС обработки данных, или решающие ИС, в которых данные подвергаются обработке по сложным алгоритмам. К таким системам в первую очередь относят автоматизированные системы управления и системы поддержки принятия решений.

Классификация по сфере применения

Поскольку ИС создаются для удовлетворения информационных потребностей в рамках конкретной предметной области, то каждой предметной области (сфере применения) соответствует свой тип ИС. Перечислять все эти типы не имеет смысла, так как количество предметных областей велико, но можно указать в качестве примера следующие типы ИС:

Экономическая информационная система — информационная система, предназначенная для выполнения функций управления на предприятии.

Медицинская информационная система — информационная система, предназначенная для использования в лечебном или лечебно-профилактическом учреждении.

Географическая информационная система — информационная система, обеспечивающая сбор, хранение, обработку, доступ, отображение и распространение пространственно-координированных данных (пространственных данных).

Классификация по охвату задач (масштабности)

Персональная ИС предназначена для решения некоторого круга задач одного человека.

Групповая ИС ориентирована на коллективное использование информации членами рабочей группы или подразделения.

Корпоративная ИС автоматизирует все бизнес-процессы целого предприятия (организации) или их значительную часть, достигая их полной информационной согласованности, безызбыточности и прозрачности. Такие системы иногда называют информационными системами предприятия и системами комплексной автоматизации предприятия.

## 1.2 ИС «Рабочее пространство»

Рабочее пространство – это информационная система предоставляющая функционал для организации различных видов деятельности, такой как ведение заметок, управление знаниями и данными, управление проектами и задачами.

Примеры данных приложений:

Notion — приложение, которое предоставляет такие компоненты, как базы данных, доски, вики, календари и напоминания. Пользователи могут подключать эти компоненты для создания собственных систем управления знаниями, ведения заметок, управления данными, управления проектами. Эти компоненты и системы могут использоваться индивидуально или совместно с другими.

Evernote — веб-сервис и набор программного обеспечения для создания и хранения заметок. В качестве заметки может выступать фрагмент форматированного текста, веб-страница целиком, фотография, аудиофайл или рукописная запись. Заметки могут также содержать вложения с файлами другого типа. Заметки можно сортировать по блокнотам, присваивать им метки, редактировать и экспортировать.

Google Workspace — набор облачных сервисов, предоставляемых компанией Google. Google Workspace включает в себя Gmail, Календарь и Google Meet для связи; Google Drive для хранения; Документы, таблицы, слайды, формы и сайты для совместной работы; и панель администратора и хранилище для управления пользователями и службами.

ClickUp — сервис, включающий в себя следующие функции: составления списка целей и задач, чат, создание документов, уведомлений, составление расписания.

## 1.3 Предприятие и информационные системы

Сегодня предприятия используют в своей работе огромное количество информационных систем различных видов. Практически все даже самые небольшие предприятия используют в своей работе информационные системы для работы с базами данных для учета деятельности и облегчения работы, хранения данных клиентов. Используют различные сервисы позволяющие обеспечить коммуникацию между сотрудниками и другими предприятиями и многое другое.

Разрабатываемый в ходе данной дипломной работы сайт так же может использоваться не только пользователями в личных целях но и ИП и предприятиями.

Пример использования можно привести на базе производственной практики компьютерного центра «Икс-Про». Данный компьютерный центр использует в своей деятельности такие информационные системы как 1С и «Битрикс24» для организации и учета деятельности. Сайт может использоваться сотрудниками предприятия для создания личных рабочих заметок и напоминаний, особенно учитывая, что при прохождении практики были замечены некоторые проблемы с личной организованностью сотрудников. Так же данный сайт имеет большой потенциал усовершенствования и при дополнении функционала эффективность и удобство его использования в подобных целях может значительно возрасти.

# 2 СРЕДА РАЗРАБОТКИ

## 2.1 Программные средства для разработки информационных систем на примере сайта

Одной из наиболее актуальных и востребованных программных средств для разработки информационных систем сайтов являются фреймворки.

Фреймворки — это программные продукты, которые упрощают создание и поддержку технически сложных или нагруженных проектов. Фреймворк, как правило, содержит только базовые программные модули, а все специфичные для проекта компоненты реализуются разработчиком на их основе. Тем самым достигается не только высокая скорость разработки, но и большая производительность и надёжность решений.

Веб-фреймворк — это платформа для создания сайтов и веб-приложений, облегчающая разработку и объединение разных компонентов большого программного проекта. За счёт широких возможностей в реализациии бизнес-логики и высокой производительности эта платформа особенно хорошо подходит для создания сложных сайтов, бизнес-приложений и веб-сервисов.

Основные преимущества фреймворков

С точки зрения бизнеса разработка на фреймворке почти всегда экономически эффективнее и качественнее по результату, нежели написание проекта на чистом языке программирования без использования каких-либо платформ. Разработка без использования платформы может быть правильным решением только в двух случаях — либо проект совсем простой и не требующий дальнейшего развития, либо очень нагруженный и требует очень низкоуровневой оптимизации (например, веб-сервисы с десятками тысяч обращений в секунду). Во всех других случаях разработка на программной платформе быстрее и качественнее.

Одним из главных преимуществ в использовании фреймворков является то, что фреймворк определяет унифицированную структуру для построенных на его базе приложений. Поэтому приложения на фреймворках значительно проще сопровождать и дорабатывать, так как стандартизированная структура организации компонентов понятна всем разработчикам на этой платформе и не требуется долго разбираться в архитектуре, чтобы понять принцип работы приложения или найти место реализации того или иного функционала. Большинство фреймворков для разработки веб-приложений использует парадигму MVC (модель-представление-контроллер) — то есть очень во многих фреймворках идентичный подход к организации компонентов приложения и это ещё больше упрощает понимание архитектуры приложения даже на незнакомом разработчику фреймворке.

Проектирование архитектуры ПО при разработке на фреймворке тоже очень упрощается — в методологиях фреймворков обычно заложены лучшие практики программной инженерии и просто следуя этим правилам можно избежать многих проблем и ошибок в проектировании. По сути, фреймворк — это множество конкретных и абстрактных классов, связанных между собой и упорядоченных согласно методологии фреймворка. Конкретные классы обычно реализуют взаимные отношения между классами, а абстрактные классы представляют собой точки расширения, в которых заложенный во фреймворк базовый функционал может быть использован «как есть» или адаптирован под задачи конкретного приложения. Для обеспечения расширения возможностей в большинстве фреймворков используются техники объектно-ориентированного программирования: например, части приложения могут наследоваться от базовых классов фреймворка или отдельные модули могут быть подключены как примеси.

Экосистемы веб-фреймворков также богаты на готовые реализации многих функциональных возможностей. Разработчики могут воспользоваться уже созданной сообществом реализацией. А это не только сокращает затраты времени и денег, но и позволяет добиться более высокой стабильности решения — компонент, который используется и дорабатывается тысячами других разработчиков обычно более качественно реализован и лучше протестирован на всевозможных сценариях, нежели решение, которое может в адекватные сроки разработать один разработчик или даже небольшая команда.

Популярные php-фреймворки:

Yii

Symphony

Zend

Laravel

Kohana

CodeIgniter

Наиболее популярные Ruby-фреймворки:

Ruby on Rails

Sinatra

Популярные Java-фреймворки:

Spring MVC

JSF

GWT

## 2.2 Среда разработки python django

Django - свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC[6]. 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 сайтах Instagram, Disqus, Mozilla, The Washington Times, Pinterest, YouTube, Google и др.

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).

Некоторые возможности Django:

1. ORM, API доступа к БД с поддержкой транзакций;
2. встроенный интерфейс администратора, с уже имеющимися переводами на многие языки
3. диспетчер URL на основе регулярных выражений;
4. расширяемая система шаблонов с тегами и наследованием;
5. система кеширования;
6. интернационализация;
7. подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты;
8. «generic views» — шаблоны функций контроллеров;
9. авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и т.д.
10. система фильтров («middleware») для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий;
11. библиотека для работы с формами (наследование, построение форм по существующей модели БД)
12. встроенная автоматическая документация по тегам шаблонов и моделям данных, доступная через административное приложение
13. 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.

# 3 РАЗРАБОТКА САЙТА ДЛЯ ПРЕДПРИЯТИЯ

## 3.1 Подготовка к работе и планирование сайта

Первым этапом разработки информационной системы является планирование.

Данная информационная система должна предоставлять возможность регистрации и авторизации с созданием страницы пользователя, на которой пользователь сможет размещать содержимое: текст, картинки.

Структура базы данных для данной информационной системы выглядит следующим образом:

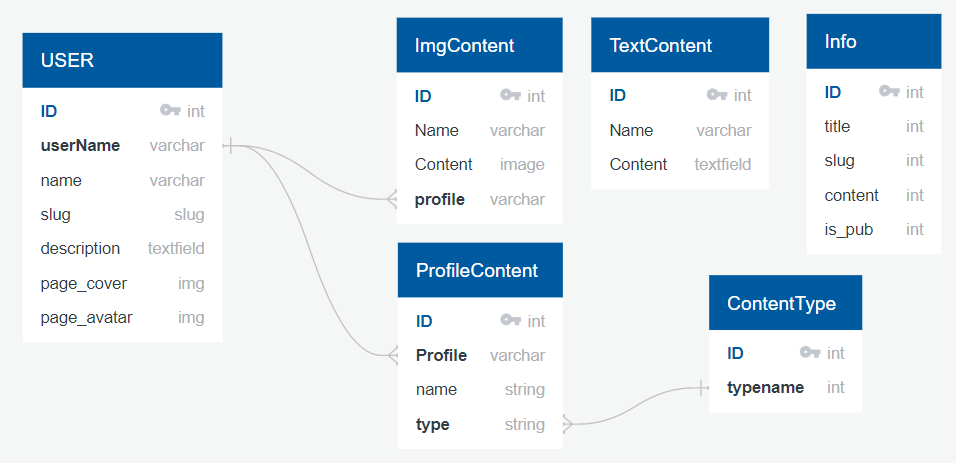


Рисунок 1. Структура БД ИС[[1]](#footnote-1)

Перед началом разработки программного кода необходимо установить необходимые среды инструменты и библиотеки

Производится установка Django, модулей, создание проекта Django.

## 3.2 Начало создания сайта

Проект Django создается с помощью команды командной строки из директории в которой будет храниться проект:

django-admin startproject НазваниеПроект

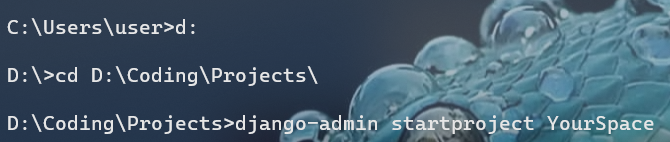


Рисунок 2. Создание проекта[[2]](#footnote-2)

Следующим этапом является создание приложения Django.

Приложение – это независимая часть проекта которая может быть отключена или подключена к любому проекту Django, имеет собстенные URL, модели(таблицы базы данных) шаблоны, классы и функции представления и другие элементы проекта Django.

Приложение создается следующей командой из директории проекта:

manage.py startapp НазваниеПриложения

manage.py – это файл, через который происходит управление проектом, команды этого файла запускается из главной директории проекта в которой находится данный файл.

Приложение данного проекта называется home и команда будет выглядеть следующим образом:

manage.py startapp home

Так же после создания проекта появляется возможность запустить локальный отладочный сервер, он запускается следующей командой:

manage.py runserver.

При успешном запуске в командную строку выводится сообщение об этом и локальный URL по которому доступен сайт. В данном случае это <http://127.0.0.1:8000/>. При переходе по данной ссылке выведется стандартная страница django об успешном запуске сервера.



Рисунок 3. Локальный Адрес сайта[[3]](#footnote-3)

## 3.3 Настройка основных файлов

settings.py – это файл с основными настройками проекта.

Одна из настроек по умолчанию, DEBUG = True, означает режим отладки проекта и именно в таком режиме должен находится проект при разработке, при данном режиме сообщения об ошибках в браузере будут более подробными и отображать необходимую информацию для её исправления, но при внедрении информационной системы в эксплуатацию значение данной настройки меняется на False.

После создания приложения home его необходимо зарегистрировать в settings.py:

INSTALLED\_APPS = [

Другие приложения

'home.apps.HomeConfig',

]

Создание первой функции представления

Импорт необходимыех для работы функций:

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.http import HttpResponse, HttpResponseNotFound, Http404

from django.views.generic import DetailView, CreateView

from django.contrib.auth.views import LoginView

from django.urls import reverse\_lazy

from django.contrib.auth import login

from django.contrib.auth import logout as django\_logout

Так же импортируем файлы models и forms в которых соответственно будут прописаны модели(таблицы базы данных) и формы.

from .models import \*

from .forms import \*

Создание функции представления главной страницы:

Сначала объявляется функция, её название и в скобочках прописывается атрибут request. Далее прописывается context, это список передаваемых в шаблон значений, через него будет передаваться название страницы в шаблон.

def start(request):

context = {

'title': "Let's start!",

}

return render(request, 'home/start.html', context = context)

Функция завершается строкой return render(request, 'home/start.html', context = context). Которая означает что данная функция при вызове возвращает веб-страницу шаблоном которой является файл start.html, контекст страницы – список context.

Что бы эта функция сработала её нужно вызвать, функции и классы представления вызываются в файле urls.py. как видно из названия, в данном файле прописываются URL пути приложения.

URL данной функции будет выглядеть следующим образом:

urlpatterns = [

…

path('start/', start, name = 'startpage'),

…

]

'start/' – адрес, по которой будет вызываться функция, start – сама функция, name = 'startpage' – имя URL адреса. Имя прописывается для того что бы по нему можно было вызвать данный URL даже если сам URL изменится.

## 3.4 Создание HTML шаблона

Шаблон – это HTML файл отображающий какую то часть HTML страницы.

Часто на сайте есть повторяющиеся на нескольких страницах элементы. Чтобы не прописывать их на каждой странице, используются шаблоны. Обычно шаблон какой либо функции или класса, отображаемой на странице расширяет базовый шаблон на котором отображаются повторяющиеся элементы.

HTML страница составляется из нескольких таких шаблонов.

Сначала создается базовый шаблон:

Импор необходимых путей модулей и файлов:

{% load static %}

{% url 'home' as homelink %}

{% load i18n %}

{% load home\_tags %}

{% getInfo as Info %}

Далее пишется HTML код базового шаблона

Использование шаблонов позволяет вставлять в HTML код различные переменные, прописанные в других файлах. В данном случае вместо {{ title }} в шаблон будет подставлять название страницы, прописанное в файле view:

<title>YourSpace – {{ title }}</title>

Стили берутся из директории static, которая импортирована выше, её путь можно записать следующим образом:

<link rel="stylesheet" type="text/css" href="{% static 'home/css/mainst.css' %}">

<link type="image/x-icon" href="{% static 'home/img/gal.ico' %}" rel="shortcut icon">

<link type="image/x-icon" href="{% static 'home/img/gal.ico' %}" rel="icon">

Так же прописываем блок для подключения ссылок из других шаблонов:

{% block headlink %}

{% endblock %}

В основном в шаблоне будет прописан хедер страницы, где будут находиться основные разделы сайта:

<header>

<div class="head">

<a class="mlink" href="{{ homelink }}">

<img class="logo" src="{% static 'home/img/1534058.png' %}">

<h1>YourSpace</h1>

</a>

<ul class="headlist">

<li class="hli hl"><a href="{% url 'guides' %}">{% trans 'Guides & tutorials' %}</a></li>

<li class="hli hl"><a href="{{Info.0.get\_absolute\_url}}">{% trans 'Info' %}</a></li>

В хедере сайта так же будет находиться форма переключения языка. Язык переключается с помощью модуля i18n, который был импортирован выше. Фразы, которые переводятся, заключены в скобки:

{% trans 'Переводимое слово' %}.

<li class="hli lang">

<form name="lang" action="{% url 'set\_language' %}" method="post">{% csrf\_token %}

<input name="next" type="hidden" value="{{ redirect\_to }}">

<select name="language">

{% get\_current\_language as LANGUAGE\_CODE %}

{% get\_available\_languages as LANGUAGES %}

{% get\_language\_info\_list for LANGUAGES as languages %}

{% for language in languages %}

<option value="{{ language.code }}"{% if language.code == LANGUAGE\_CODE %} selected{% endif %}>

{{ language.code }} {{ language.name\_local }}

</option>

{% endfor %}

</select>

</form>

</li>

<li class="hli"><button class="thbut" type="button">

<img class="thimg " src="{% static 'home/img/dtheme.png' %}">

<img class="thimg2 nc" src="{% static 'home/img/ltheme.png' %}">

</button></li>

{% block headbutt %}

{% endblock %}

{% if request.user.is\_authenticated %}

<div class="usercont">

<div class="">

<img src="{{ user.profile\_pic.url }}">

</div>

<p>{{user.username}}</p>

</div>

{% else %}

<li class="hli"><a class="login" href="{% url 'signin' %}">{% trans 'Sign In' %}</a></li>

{% endif %}

</ul>

</div>

</header>

<div class="profwind profwindnvis">

<div class="usercont">

<div class="">

<img src="{{ user.profile\_pic.url }}">

</div>

<a href="{% url 'userredirect' %}">{{user.username}}</a>

</div>

<a href="{% url 'logout' %}">Logout</a>

</div>

С помощью блоков шаблона можно вставлять в базовый шаблон содержимое других шаблонов:

{% block content %}

{% endblock %}

Далее подключаем скрипты, создаем блок для подключения дополнительных скриптов из других шаблонов:

<script src="{% static 'home/js/script1.js' %}"></script>

<script src="{% static 'home/js/userwin.js' %}"></script>

{% block script %}

{% endblock %}

В шаблоне start.html прописываем содержимое которое будет расширять базовый шаблон:

{% block content %}

<main>

<h3>{% trans 'Your personal work' %}<span class="stext">{%trans 'space'%}</span>!</h3>

</main>

{% endblock %}

Далее прописываем основные стили в файле CSS.

В результате получается главная страница сайта:

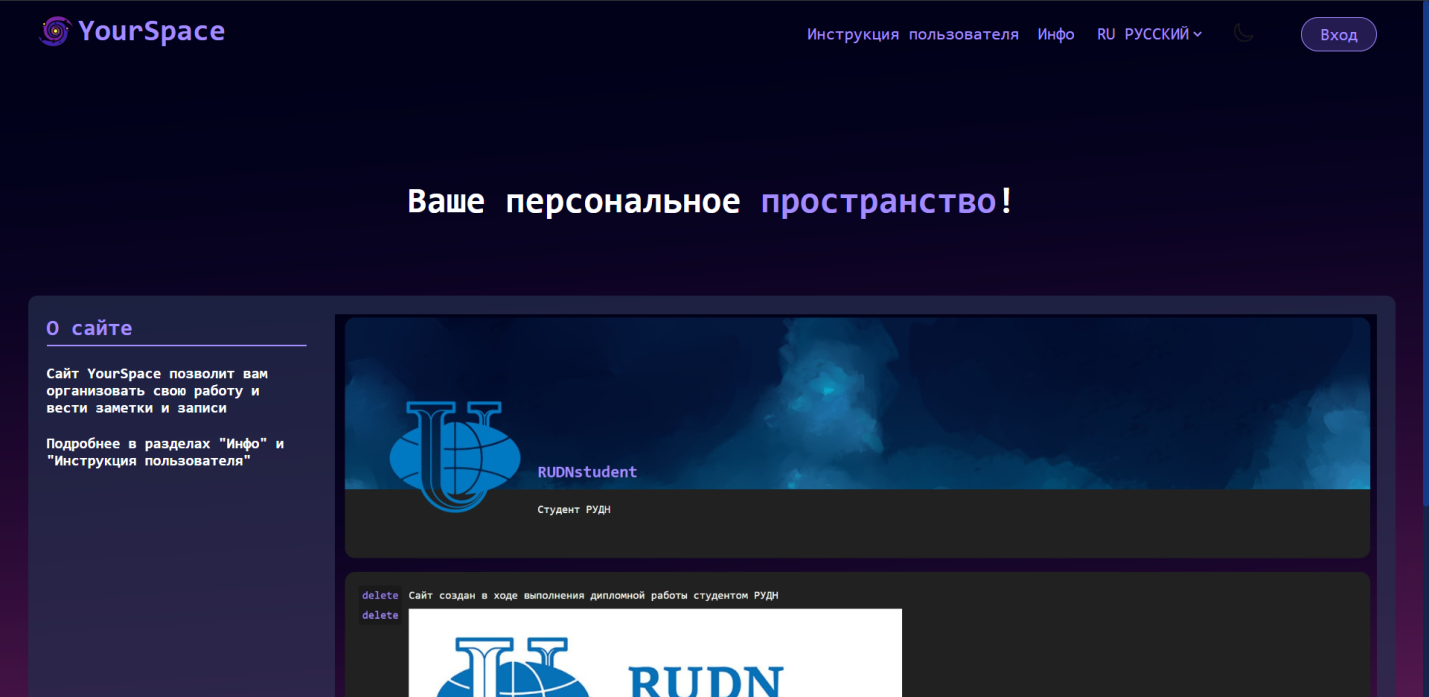


Рисунок 4. Главная страница сайта[[4]](#footnote-4)

## 3.5 Настройка моделей

Фреймворк Django по умолчанию предоставляет стандартную модель пользователя и её нужно лишь дополнить.

Сначала в файле models.py импортируются необходимые модули:

from django.db import models

from django.urls import reverse

from django.contrib.auth.models import AbstractUser

Далее стандартная модель пользователя User дополняется в соответствии с ранее представленной диаграммой с помощью AbstractUser:

class User(AbstractUser):

slug = models.SlugField(unique=True)

descr = models.TextField(null=True, blank=True)

profile\_pic = models.ImageField(null=True, blank=True, upload\_to="images/profile/")

cover\_pic = models.ImageField(null=True, blank=True, upload\_to="images/profile/")

Далее создаются связанные с данной таблицей модели.

Модель ProfileContent, отвечающая за вывод списка объектов, добавленных пользователем:

class ProfileContent(models.Model):

profile = models.ForeignKey(User, on\_delete=models.CASCADE)

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

conttype = models.ForeignKey('ContentType', null=True, on\_delete=models.SET\_NULL)

Дополняем данную модель следующей функцией, которая позволит отображать названия объектов модели в панели администратора:

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

return self.name

Модель ContentType, содержащая список типов данных, доступных для добавления пользователем на сайт:

class ContentType(models.Model):

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

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

return self.typename

Модель TextContent, содержащая текстовые поля, добавленные пользователями:

class TextContent(models.Model):

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

content = models.TextField(null=True, blank=True)

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

return self.name

Модель ImgContent, содержащая картинки, добавленные пользователями:

class ImgContent(models.Model):

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

user = models.ForeignKey(User, null=True, on\_delete=models.SET\_NULL)

content = models.ImageField(null=True, blank=True, upload\_to=user\_directory\_path)

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

return self.name

Далее функция, создающая для каждого пользователя папку в папке MEDIA, куда будут сохраняться картинки данного пользователя:

def user\_directory\_path(instance, filename):

MEDIA\_ROOT/user\_<id>/<filename>

return 'user\_{0}/{1}'.format(instance.user.id, filename)

Так же создается таблица для информации о сайте:

class Info(models.Model):

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

slug = models.SlugField(unique=True)

content= models.TextField()

is\_pub = models.BooleanField(default=True)

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

return self.title

Так же дополняем данную модель функцией, которая создаст для каждого объекта уникальный адрес на основе поля slug:

def get\_absolute\_url(self):

return reverse('info\_focus', kwargs={'slug': self.slug})

Изменения в структуре базы данных применяются следующими командами в командной строке:

Первая команда создаст миграцию – файл, который содержит в себе изменения, которые необходимо внести в структуру базы данных. Эти файлы создаются для каждого изменения и хранятся для того что бы в случае неудачного изменения структуры БД можно было произвести откат к прежней версии.

manage.py makemigrations

Следующая команда применяет изменения в файле миграции:

manage.py migrate

После исполнения данных команд создается база данных со всеми прописанными в файле models.py моделями и другими стандартными моделями.

Далее создается страница Инфо, которая содержит основную информацию о сайте.

Далее прописывается класс представления DetailView:

class info\_focus(DetailView):

model = Info

template\_name = 'home/infofocus.html'

context\_object\_name = "post"

def get\_context\_data(self, \*, object\_list=None, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['title'] = context['post']

return context

def get\_queryset(self):

return Info.objects.filter(is\_pub=True)

После этого создается URL в файле urls.py:

urlpatterns = [

…

path('info/<slug:slug>/', info\_focus.as\_view(), name="info\_focus"),

…

]

## 3.6 Создание тегов

В файле home\_tags прописываются теги – функции, которые могут вызываться в шаблоне. С помощью тега на страницу с информацией будет выводиться содержимое модели Info:

from django import template

from home.models import \*

register = template.Library()

@register.simple\_tag()

def getInfo(filt=None, selected=0):

if not filt:

return Info.objects.filter(is\_pub=True)

else:

return Info.objects.filter(pk\_\_gt=filt, is\_pub=True)

Условие в данной функции позволит выводить только те объекты, у которых есть галочка в поле is\_pub. Это позволит снять или с публикации какой то объект и добавить потом обратно при этом не удаляя его, например для редактирования.

Так же создаются другие теги:

Тег ThisProfileContent выбирает записи текущей страницы пользователя:

@register.simple\_tag()

def ThisProfileContent(pageuser=None):

if not pageuser:

pass

else:

return ProfileContent.objects.filter(profile=pageuser)

Тег getTextContent выбирает объект таблицы TextContent соответствующий объекту таблицы PageContent страницы пользователя:

@register.simple\_tag()

def getTextContent(name=None):

if not name:

pass

else:

return TextContent.objects.get(name=name)

Тег getImgContent выбирает объект таблицы ImgContent соответствующий объекту таблицы PageContent страницы пользователя:

@register.simple\_tag()

def getImgContent(name=None):

if not name:

pass

else:

return ImgContent.objects.get(name=name)

Тег getTypes выбирает записи таблицы ContentType:

@register.simple\_tag()

def getTypes():

return ContentType.objects.all()

## 3.7 Создание основных страниц

Данный шаблон расширяет базовый шаблон, так же импортируются все необходимые модули, директории и теги:

{% extends 'home/base.html' %}

{% load static %}

{% load i18n %}

{% load home\_tags %}

Подключение стилей данного шаблона:

{% block headlink %}

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

{% endblock%}

Вывод списка информационных статей, условие в котором отображает текущую статью не как ссылку а как текст:

<ul class="infosect">

{% for a in Info %}

{% if post.pk == a.pk %}

<li class="linkli curinfo">

<div class="stext infolink">

{{ a.title }}

<hr>

</div>

</li>

{% else %}

<li class="linkli">

<a class="infolink" href="{{ a.get\_absolute\_url }}">

{{ a.title }}

<hr>

</a>

</li>

{% endif %}

{% endfor %}

</ul>

Вывод текущей статьи:

<div class="infosect">

<h2 class="stext">{{ post.title }}</h2>

<hr>

<div class="cont">

<p>{{ post.content | linebreaks }}</p>

</div>

</div>

Далее прописываются стили страниц с информацией в файле info.css.

Готовые страницы с информацией выглядят следующим образом:

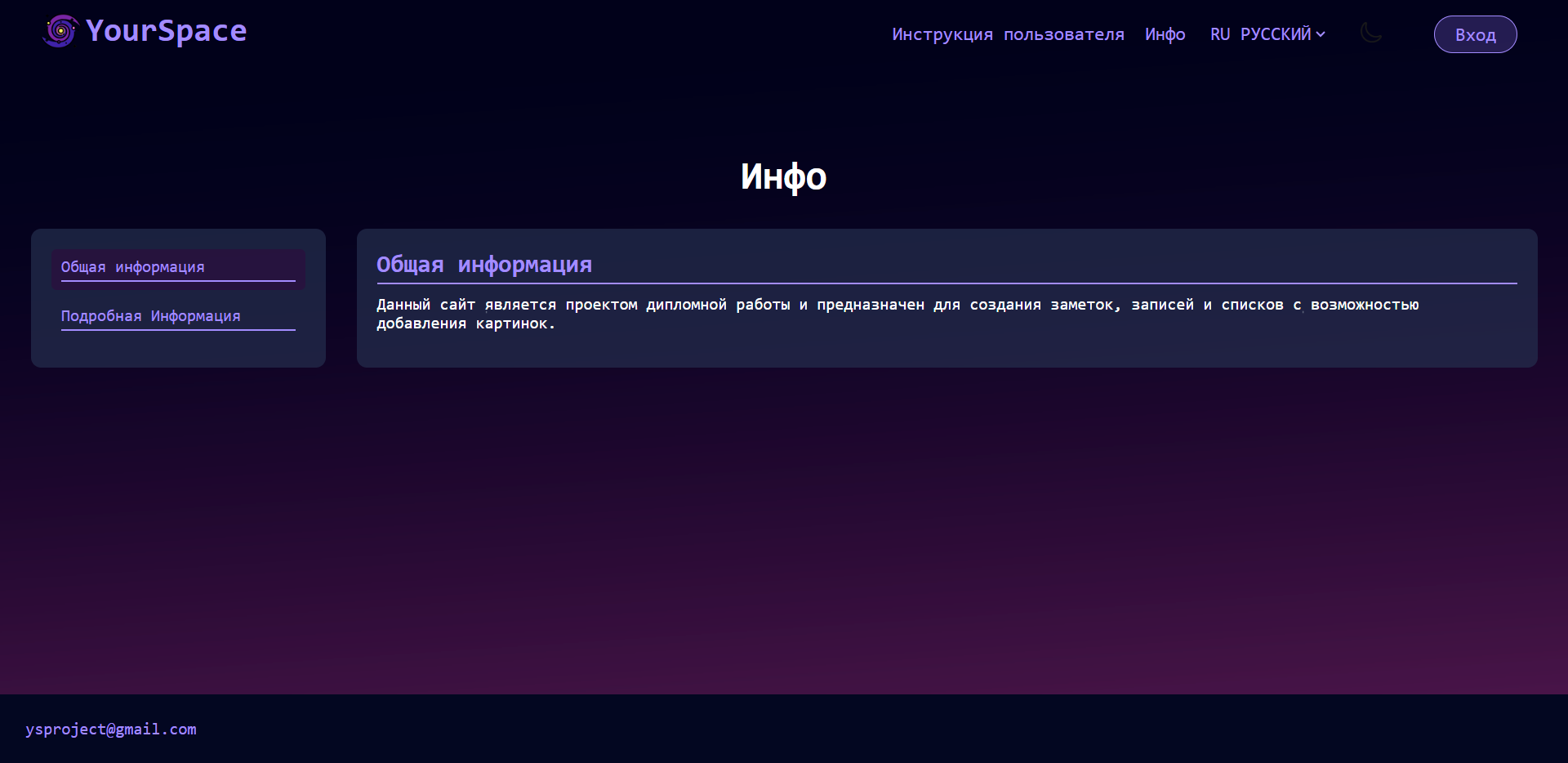


Рисунок 5. Страница с информацией о сайте[[5]](#footnote-5)

Следующим прописывается класс представления для страницы регистрации:

class signup(CreateView):

form\_class = signupform

template\_name = 'home/signup.html'

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

extra\_context = {'title': "Sign Up"}

Он дополняется функцией, которая выполняется при успешной отправки формы и создает нового пользователя, страницу для него, авторизует пользователя и перенаправляет на его страницу:

def form\_valid(self, form):

user = form.save()

user.slug = user.username

user.profile\_pic = "images/profile/defava.png"

user.cover\_pic = "images/profile/defcover.png"

user.descr = "Description"

user = form.save()

login(self.request, user)

return redirect('profile', slug = user.username)

Далее в файле forms.py импортируются необходимые модули и прописывается форма для регистрации пользователя:

from django import forms

from .models import \*

from django.contrib.auth.forms import UserCreationForm, AuthenticationForm

from django.contrib.auth.models import User

from captcha.fields import CaptchaField

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

class signupform(UserCreationForm):

username = forms.CharField(widget=forms.TextInput(attrs={'class': "logcon name"}))

email = forms.EmailField(widget=forms.EmailInput(attrs={'class': "logcon"}))

password1= forms.CharField(widget=forms.PasswordInput(attrs={'class': "logcon"}))

password2= forms.CharField(widget=forms.PasswordInput(attrs={'class': "logcon"}))

В классе Meta прописываются отображаемые поля и модель с которой связана форма:

class Meta:

model = User

fields = ('username', 'email', 'password1', 'password2',)

Создаются так же другие необходимые формы:

Форма авторизации пользователя:

class signinform(AuthenticationForm):

username = forms.CharField(widget=forms.TextInput(attrs={'class': "logcon"}))

password= forms.CharField(widget=forms.PasswordInput(attrs={'class': "logcon"}))

Форма добавления текста на страницу пользователем:

class AddText(forms.ModelForm):

name = forms.CharField(widget=forms.HiddenInput(attrs={'class': "TextField", 'value': "name"}))

content = forms.CharField(widget=forms.TextInput(attrs={'class': "TextField"}), label="")

class Meta:

model = TextContent

fields = ('name', 'content',)

Форма добавления картинок на страницу пользователем:

class AddImg(forms.ModelForm):

name = forms.CharField(widget=forms.HiddenInput(attrs={'class': "TextField", 'value': "name"}))

content = forms.ImageField(widget=forms.FileInput(attrs={'class': "ImgField"}), label="")

class Meta:

model = ImgContent

fields = ('name', 'content',)

Далее прописывается URL страницы регистрации:

urlpatterns = [

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

]

Создается шаблон страницы регистрации.

Подключение стилей:

{% block headlink %}

<link rel="stylesheet" type="text/css" href="{% static 'home/css/reg.css' %}">

{% endblock%}

Вывод ошибок:

<div class="nonferror">{{form.non\_field\_errors}}</div>

Далее прописывается форма регистрации. В ней используется csrf токен, повышающий безопасность сайта и не позволяющий скопировать форму для кражи данных пользователей.

<form method="post">

<h2>{% trans 'Sign Up' %}</h2>

{% csrf\_token %}

{% for f in form %}

<div class="inpbox">

{{ f }}

{% if f.label == "Username" %}

<label class="logcont" for="{{ f.id\_for\_label }}">{% trans 'UserName' %}</label>

{% elif f.label == "Email" %}

<label class="logcont" for="{{ f.id\_for\_label }}">Email</label>

{% elif f.label == "Password1" %}

<label class="logcont" for="{{ f.id\_for\_label }}">{% trans 'Password' %}</label>

{% else %}

<label class="logcont" for="{{ f.id\_for\_label }}">{% trans 'Confirm Password' %}</label>

{% endif %}

</div>

<div class="ferror">{{ f.errors }}</div>

{% endfor %}

<a class="logcont" href="{% url 'signin' %}">{% trans 'Or Sign In!' %}</a>

</div>

</form>

Далее прописываем стили.

Готовая страница регистрации выглядит следующим образом:

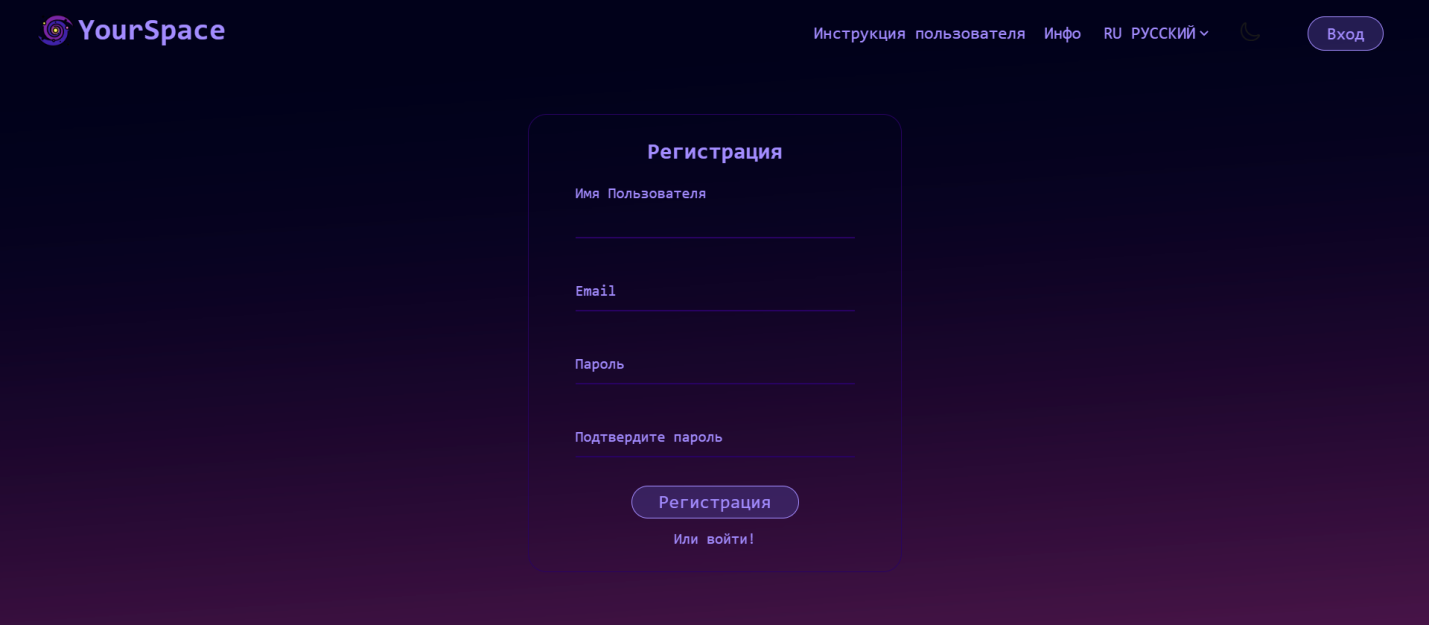


Рисунок 6. Страница регистрации[[6]](#footnote-6)

Далее создается страница авторизации.

Прописывается класс представления:

class signin(LoginView):

form\_class = signinform

template\_name = 'home/signin.html'

extra\_context = {'title': "Sign In"}

def get\_success\_url(self):

return reverse\_lazy('userredirect')

Так же создается функция представления для универсальной ссылки на текущего авторизованного пользователя:

def userredirect(request):

return redirect('profile', slug = request.user.username)

URL данных функции и класса:

urlpatterns = [

…

path('signin/', signin.as\_view(), name = 'signin'),

path('userredirect/', userredirect, name = 'userredirect'),

…

]

Далее пишется HTML код шаблона страницы входа.

Вывод сообщений об ошибках:

<div class="nonferror">{{form.non\_field\_errors}}</div>

Форма авторизации:

<form method="post">

<h2>{% trans 'Sign In' %}</h2>

{% csrf\_token %}

{% for f in form %}

<div class="inpbox">

{{ f }}

{% if f.label == "Password" %}

<label class="logcont" for="{{ f.id\_for\_label }}">{% trans 'Password' %}</label>

{% else %}

<label class="logcont" for="{{ f.id\_for\_label }}">{% trans 'UserName' %}</label>

{% endif %}

</div>

<div class="ferror">{{ f.errors }}</div>

{% endfor %}

<div class="extra">

<input type="checkbox" name="">{% trans ' Remember Me' %}

<a class="excont" href="#">{% trans ' Forget password?' %}</a>

</div>

<div class="subbox">

<input class="logcont submit" type="submit" name="" value="{% trans 'Sign In' %}"><br>

<a class="logcont" href="{% url 'signup' %}">{% trans 'Or Sign Up!' %}</a>

</div>

</form>

После этого прописываются стили страницы авторизации.

В итоге страница входа выглядит следующим образом:

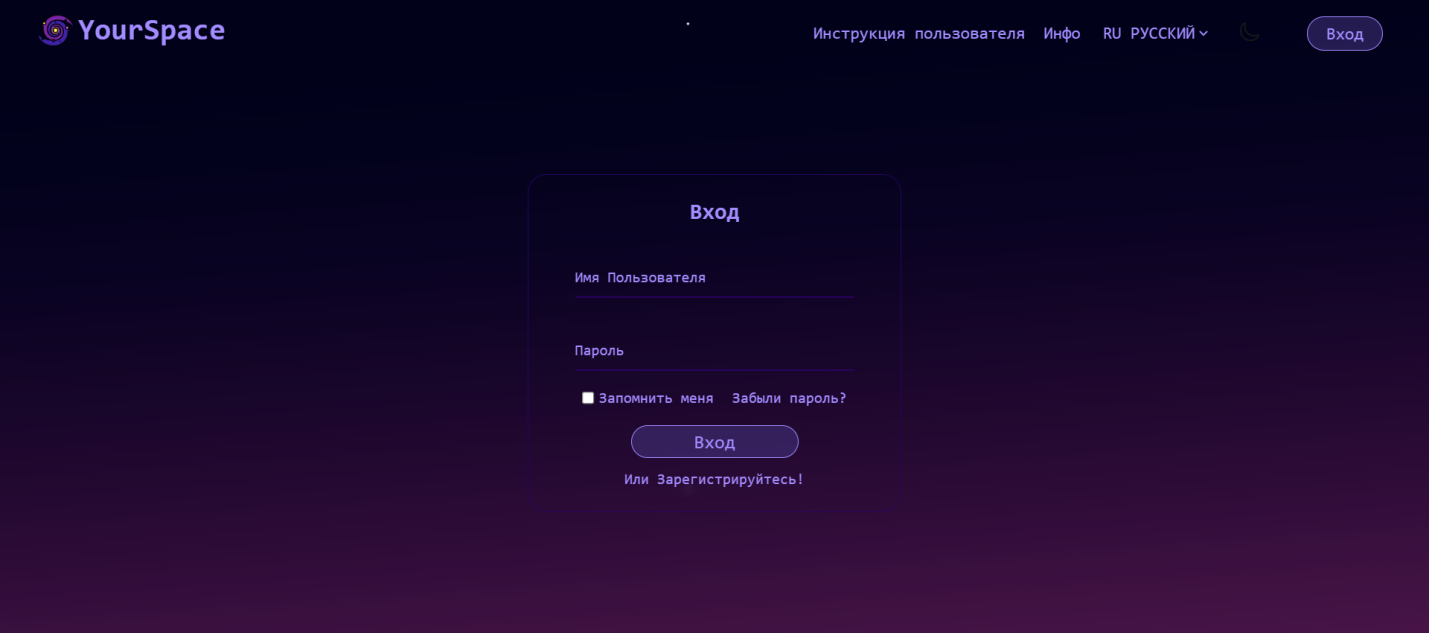


Рисунок 7. Страница авторизации[[7]](#footnote-7)

Далее создается страница пользователя.

Первым создается функция представления:

def Profil(request, slug):

p\_user = get\_object\_or\_404(User, slug=slug)

if request.method == 'POST':

form = AddText(request.POST)

if form.is\_valid():

try:

form.name="name"

f = form.save(commit=False)

f.name = "name"

f.save()

f.name = str(request.user.username)+".text"+str(f.pk)

f.save()

c = ProfileContent(profile=request.user, name=f.name, conttype=ContentType.objects.get(pk=1))

c.save()

form = AddText()

except:

form.add\_error(None, 'Ошибка добавления')

return redirect('userredirect')

form1 = AddImg(request.POST, request.FILES)

if form1.is\_valid():

try:

form1.name="name"

f1 = form1.save(commit=False)

f1.name = "name"

f1.user = request.user

f1.save()

f1.name = str(request.user.username)+".img"+str(f1.pk)

f1.save()

c1 = ProfileContent(profile=request.user, name=f1.name, conttype=ContentType.objects.get(pk=2))

c1.save()

form1 = AddImg()

except:

form1.add\_error(None, 'Ошибка добавления')

return redirect('userredirect')

else:

form = AddText()

form1 = AddImg()

context = {

'title': p\_user,

'p\_user': p\_user,

'form': form,

'form1': form1,

}

return render(request, 'home/mainprofile.html', context=context)

Далее прописывается URL:

urlpatterns = [

…

path('profile/<slug:slug>/', Profil, name = 'profile'),

…

]

Создание HTML шаблона страницы пользователя:

Подключение тегов:

{% ThisProfileContent pageuser=p\_user as ProfileContent %}

{% getTypes as type %}

Далее представлен HTML код страницы пользователя.

Верхняя часть профиля, содержащая фото, фоновое изображение, имя пользователя и описание профиля:

<div class="profiletop">

<div class="pagecover">

<img src="{{ p\_user.cover\_pic.url }}">

</div>

<div class="profinfo">

<div class="profileava">

<img src="{{ p\_user.profile\_pic.url }}">

</div>

<h2 class="stext">{{p\_user}}</h2>

<div class="descr">{{p\_user.descr}}</div>

</div>

</div>

Основное содержимое страницы:

<div class="maincontent">

{% for c in ProfileContent %}

{% if c.conttype == type.0 %}

<div class="object">

<a href="{% url 'delete' o\_id=c.pk %}"><p>delete</p></a>

{% getTextContent name=c.name as text%}

<div>{{text.content | linebreaks }}</div>

</div>

{% elif c.conttype == type.1 %}

<div class="object">

<a href="{% url 'delete' o\_id=c.pk %}"><p>delete</p></a>

{% getImgContent name=c.name as img%}

<div><img src="{{img.content.url}}"></div>

</div>

{% else %}

<div>{{c.conttype}} неизвестный тип данных</div>

{% endif %}

{% endfor %}

Формы добавления содержимого на страницу:

<div class="forms">

<select class="seltype">

{% for t in type %}

<option>{{t}}</option>

{% endfor %}

</select>

<form name="textform" class="hform show textform" method="post">

{% csrf\_token %}

{{ form }}

</form>

<form name="imgform" class="hform imgform" method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form1 }}

<input type="submit" name="">

</form>

</div>

</div>

Далее прописываются стили страницы пользователя.

Готовая страница пользователя выглядит следующим образом:

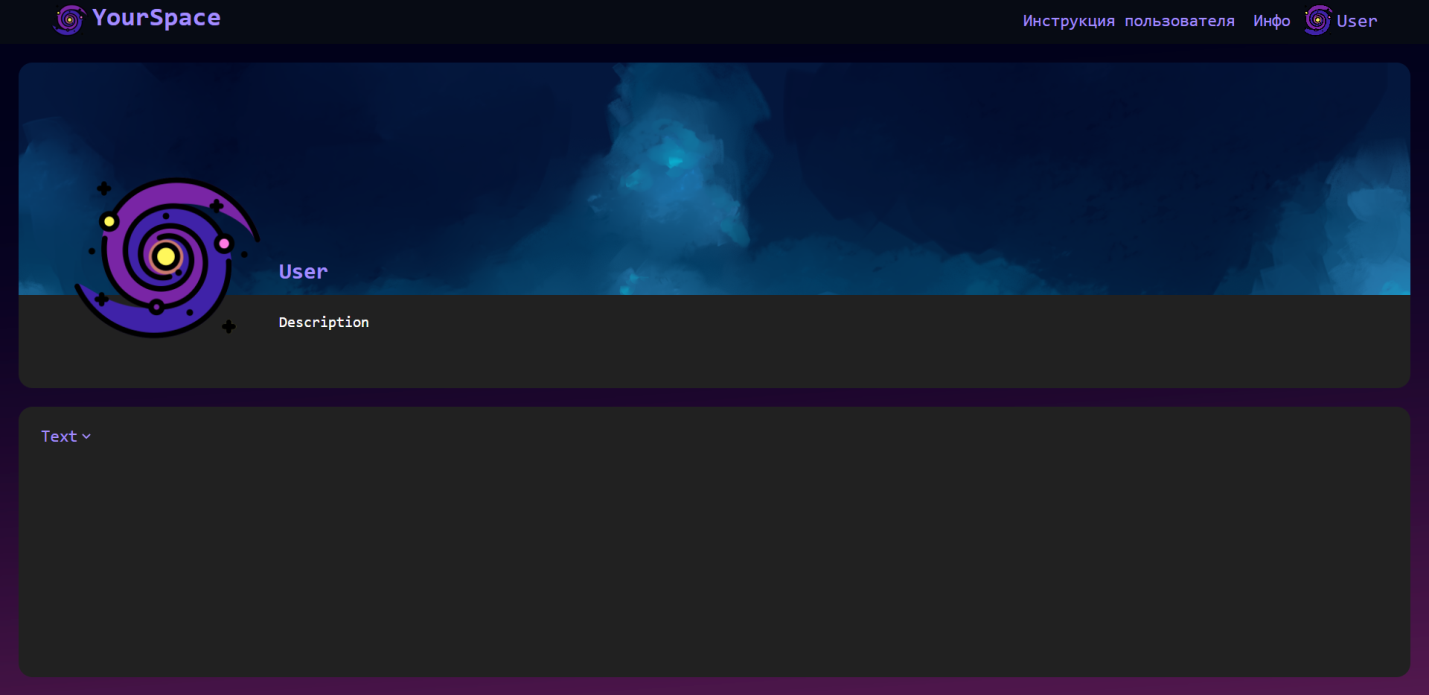


Рисунок 8. Страница пользователя[[8]](#footnote-8)

Далее представлено тестирование основных функций: добавление текста, добавление изображений, удаление текста и изображений.

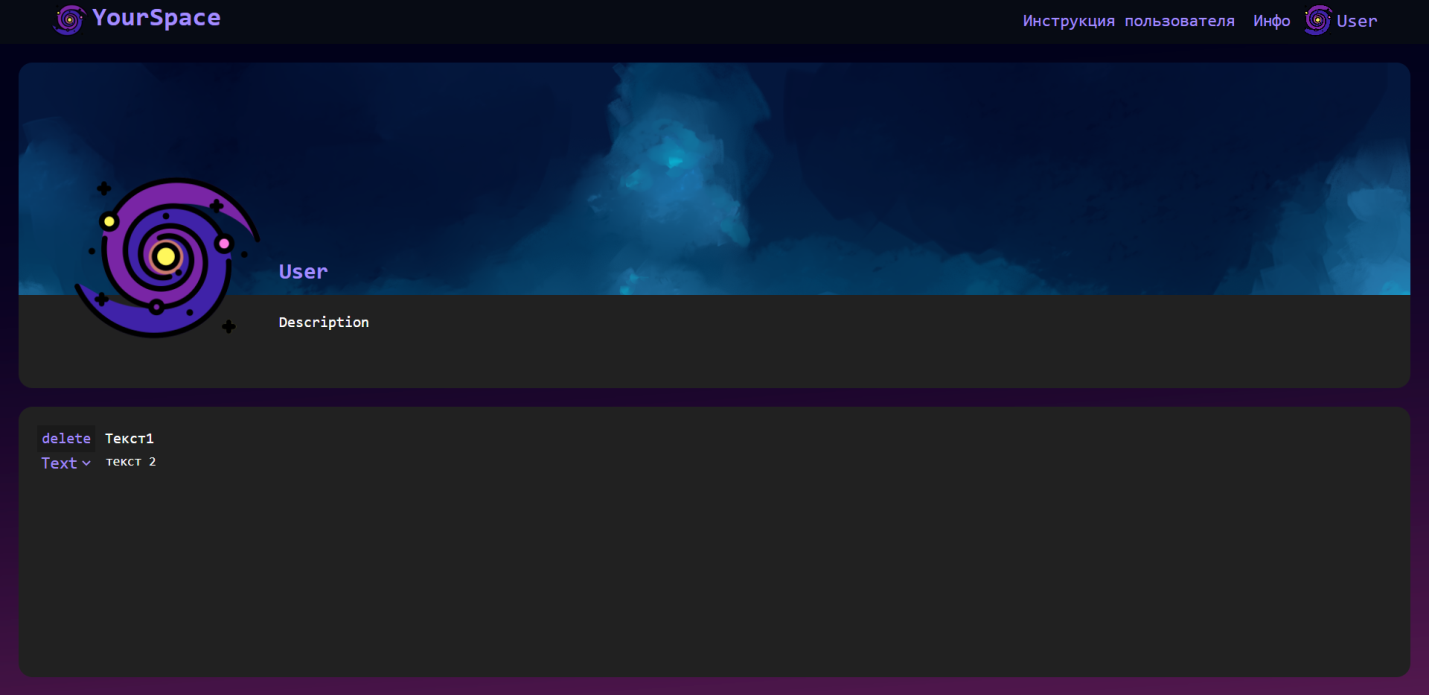
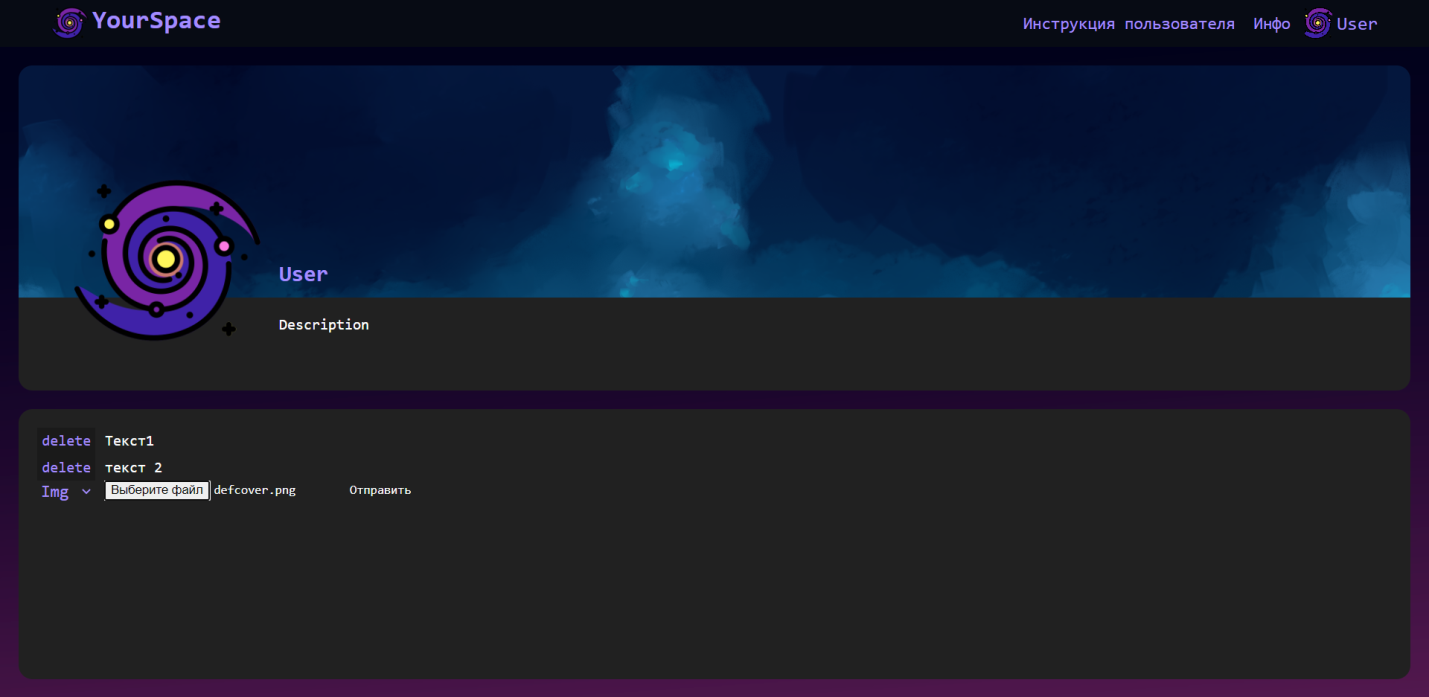


Рисунок 9. Добавление текста на страницу[[9]](#footnote-9)

 Рисунок 10. Добавление картинок на страницу[[10]](#footnote-10)

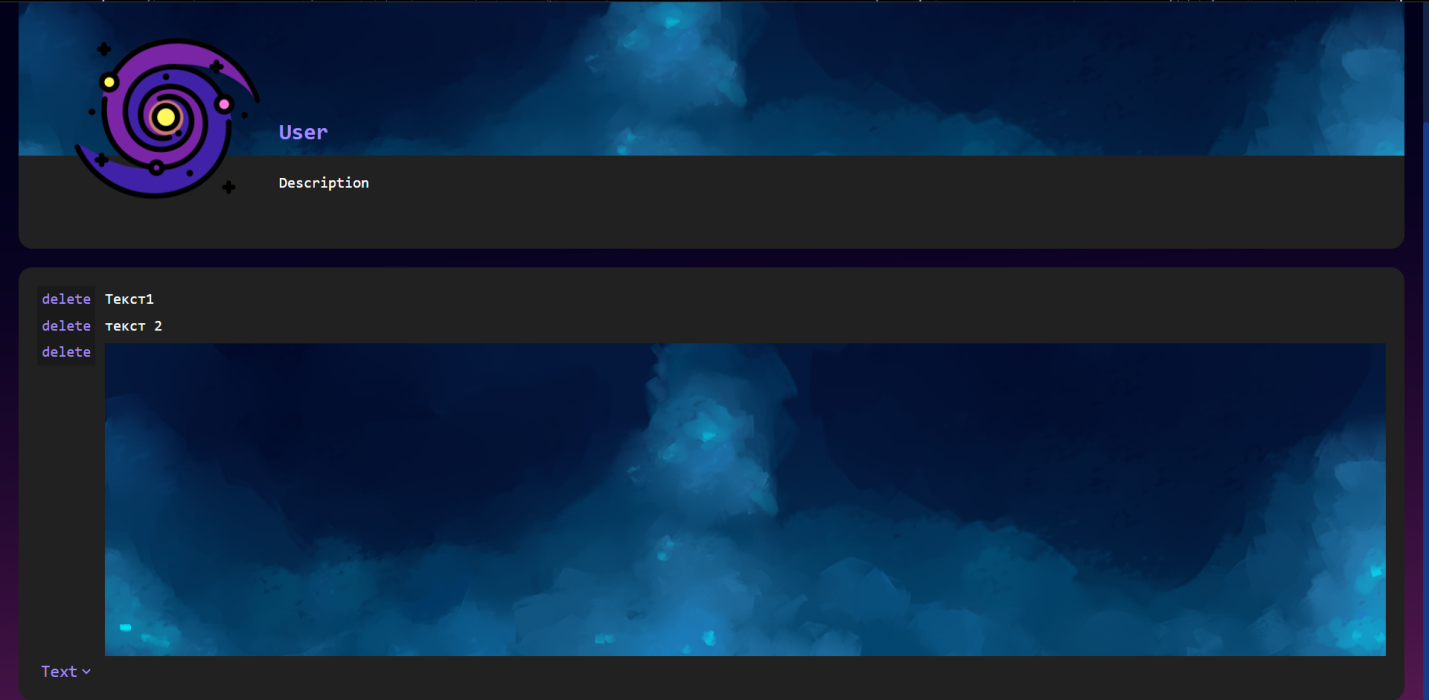


Рисунок 11. Отображение картинки пользователя на странице[[11]](#footnote-11)

Далее создается функция представления для удаления содержимого страниц. Условие в ней удаляет запись из списка содержимого страницы и соответствующее содержимое либо из таблицы с текстом либо из таблицы с картинками.

def delete(request, o\_id):

obj = ProfileContent.objects.get(pk=o\_id)

objname = obj.name

if obj.conttype == ContentType.objects.get(pk=1):

tobj = TextContent.objects.get(name=objname)

obj.delete()

tobj.delete()

elif obj.conttype == ContentType.objects.get(pk=2):

iobj = ImgContent.objects.get(name=objname)

obj.delete()

iobj.delete()

return redirect('profile', slug = request.user.username)

Так же прописывается функция выхода из учетной записи:

def logout(request):

django\_logout(request)

return redirect('home')

Кнопки удаления и выхода являются ссылками на данные функции и для их работы необходимо так же указать URL в файле urls.py:

urlpatterns = [

…

path('logout/', logout, name = 'logout'),

path(r'^(?P<o\_id>\d+)delete/', delete, name = 'delete'),

]

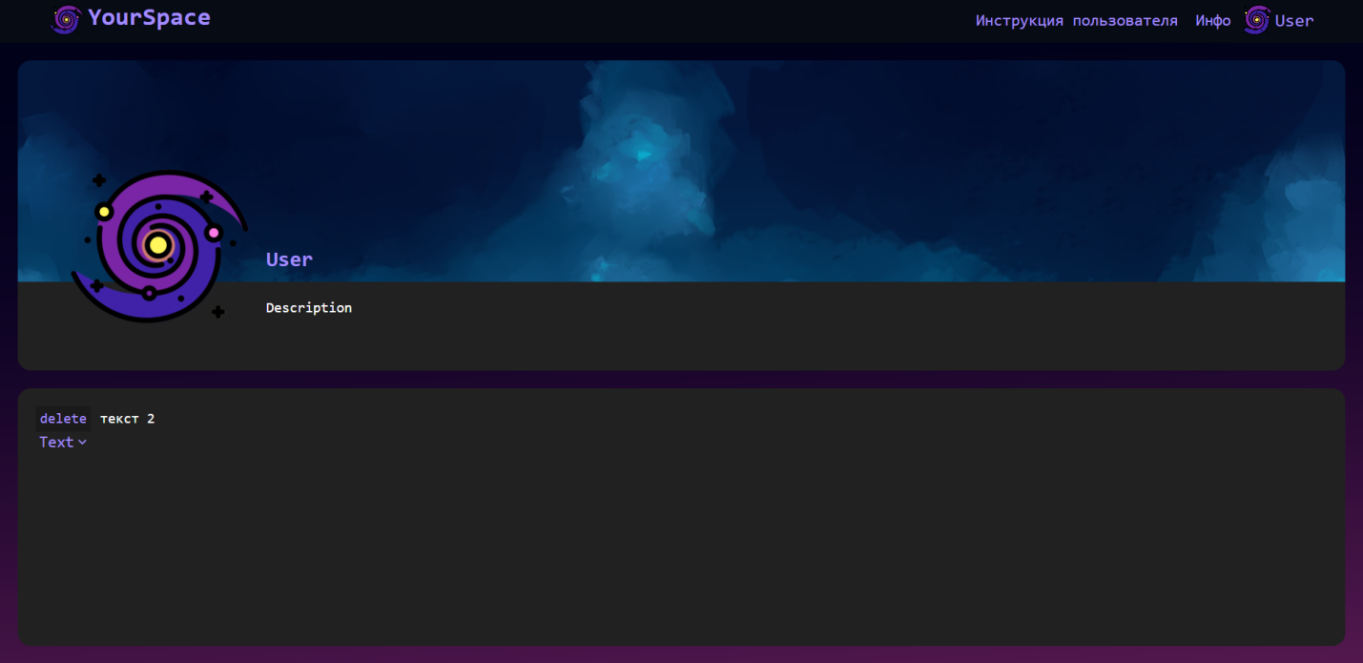


Рисунок 12. Удаление объектов со страницы[[12]](#footnote-12)

# ЗАКЛЮЧЕНИЕ

Разрабатываемая информационная система была реализована в виде сайта. Данная информационная система решает проблему организации деятельности и информации пользователя. В ходе дипломной работы была выполнена цель разработать ИС «Рабочее пространство», а так же выполнены следующие задачи:

Рассмотрены типы информационных систем

изучена предметная область, для которой разработан ресурс,

выбраны средства и технологии для разработки,

спроектирован и разработан сайт для предприятия.

Внедрение данного программного продукта позволяет пользователям вести заметки и записи а так же увеличить эффективность работы.

В данном программном продукте был реализован удобный и простой пользовательский интерфейс. Также имеется возможность отладить или усовершенствовать продукт, в случае обнаружения недоработок или необходимости добавить дополнительные функции.

Выполненная дипломная работа несёт высокую ценность не только благодаря разработанному решению, но и благодаря проделанной практической работе, которая сделала большой вклад в развитие практической деятельности выпускника, и отражает готовность выпускника к профессиональной деятельности.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.201-78 Единая система программной документации (ЕСПД). Техническое задание. Требования к содержанию и оформлению (с Изменением N 1). [Электронный ресурс]/ АО «Кодекс» 2020 - Электрон. дан. – Режим доступа: http://docs.cntd.ru/document/1200007648 (дата обращения 26.11.2022)

2. Статья «Инфомационная система» [Электронный ресурс] /ru.wikipedia.org –2023 – Электрон. дан.- Режим доступа: https://ru.wikipedia.org/wiki/Информационная\_система (дата обращения 15.01.2023)

3. Официальная документация Django [Электронный ресурс] /docs.djangoproject.com –2023 – Электрон. дан.- Режим доступа: https://docs.djangoproject.com/en/4.2/ (дата обращения 16.01.2023)

4. Django 3 для python (уроки): серия видео на платформе youtube [Электронный ресурс] – Электрон. дан.- Режим доступа: https://www.youtube.com/playlist?list=PLA0M1Bcd0w8xO\_39zZll2u1lz\_Q-Mwn1F (дата обращения 09.01.2023)

5. Официальный сайт HTML Academy [Электронный ресурс]/htmlacademy.ru/ –2022 – Электрон. дан.- Режим доступа: https://htmlacademy.ru/ (дата обращения 06.12.2022)

6. Официальный сайт MDN Web Docs [Электронный ресурс]/developer.mozilla.org/ –2023 – Электрон. дан.- Режим доступа: https://developer.mozilla.org/ru/ (дата обращения 10.03.2023)

7. Официальный сайт Stack Overflow [Электронный ресурс]/stackoverflow.com/ –2023 – Электрон. дан.- Режим доступа: https://stackoverflow.com/ (дата обращения 20.01.2023)

8. Официальный сайт Хабр [Электронный ресурс]/ qna.habr.com/ –2023 – Электрон. дан.- Режим доступа: https://qna.habr.com/ (дата обращения 15.02.2023)

1. Создано автором [↑](#footnote-ref-1)
2. Создано автором [↑](#footnote-ref-2)
3. Создано автором [↑](#footnote-ref-3)
4. Создано автором [↑](#footnote-ref-4)
5. Создано автором [↑](#footnote-ref-5)
6. Создано автором [↑](#footnote-ref-6)
7. Создано автором [↑](#footnote-ref-7)
8. Создано автором [↑](#footnote-ref-8)
9. Создано автором [↑](#footnote-ref-9)
10. Создано автором [↑](#footnote-ref-10)
11. Создано автором [↑](#footnote-ref-11)
12. Создано автором [↑](#footnote-ref-12)