ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА ОБЛАЧНОГО СЕРВИСА ДЛЯ ГЕМАТОЛОГИЧЕСКОГО АНАЛИЗА ИЗОБРАЖЕНИЙ КРОВИ

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем
очной формы обучения, группы 07001402
Выгоняйло Виктора Родионовича

Научный руководитель
к.т.н, доцент Михелев В.М.

БЕЛГОРОД 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6

1.1 Облачное хранилище данных 6

1.2 Веб-приложение как веб-интерфейс 8

1.3 Масштабируемость и оптимизация веб-приложений 9

1.4 Объектно-реляционное отображение и реляционные СУБД 13

1.5 Веб-фреймворк Django 14

1.6 Постановка задачи 17

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОБЛАЧНОГО СЕРВИСА ДЛЯ ГЕМАТОЛОГИЧЕСКОГО АНАЛИЗА ИЗОБРАЖЕНИЙ КРОВИ 19

2.1 Проектирование инфраструктуры облачного сервиса 19

2.2 Проектирование веб-приложения 20

2.2.1 Определение основного функционала 21

2.2.2 Проектирование базы данных 22

2.3 Реализация спроектированного приложения 28

2.3.1 Программирование моделей 29

2.3.2 Задание маршрутов 32

2.3.3 Программирование шаблонов 33

2.3.4 Программирование форм 35

2.3.5 Программирование контроллеров 36

2.3.6 Программирование middleware 37

2.3.7 Создание панели администрирования 39
ГЛАВА 3. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ВЕБ-ПРИЛОЖЕНИЯ... 41
3.1 Определение необходимых тестовых условий 41
3.2 Тестирование системы аутентификации и хранения сессий 43
3.3 Тестирование загрузки, хранения и обработки изображений 46
ЗАКЛЮЧЕНИЕ 49
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 50
ПРИЛОЖЕНИЕ 1 52
ПРИЛОЖЕНИЕ 2 53

ВВЕДЕНИЕ

В современном мире информационные технологии активно внедряются в различные сферы человеческой деятельности. Информатизация деятельности человека способствует увеличению цифровой информации, которая нуждается в хранении, обработке и сортировке. В том числе это затрагивает и сферу здравоохранения. Единая информационная система, электронная система учета пациентов, электронные медицинские карты, цифровые снимки и другие результаты информатизации сферы здравоохранения.

Всё это является источником большого количества цифровых данных, которые без должной обработки не будут эффективно использованы. Для хранения и удобного доступа к большим объемам информации отлично подходят “облачные” сервисы.

Темой данной дипломной работы является создание “облачного” сервиса для гематологического анализа крови. Который отлично подойдет для задачи хранения и обработки медицинских снимков крови. Данный сервис направлен на решение следующих задач:

— хранение изображений;
— помощь в анализе изображений;
— унификация способа доступа к инструменту анализа;
— автоматизация процесса анализа.

Целью данной дипломной работы является разработка и реализация информационной инфраструктуры “облачного” сервиса для хранения и обработки медицинских данных, а также реализация веб-интерфейса веб-приложения.

Поставленная цель подразумевает решение следующих задач:

1. анализ существующих принципов построения архитектуры и технологий для создания веб-приложений

2. изучение возможности хранения данных в облачном сервисе, как основа для разрабатываемой системы;
3. выбор программных средств;
4. проектирование и реализация веб-сервиса;
5. тестирование разработанного приложения на корректность работы;
6. тестирование надежности хранения данных;
7. анализ данных, полученных в результате тестирования;
8. формирование вывода по проделанной работе.

Дипломный проект состоит из трех глав.

В первой главе “Анализ предметной области” производится постановка проблемы хранения больших объемов медицинских данных для последующей обработки. Проводится сравнительных анализ существующих технологий хранения данных. Глава оканчивается постановкой задачи для выполнения.

Во второй главе “Проектирование и программная реализация облачного сервиса для гематологического анализа изображений крови” описывается процесс проектирования информационной системы и реализации веб-приложения.

Третья глава “Тестирование разработанного веб-приложения” посвящена тестированию разработанного функционала. Отдельно рассматривается функционал, связанный с пользователями и с обработкой изображений.

Данная работа содержит 51 страниц, 12 рисунков, 3 таблицы и приложение, количество использованных источников литературы – 11.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Облачное хранилище данных

Облачное хранилище данных – модель онлайн-хранилища, в котором хранимые данные располагаются на многочисленных удаленных серверах в сети Интернет, предоставляемых в пользование клиентам, как правило, третьей стороной. При такой модели хранения данных, приобретаемые или взятые в аренду сервера, количество которых или внутренняя структура не видна конечному клиенту. В этом плане это отличается от модели хранения на собственных выделенных серверах. Все данные располагаются и обрабатываются в так называемом “облаке”, которое для клиента выглядит, как единый виртуальный сервер. Но физическое местоположение серверов может быть удаленно друг от друга.

Многочисленные IT-компании ассоциируют с “облачными” сервисами ряд проблем, связанных с безопасностью, а именно:

— Одним из важнейших критериев при работе с “облаком” является вопрос конфиденциальности и приватности данных при пересылке и хранении. Если данные клиента при пересылке не будут защищены, то их может просматривать провайдер, если системы защиты провайдера будут взломаны злоумышленниками, то данные пользователей могут стать общественным достоянием.

— Надежность своевременного получения и доступность данных в “облаке” зависит от множества критериев: каналы передачи данных, надежность последней мили, качество соединения, предоставляемого Интернет-провайдером, доступность сервиса в текущий момент времени.

— Общая производительность при хранении и обработке данных на удаленном сервере может быть ниже, чем при обработке и хранении данных в локальном хранилище.

— Дополнительная абонентская плата, взымаемая за дополнительные возможности при пользовании сервисом (увеличения объема доступного хранилища, передача больших файлов и т. д.).

— существует риск ликвидации данных клиента, если компания, предоставляющая услуги, перестанет существовать.

Несмотря на все эти нюансы безопасности, популярность “облачных” сервисов растет, так как возможность удаленной работы с файлами имеет ряд своих преимуществ:

— возможность доступа к данным с любого устройства, имеющего доступ в сеть Интернет;

— возможность организации совместной работы с данными;

— высокая вероятность сохранения данных, даже в случае аппаратных сбоев;

— абонентская плата взымается только за место в хранилище, которое фактически используется, а не за аренду всего сервера, все ресурсы которого клиент может никогда не задействовать;

— уменьшение издержек производства, так как клиент не занимается поддержкой и обслуживанием инфраструктуры хранилища данных;

— провайдер сервиса не вовлекает клиента в процесс резервирования и сохранения целостности данных пользователя;

— использование облачного шлюза.

Облачный шлюз – технология, которая используется для более удобного предоставления пользования “облака” клиенту. Это достигается по средствам специального программного обеспечения, пространство памяти в хранилище может быть предоставлено пользователю, как локальный диск на собственном устройстве. Таким образом достигается абсолютная прозрачность для клиента

при работе с “облаком”. При наличии достаточно быстрой связи с “облаком”, клиент может не ощутить, что работает на самом деле с данными на удаленном сервере [1].

1.2 Веб-приложение как веб-интерфейс

Веб-приложение – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером по средствам веб-браузера. Веб-приложение устроено так, что его логика его работы распределена между веб-сервером и веб-браузером. Хранение данных осуществляется на сервере, а данные передаются по сети. Благодаря такому подходу клиенты не зависят от конкретного программного обеспечения, поэтому веб-приложения являются кроссплатформенными службами.

Особенности веб-браузеров заключаются в том, что функционал выполняется одинаково, независимо от программного обеспечения пользователя, это дает существенное преимущество при построении веб-приложений. Не возникает необходимости создавать различные версии приложений для различных операционных систем. Но различные реализации HTML, CSS, DOM и других спецификаций в веб-браузерах, могут создать проблемы при разработке и дальнейшей поддержке веб-приложений.

Веб-приложение состоит из, как правило, “тонкого” клиента и серверной части, тем самым реализуя технологию “клиент-сервер”. Клиентская часть реализует интерфейс пользователя, формирует сообщения запросов к серверу и обрабатывает ответы от него. Создание серверной части веб-приложения может выполнятся при помощи множества технологий и различных языков программирования, способные осуществлять вывод данных в стандартную консоль.

Таким образом, веб-интерфейсом для “облачного” сервиса может являться клиентская часть веб-приложения.

Веб-интерфейс представляет собой веб-страницу или их совокупность, которые предоставляются пользователю для взаимодействия с сервером или устройством посредствам протокола HTTP.

Как правило, в общем виде, “облачный” веб-интерфейс не обладает сложным функционалом и предоставляет примерно следующие возможности: регистрация пользователя, после которой клиенту предоставляется место на жестком диске; предоставление пользователю хранилище для различного вида информации, она будет доступна в любой точке мира, с любого устройства, где сеть подключение к сети Интернет [2].

1.3 Масштабируемость и оптимизация веб-приложений

Масштабируемость – в электронике и информатике означает способность системы, сети или процесса справляться с увеличением рабочей нагрузки при добавлении ресурсов.

Масштабируемость является важным аспектом любых электронных систем, баз данных и программных комплексов. Тоже самое касается и веб-приложений. Их кроссплатформенность и простота использования делает их весьма популярными. А значит существует потенциальная возможность высокой нагрузки.

Система называется масштабируемой, если она способна увеличивать свои производительные мощности пропорционально дополнительно затраченным ресурсам. Этот параметр можно оценить отношением прироста производительности системы к количеству дополнительно затраченных ресурсов. Чем ближе это отношение к единице, тем лучше. Также под масштабируемостью понимают наличие возможности добавление

дополнительных ресурсов без структурных изменений центрального узла системы [3].

Разделяют вертикальное и горизонтальное масштабирование.

Горизонтальная масштабируемость – разнесение структурных компонентов системы на отдельные физические устройства, и увеличение количества серверов, выполняющих одну и ту же функцию одновременно. Этот способ масштабирование подразумевает добавление новых системных узлов для общего увеличения производительности.

Стоит отметить, что в системах с плохой масштабируемостью увеличение ресурсов не дает желаемого прироста производительности, приводя лишь к незначительному увеличению производительности. В таких случаях стоит прибегать к вертикальному масштабированию.

Вертикальное масштабирование заключается в увеличении производительности каждого компонента системы с целью повышения общей производительности. Подразумевается замена компонентов системы на более мощные и быстрые. Быстрый рост требований и развитие технологий вынуждает этот процесс. Этот способ масштабирования является самым простым, так как не требует изменения прикладных программ.

Рост популярности веб-приложений приводит увеличению затрат ресурсов. Изначально, с возрастающей нагрузкой можно бороться с помощью оптимизации алгоритмов и архитектуры приложения [4].

Большую часть веб-приложений можно назвать распределёнными, так как в их архитектуре можно выделить как минимум три слоя: веб-сервер, бизнес-логика, хранение данных. На рисунке 1.1 изображена архитектура веб-приложения, где бизнес-логика представляет приложение, а хранимые данные представлены базой данных.

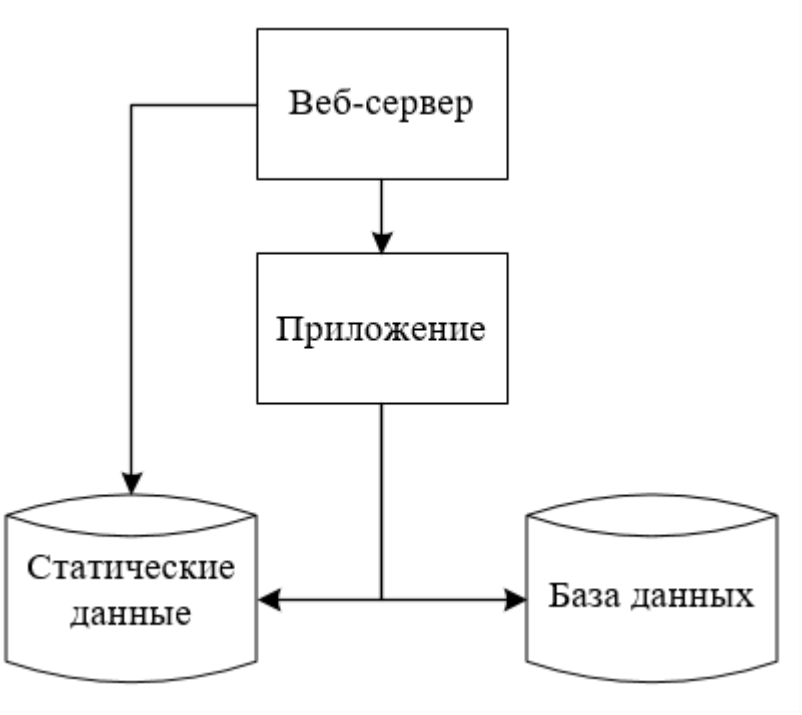

Рис. 1.1. Архитектура веб-приложения

На рисунке 1.1 представлены различные слои веб-приложения и каждый из них может быть масштабирован.

Одним из первых шагом оптимизации является перенос базы данных приложения на отдельный хост. Чтобы приложение и база данных находились на разных устройствах [5].

Подходя к вопросу масштабирования веб-приложения стоит определить, какой из слоев системы функционирует медленнее всего. Для этого можно воспользоваться различными утилитами, которые смогут оценить потребление ресурсов процессора, памяти и потребления диска. Как правило, это зависит от архитектуры приложения, но чаще всего самым проблемным слоем является база данных и программный код. Если веб-приложение рассчитано на работу с большими объемами данных, как это чаще всего бывает с “облачными” сервисами, то “узким местом” будет хранение статических данных.

Проблемы с масштабируемостью баз данных, как правило, делятся на два класса: необходимость хранения большого количества данных и производительность.

Как и в случае с распределением приложения и его базы данных на разные хосты, ради уменьшения нагрузки, точно также можно разнести базу данных на несколько хостов. При этом появляется проблема синхронизации базы данных, которую можно решить, реализовав схему master-slave (рис. 1.2) с синхронной или асинхронной репликацией данных. В случае с PostgreSQL реализовать синхронную репликацию можно с помощью Slony-I, асинхронную – PgPool-II или WAL (9.0). Решить проблему разделения запросов чтения и записи, а также балансировки нагрузку между имеющимися slave'ами, можно путём настройки специального слоя доступа к базе данных (PgPool-II).

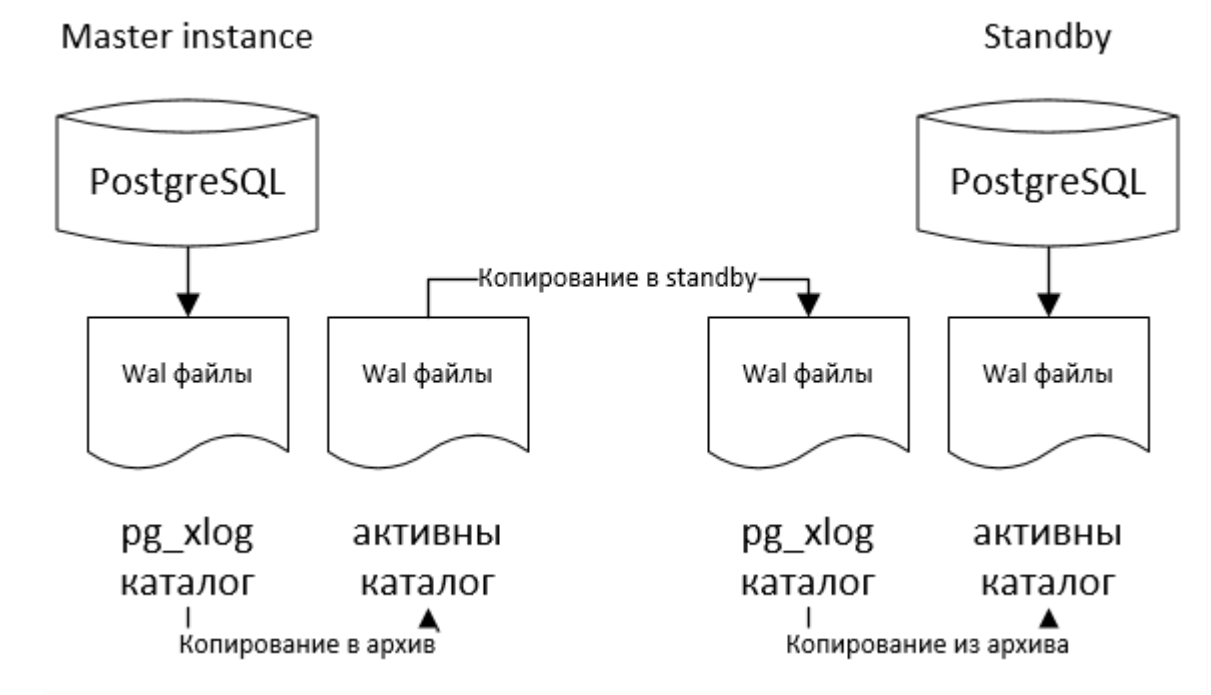

Рис. 1.2. Схема master-slave в PosgreSQL

При необходимости хранения больших объемов статических данных возникает вопрос масштабирования файловой системы, выделяются две проблемы: ограниченность памяти и скорость доступа к данным. Проблему с нехваткой памяти можно решить несколькими способами: использование

распределенной файловой системы, хранение данных в базе данных с поддержкой шардинга или его организации на уровне программного кода.

Чем больше становится система приложения, тем больше она требует постоянного мониторинга, чтобы следить за нагрузкой и работой узлов. Не существует универсального решения вопроса оптимизации и масштабирования. Но вышеперечисленные методы являются базовыми решениями со своими плюсами и минусами [6].

1.4 Объектно-реляционное отображение и реляционные СУБД

ORM – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая “объектную виртуальную базу данных”.

Особенность технологии в организации работы с данными в базе данных в терминах классов, а не её таблицами, затем преобразовать термины и данные классов в данные, пригодные для хранения в базе данных. На эту технологию также ложится процесс обеспечения интерфейса для CRUD-операций над данными. Этот процесс избавляет программиста от необходимости написания SQL-кода для взаимодействия с СУБД.

При использовании реляционной базы данных с целью хранения объектно-ориентированных данных происходит семантический разрыв. Вынуждая программистов создавать программное обеспечение для обработки данных в объектно-ориентированном виде, которое должно уметь сохранять эти данные в реляционной форме. Такое преобразование данных из одного формата в другой сильно снижают производительность системы, накладывая ограничения друг на друга.

Реляционные базы данных используют совокупность таблиц для представления простых данных. Часто, для хранения целостной информации об объекте в реляционной базе данных используется несколько таблиц. Это приводит к дополнительным затратам на объединение таблиц, для получения общей информации, относящейся к объекту.

Системы управления реляционными базами данных не реализуют реляционного представления физического уровня связей, поэтому последовательное выполнение нескольких запросов может быть слишком затратным процессом.

Некоторые реализации ORM автоматически синхронизируют загруженные в память объекты с базой данных, после создания SQL-запроса происходит процесс преобразования полученных данных в объект, путем копирования данных в поля объекта, как и во всех других реляционных ORM. После этого объект следит за всеми изменениями своих параметров, чтобы в любой момент занести их обратно в базу данных.

Объектно-ориентированный доступ является более эффективным при работе с малыми объемами, тем самым сокращая семантические провалы между объектной и реляционной формами данных. Но реляционные базы данных показывают высокую производительность при глобальных запросах, которые охватывают большой участок базы данных [7].

1.5 Веб-фреймворк Django

Django – свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC.

MVC (“Модель-Вид-Контроллер”) – схема разделения данных приложения, интерфейс пользователя и управление логикой, на три отдельных

компонента: модель, представление или вид и контроллер, благодаря этому, модификация каждого компонента происходит независимо друг от друга.

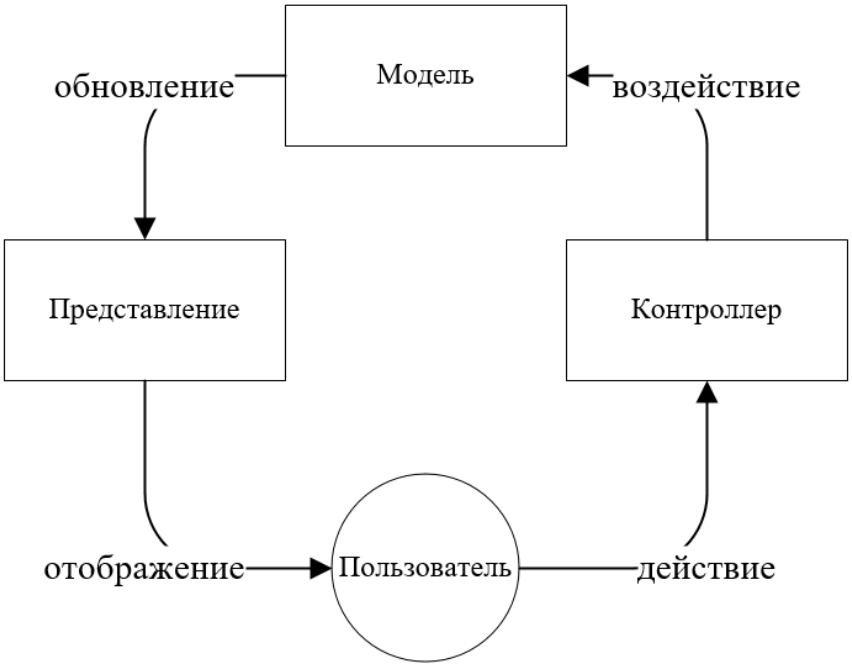

Рис. 1.3. Шаблон проектирования “Модель-Представление-Контроллер”

На рисунке 1.3 проиллюстрирован принцип работы схемы разделения данных:

— модель (Model) отвечает за предоставление данных, реагирует на команды контроллера, ответно реагируя изменением состояния;

— представление (View) отвечает за отображение данных модели для пользователя, реагируя на изменения модели;

— контроллер (Controller) интерпретирует действия пользователя, оповещает модель о необходимости изменений.

Веб-приложения на Django состоят, как правило, из одного или нескольких приложений (модулей), которые должны быть независимыми и взаимоподключаемыми. Это является одним из существенных архитектурных отличий от этого фреймворка от некоторых других. Также, существенным отличием является явная конфигурация URL с помощью регулярных выражений, а не выводится автоматически из структуры моделей контроллеров.

При работе с базой данных Django использует собственный ORM, в котором модели данных описываются с помощью классов Python, и по их описанию генерируются сущности базы данных.

Веб-фреймворк Django широко используется крупными компаниями в известных проектах, таких как Mozilla, Pinterest, YouTube, Instagram, Disqus, Google и др.

Также Django используется в качестве веб-компонентов в различных проектах, таких как Graphite – система построения графиков и наблюдения, FreeNAS – система хранения и обмена файлами и др.

Возможности веб-фреймворка Django:

— ORM, API для доступа к базе данных, поддерживающая транзакции;

— наличие встроенного интерфейса администратора, с встроенной локализацией на многие языки;

— интернационализация;

— диспетчер URL на основе регулярных выражений;

— расширяемая система шаблонов с тегами и наследованием;

— система кеширования;

— подключаемая архитектура приложений, которое можно установить на любое веб-приложение Django;

— система встроенных и настраиваемых фильтров “middleware” для построения дополнительных обработчиков запросов;

— шаблоны функций контроллеров “generic views”;

— аутентификация и авторизация через подключение внешних модулей: LDAP, OpenID и др;

— библиотека для работы с формами;

— встроенная автоматическая документация.

Изначально Django проектировался под управление сервером Apache, с использованием модуля mod python и с PostgreSQL в качестве базы данных.

После включения стандарта взаимодействия Python-программы с веб-сервером WSGI, Django может работать под управлением FastCGI, mod wsgi, или SCGI на Apache и других серверах, таких как lighttpd, nginx и сервера uWSGI [8].

В настоящие время Django поддерживает работу с такими СУБД как: MySQL, SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere и Oracle.

Django появился в 2005 году, и постепенно стал одним из лучших фреймворков. Django является чрезвычайно популярным и полнофункциональным серверным веб-фреймворком, написанным на Python.

1.6 Постановка задачи

Анализ предметной области показал, что “облачным” сервисом является не только высоконагружаемая система, но также и веб-приложение, которое позволяет пользователю загружать и хранить данные на удаленном сервере. Обеспечивая удобный доступ к собственным файлам пользователя в любое время, с любого устройства которое может подключаться к локальной сети, в которой находится сервер или в сети Интернет.

Данная глава целиком посвящена обзору различных архитектурных решений и программным средствам. На основе полученных данных, после ознакомления с теоретическими материалами, было сделано предположение, что для данной задачи, хранить данные в “облаке” – оптимальный вариант. Это обусловлено преимуществами “облачных” сервисов.

Специфика построений веб-приложений с помощью веб-фреймворка Django предполагает написание независимых модулей или приложений, которые можно легко добавить в существующее веб приложение. Это позволят в дальнейшем интегрировать дополнительные модули обработки и анализа хранимых данных [9].

Таким образом, в рамках данной работы, необходимо спроектировать и реализовать информационную инфраструктуру, учитывая особенности используемых программных средств и архитектурных решений. В результате чего развернуть на удаленном сервере веб-приложение для загрузки, обработки и хранения данных в “облачном” сервисе. После разработки необходимо провести тестирование спроектированного веб-приложения для выявления потенциально слабых сторон разработанной информационной инфраструктуры, с целью дальнейшего совершенствования конечного продукта и определения возможностей приложения.

Веб-приложение будет содержать модуль обработки изображений, который написан на языке Python. Это обусловливает выбор Python фреймворка для создания веб-приложения. Обеспечивая простую интеграцию модуля обработки в веб-приложение [10].

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОБЛАЧНОГО СЕРВИСА ДЛЯ ГЕМАТОЛОГИЧЕСКОГО АНАЛИЗА ИЗОБРАЖЕНИЙ КРОВИ

2.1 Проектирование инфраструктуры облачного сервиса

При проектировании информационных систем всегда стоит учитывать наличие определенных критериев в любой момент эксплуатации системы после ее запуска, а именно:

— пропускная способность системы;
— время отклика системы на запросы;
— отказоустойчивость системы;
— требуемая функциональность системы;
— безопасность данных;
— простота сопровождения и использование системы.

После определения возможных программных средств и архитектурных решений, на основе анализа выбраны технологии для достижения цели в данной дипломной работе. Теперь возникает задача разработки информационной структуры с учетом функциональных возможностей приложения, которое должна быть спроектировано.

В контексте задачи обработки и хранения данных на удаленном сервере “облачная” инфраструктура должна поддерживать:

— интуитивно понятный интерфейс для простоты использования и прозрачность доступа для пользователя;
— приемлемую скорость обработки хранящихся данных;
— возможность горизонтального масштабирования;
— безопасность хранимых данных.

Горизонтальная масштабируемость достигается по средствам увеличения производительных мощностей удаленного сервера на котором запущено веб-приложение, либо его перенос на более мощный удаленный сервер.

Возможности веб-фреймворка Django позволяют создать интуитивно понятный интерфейс для пользователя, а также, используя встроенные методы защиты приложения от взлома, обеспечить безопасность данных. Безопасность самого сервера ложится на провайдера, который предоставляет оборудование.

Каждое веб-приложение Django имеет собственный уникальный секретный ключ, который генерируется при создании проекта. Этот ключ используется для генерации так называемых токенов, которые являются сертификатом клиента для входа. Таким образом обеспечивается безопасность. С любым запросом клиента к серверу отправляется и токен, который был выдан для клиента. И при запросе веб-приложение проверяет, был ли он сгенерирован самим веб-приложением. Это является уникальным пропуском для каждого запроса.

2.2 Проектирование веб-приложения

Создаваемое приложение рассчитано на взаимодействие с пользователями и обработкой их данных, а, следовательно, можно выделить две основных составляющих:

— набор видов и контроллеров для взаимодействия с пользователем и моделями;

— логика для обработки хранимых данных.

Далее приводятся подпункты 2.2.1 и 2.2.2 в которых описан функционал приложения для работы с пользователями и проектирование базы данных соответственно.

2.2.1 Определение основного функционала

Основные функции, которые должно поддерживать веб-приложение:

— возможность регистрации новых пользователей;

— возможность пользователей редактирования изменения данных собственного аккаунта;

— загрузка изображений;

— просмотр коллекции собственных файлов, удаление и изменение;

— обработка выбранных файлов с сохранением результата обработки;

— просмотр детализации выполненной обработки.

Так как, когда речь идет о сервисе, который предполагает хранение данных пользователей встает вопрос о безопасности, а, следовательно, должна быть хорошо проработана и организованна следующая логика приложения:

— предотвращение попытки загрузки файла либо доступа к форме загрузки файла неавторизованным пользователям;

— преднамеренные или ошибочные попытки получения списка хранящихся файлов неавторизованных пользователей или списка, не принадлежащего текущему пользователю;

— предотвращение попытки завершения или открытия авторизованной сессии, если пользователь не был аутентифицирован;

— отработка корректной валидации некорректных данных в полях форм.

После корректной авторизации или регистрации должна создаваться активная сессия, которая позволяет текущему пользователю получить доступ к определенному функционалу приложения.

При попытках несанкционированного доступа или SQL-инъекций на стороне веб-сервера должны отработать сценарии, возвращающие пользователя на страницу авторизации.

Классы-валидаторы отвечают за обработку только тех данных, которые допустимы и могут хранится в базе данных.

После моделирования обработки нештатных ситуаций заданы особенности функционала, который теперь должен быть реализован. При правильном и грамотном подходе программирования веб-приложения возможно предоставить необходимый уровень надежности и безопасности хранения данных пользователей.

Завершающим этапом проектирования информационной инфраструктуры будет проектирование базы данных, это важная часть проектирования, которая конкретно отвечает за хранения данных.

2.2.2 Проектирование базы данных

Проектирование физического уровня базы данных информационной системы осуществляется на уровне СУБД. Особенности физической модели базы данных зависит от используемой СУБД. Физическая модель отвечает за представление объектов логической модели с помощью объектов конкретной базы данных. На рисунке 2.1 представлена физическая модель базы данных.

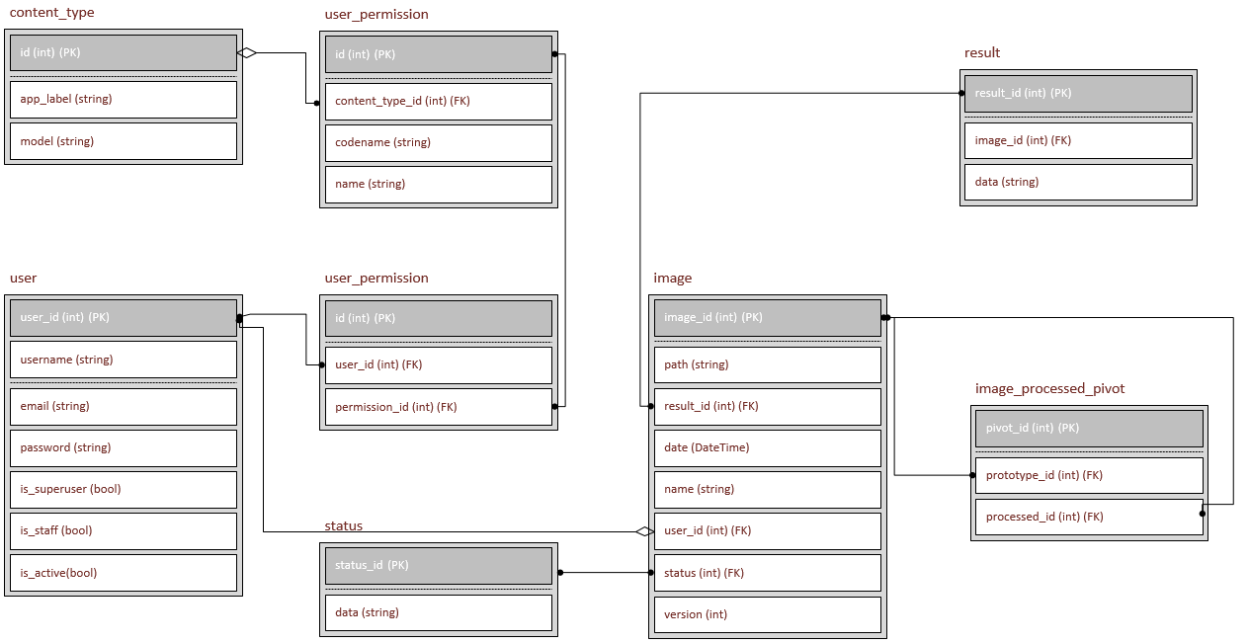

Рис. 2.1. Физическая модель базы данных

Логическая модель базы данных, в отличии от физической, строится в терминах информационных единиц, без привязки к конкретной СУБД. Логическая модель может быть необязательно выражена средствами реляционных моделей данных. Логическое проектирование заключается в определении числа сущностей и структур, которые выражены таблицами. Спецификацию логической модели базы данных определяет предметная область и ее особенности. Необходимо определить число сущностей, их свойства и связи между сущностями.

В структуре базы данных можно выделить три основных сущности: пользователь, изображение и результат обработки. В процессе приведения базы данных к третьей нормализованной форме можно выделить состояние изображения, которое определяет загружено изображение пользователем либо получено в результате обработки пользовательского изображения.

В модели присутствуют следующие сущности: пользователь, изображение (снимок крови с электронного микроскопа), “результат обработки изображения”.

Сущность “пользователь” отвечает за хранение данных учетной записи авторизованного пользователя: логин, электронный адрес, пароль и внешний ключ для связи с сущностью “изображение”. В таблице 2.1 приведено описание свойств сущности “пользователь”.

Таблица 2.1

Свойства сущности “пользователь”

Название	Тип	Ненулевое	Значение по умолчанию	Уникальное
user_id	Целочисленный. 4 байта	да	инкремент	да
user_login	Текстовый. 64 символа в длину	да	нет	да
user_email	Текстовый. 64 символа в длину	да	нет	да
user_password	Текстовый. 64 символа в длину	да	нет	нет

Свойство user_id служит идентификатором записи пользователя, которое генерируется автоматически при создании записи и является уникальным. Оно является первичным ключом, а также служит для связи с другими сущностями.

Свойство user_login является уникальным именем пользователя для его идентификации в приложении. Пользователь задает его сам при регистрации. Это поле служит для авторизации пользователя в приложении.

Свойство user_email является уникальным полем для адреса электронной почты пользователя. Данные этого поля используются для восстановления учетной записи пользователя в приложении. А также может быть использовано для рассылки оповещений самим приложением.

Свойство user_password служит для хранения пароля от учетной записи пользователя. Пользователь задает это поле при регистрации, а в дальнейшим его информация служит для авторизации.

Сущность “изображение” отвечает за хранение данных снимков крови внутри сервиса: идентификатор изображения, идентификатор пользователя, версия изображения, состояние изображения, дата создания изображения, имя изображения и идентификатор прототипа изображения. В таблице 2.2 приведены свойства сущности “изображение”.

Таблица 2.2

Свойства сущности “изображение”

Название	**Тип**	**Ненулевое**	**Значение по умолчанию**	**Уникальное**
image_id	Целочисленный. 4 байта	да	нет	да
image_userID	Текстовый. 64 символа в длину	да	нет	нет
image_path	Символьный. 1024 символа в длину	да	нет	да
image_version	Текстовый. 64 символа в длину	да	нет	нет

Таблица 2.2 (продолжение)

Название	Тип	Ненулевое	Значение по умолчанию	Уникальное
image_stat	Целочисленный. 4 байта	да	нет	нет
image_date	Дата\Время	да	текущее время и дата	нет
image_name	Текстовый. 64 символа в длину	нет	“image” + id	нет
image_prevImgID	Целочисленный. 4 байта	нет	нет	нет

Свойство image_id служит идентификатором записи сущности “изображение”, которое генерируется автоматически при создании записи и является уникальным. Оно является первичным ключом, а также служит для связи с другими сущностями.

Свойство image_userID является внешним ключом, определяет принадлежность записи об изображении к пользователю, который загрузил его или сгенерировал.

Свойство image_path хранит путь к файлу изображения в файловой системе сервера на котором оно хранится.

Свойство image_version хранит в себе данные о том, какая именно обротка была изображения была произведена, так как одно и тоже изображение можно обработать несколько раз, получив одинаковое результирующее изображение, но его разные версии.

Свойство image_stat служит для определения, это изображение загружено пользователем или оно сгенерировано алгоритмом в результате обработки алгоритмом.

Свойство image_date показывает дату создания изображения, эти данные добавляются автоматически на уровне СУБД создании записи в базе.

Свойство image_name служит способом пользователя унифицировать для себя загруженные изображения. Пользователь всегда может изменить значение этого поля на произвольное.

Свойство image_prevImgID показывает от какого изображения было сгенерировано текущее, в процессе обработки. Это поле является внешним ключом для рекурсивной связи с сущностью “изображение”.

Сущность “результат обработки изображения” хранит метаданные об обработке изображения крови. Каждый экземпляр этой сущности связан с экземпляром сущности “изображение”. В таблице 2.3 приведено описание сущности “результат обработки изображения”.

Таблица 2.3

Свойства сущности “результат обработки изображения”

Название	**Тип**	**Ненулевое**	**Значение по умолчанию**	**Уникальное**
result_id	Целочисленный. 4 байта	да	инкремент	да
result_imageID	Целочисленный. 4 байта	да	нет	да
result_data	Символьный. в длину	да	нет	нет

Свойство result_id служит идентификатором записи сущности “результат обработки изображения”, которое генерируется автоматически при создании записи и является уникальным. Оно является первичным ключом, а также служит для связи с другими сущностями.

Свойство result_imageID является внешним ключом, определяет принадлежность записи об результате обработки к изображению.

Свойство result_data хранит метаданные обработки изображения, которые используются для анализа.

На этом этап можно считать даталогическое проектирование базы данных для “облачного” сервиса для анализа изображений крови оконченным и можно приступить к этапу реализации и программирования.

2.3 Реализация спроектированного приложения

Для реализации спроектированного приложения будет использован набор компонентов веб-фреймворка, для быстрой и качественной реализации веб-приложения.

При разработке веб-приложений, часто встают похожие задачи: аутентификация пользователь (регистрация, вход, выход), загрузка файлов, логика форм, панель управления и администрирования веб-приложения и т. д. Этот, также, как и другие фреймворки, существует для того чтобы облегчить процесс разработки. В своей комплектации они уже имеют наборы классов и методов, позволяющие быстро решить задачи, которые чаще всего встречаются при разработке веб-приложений.

2.3.1 Программирование моделей

Если рассматривать приложение, как многоуровневую структуру, то модель находится в основе этой архитектуры. Модель – часть MVC, которая практически всегда остается неизменной.

Каждая модель приложения представляет собой Python классом, который наследуется от родительского класса django.db.models.Model. Родительский класс имеет весь необходимый функционал для обеспечения взаимодействия объектов классов моделей с базой данных. В результате чего, Django предоставляет понятный и простой способ представления сущностей и их свойств в базе данных по средствам моделей.

Каждая модель описывает одну соответствующую таблицу в базе данных, а каждое свойство класса модели – соответствующее поле в этой таблице.

Используя преимущества реляционных баз данных Django позволяет задавать различные отношения между сущностями:

— отношение “многие ко многим”;

— отношение “многие к одному”;

— отношение “один к одному”.

Для определения отношения “многие к одному” используется внешний ключ ForeignKey. Он реализуется по средствам соответствующего атрибута для поля в модели.

При создании такого поля, методу ForeignKey следует определить параметр, который будет указывать, на какую модель будет ссылаться это поле, метод автоматически ссылается на идентификатор указанной модели.

Такая связь будет указывать на то, что какой-то экземпляр одного класса может ссылаться на многие экземпляры другого. Как, например, один пользователь может загружать много файлов.

Для определения отношения “многие ко многим” также используется атрибут, только он имеет тип ManyToManyField. Он используется также, как и остальные методы. Важным нюансом использования является то, что этот атрибут должен находится лишь в одной из двух связанных моделях.

Этот тип связи указывает на то, что один экземпляр класса может, как и ссылаться на многие экземпляры другого класса, так и принадлежать многим экземплярам этого же класса.

Для определения отношения “один к одному” также используется атрибут, типа OneToOneField. Связь такого типа используется при необходимости расширения или дополнения класса объекта. Например, когда данные для аутентификации пользователя: логин, пароль и email хранится в одной таблице, а персональные данные – в другой, но только один пользователь может иметь только один пароль [11].

В приложении присутствуют три модели, которые необходимо реализовать: пользователь, изображение и результат обработки.

Для создания системы аутентификации фреймворк предоставляет функционал, который уже содержит: аккаунты, группы, права и сессии.

Итак, при сравнительно простом программировании имеется:

— пользователи;

— роли, определяющие права пользователей на выполнение определенных действий;

— разделение пользователей на группы;

— хранение и хэширование паролей.

Так как модель пользователя является вшитой в фреймворк, а для данной концепции приложения не используется возможность расширения модели пользователя, стандартная модель полностью удовлетворяет требуемый функционал, то листинге 2.1 приведен код модели “изображение”.

Листинг 2.1. Программный код модели Image

```
class Image(models.Model):
    image_userID = models.ForeignKey(User,
on_delete=models.CASCADE)
    image_path = models.FileField()
    image_version = models.IntegerField(default=0,
null=True)
    image_date =
models.DateTimeField(auto_now_add=True)
    image_state = models.IntegerField(default=0,
null=True)
    image_name = models.CharField(max_length=250,
default='image')
    image_prevImgID = models.IntegerField(default=None,
null=True)
```

В листинге 2.1 видно, что данная модель связанна отношением “один ко многим” с моделью пользователя. Оно связанно атрибутом внешнего ключа, который ссылается на модель User. Также нужно обратить внимание на свойство image_path, которое имеет специальный метод в атрибуте, который описывает, что это поле будет хранить путь загружаемого файла FileField. Вместо того, чтобы задавать тип этого поля текстовым, используется специальный тип, который указывает формам, что для данного поля необходимо добавить элемент для загрузки файла на форму.

Также используется свойство, которое будет заполнять поля даты в тот же момент, когда была сохранена запись.

Из моделей, при миграции базы данных, генерируется SQL код, который выполняется СУБД.

Реализацию модели, хранящую результат обработки изображений можно просмотреть в дополнительной документации дипломной работы. Программный код модели Result приведен в приложении.

2.3.2 Задание маршрутов

Работа веб-приложения организованна следующим образом: клиент проходит по определенному адресу, формируя и отправляю запрос серверу, который в свою очередь совершает необходимые, формирует ответ клиенту и отправляет его.

Веб-фреймворк Django предлагает явную настройку этих маршрутов, в формате “адрес – действие по этому адресу”. Можно явно указывать по какому адресу необходимо пройти и какой контроллер отработает при переходе по этому маршруту.

Для этого имеются так называемые urlpatterns, которые хранят в себе список маршрутов для определенного приложения. В листинге 2.2 приведен список маршрутов для логики работы с изображениями.

Листинг 2.2. Файл маршрутов i2d/urls.py

```
urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^add-image/$', views.CreateImage.as_view(),
name='add_image'),
    url(r'^images/$', views.ShowImages.as_view(),
name='images'),
    url(r'^(?P<pk>[0-9]+)/$', views.ShowImage.as_view(),
name='detail'),
    url(r'image/(?P<pk>[0-9]+)/$', views.UpdateImage.as_view(),
name='update_image'),
    url(r'image/(?P<pk>[0-9]+)/delete/$',
views.DeleteImage.as_view(), name='delete_image'),
    url(r'image/(?P<pk>[0-9]+)/processing/$', views.processing,
name='processing'),
]
```

Таким образом приложение понимает, как правильно реагировать на запросы клиента и какой контроллер задействовать, чтобы сгенерировать верный ответ от сервера.

Для записи выражения адреса используются регулярные выражения, это позволяет создавать динамические маршруты, в пути которых используются идентификаторы объектов базы данных.

Например, необходимо просмотреть список всех изображений, которые загрузил определенный пользователь, ответ на этот запрос можно получить, пройдя по адресу i2d/, но если необходимо посмотреть детальную информацию о каждом объекте из списка, то нужно сообщить серверу, какой конкретно объект клиент хочет получить. На этот случай в листинге 2.2 приведена инструкция маршрута, которая будет в себе содержать идентификатор нужного объекта, например, i2d/5. Таким образом сервер поймет, что ему необходимо сформировать отчет об объекте с индексом 5 и отправить клиенту.

Подробный листинг всех маршрутов содержится в приложении.

2.3.3 Программирование шаблонов

Интерфейс веб-приложения представляют виды – HTML страницы, которые отправляются клиенту. Так как один из основных принципов фреймворка является DRY (don’t repeat yourself), то в Django имеется встроенный шаблонизатор, который позволяет сократить написание HTML кода, который будет развернут сервером при отправке клиенту в полноценный HTML. Также это позволяет едина разово написать нужный участок кода в отдельный файл и если он часто повторяется, то просто подключить его к нужному шаблону, вместо многоразового копирования кода. Это очень удобно как с точки зрения написания видов, так и их поддержание. Если придется поменять какой-то участок кода, отвечающий за основное меню сайта, которое может находиться на каждой страничке приложения, то достаточно внести необходимые изменения всего в один файл.

Стандартно имеется базовый шаблон, в котором подключены все необходимые CSS и JavaScript файлы, в нем описывают базовую структуру страницы, а разметку самого контента подключают, как его расширение. Базовый шаблон обычно имеет название base.html и располагается в директории templates. Таким образом, благодаря шаблонизатору, имеется очень эффективный способ создания представлений.

Также есть возможность включать в код шаблона различные циклы, типа for и foreach, логические условия. Эти возможности позволят существенно сократить код.

В листинге 2.3 приведен код разметки, наследующейся от базового шаблона. Это шаблон страницы авторизации пользователей, который внутри базового скелета страницы, содержит еще и форму, в которую пользователю необходимо ввести логин и пароль своего аккаунта.

Листинг 2.3. Код разметки форы входа

```
{% extends 'account/../base.html' %}
{% block body %}
    <br/>
    <div class="container">
        <div class="row">
            <div class="col-sm-6 offset-sm-1">
                <form method="post" class="justify-content-
center">
                    {% csrf_token %}
                    {% include 'account/base_form.html' with
form=form %}
                    <button type="submit" class="btn btn-
default">
                        Войти
                    </button>
                    <a href="{% url 'account:reset_password'
%}">
                        <button type="button" class="btn btn-
link">Забыл пароль</button>
                    </a>
                </form>
            </div>
        </div>
    </div>
{% endblock %}
```

В общей сложности в приложении реализовано 18 различных шаблонов. Исходный код всех шаблонов можно просмотреть в приложении.

2.3.4 Программирование форм

В Django имеется встроенный инструмент, для решения задач, связанных с формами. Это касается как широкого спектра возможностей, от валидации полей до рендера форм. По большей мере, за весь этот функционал отвечает модуль django.forms.

Основным преимуществом использование этой библиотеки является создание отдельного класса Form, для отображения каждой формы в шаблоне. Программирование данного функционала обычно происходит в специальном файле forms.py.

Благодаря валидаторам, объекты классов, наследуемые от Form могут проверять правильность введенных данных. Предопределенный или встроенный валидатор проверяет поля методом is_valid() методом is_bound(), который проверяет непустые поля. Метод is_valid возвращает логическое значение, которое использует контроллер для дальнейших действий, в случае, сели поля формы не пройдут валидацию, то сервер вновь вернет туже самую форму, но с информацией о том, какие поля были заполнены неверно.

Благодаря возможности автоматически рендерить формы в разметку, можно едино разово создать шаблон формы, который с помощью циклов будет рендерить нужное количество полей определенного типа для каждой формы, можно существенно сократить время разработки. Особенно это будет ощутимо на больших проектах.

В приложении представлен базовый шаблон формы base_form.html.

В листинге 2.4 представлен код программирования формы для редактирования пользовательских данных.

Листинг 2.4. EditProfileForm

```
class EditProfileForm(UserChangeForm):

    def __init__(self, *args, **kwargs):
        super(EditProfileForm, self).__init__(*args, **kwargs)
        self.fields['email'].label = "Email"
        self.fields['first_name'].label = "Имя"
        self.fields['last_name'].label = "Фамилия"
        self.fields.pop('password')

    class Meta:
        model = User
        fields = ('email', 'first_name', 'last_name')
```

Из вышеприведенного листинга видно, что пользовательский класс EditProfileForm наследуется от базового класса UserChangeForm, который служит для изменения данных пользователей. А также, что класс EditProfileForm имеет вложенный мета класс Meta, который определяет какую модель использовать этой форме, для того чтобы рендерить поля. Свойство мета класса fields определяет для каких свойств класса модели необходимо рендерить поля. А внутри самого класса EditProfileForm можно переопределить лейблы для полей формы, которые будут отображены. Внутри класса не нужно указывать типы полей, так как информация об этом уже содержится в модели.

2.3.5 Программирование контроллеров

В архитектуре Django нет четкого разделения на виды и контроллеры. Логика обработки запросов “размазана” между файлами маршрутов urls.py и views.py. Так что контроллером можно считать views.py.

Поэтому в данном контексте контроллеры называются видами. Разделяют два типа видов: виды, основанные на классах – class based views и виды, основанные на функциях – function based views. При создании этого приложения для программирования логики работы приложения с пользователем были использованы function based views, а для реализации логики работы, связанной с изображениями – class based views.

Это обусловлено тем, что логика работы с пользователем уже частично реализована в стандартном пакете Django, и можно легко добавить необходимый функционал с помощью функций.

А работа с изображениями требует полного написания от программиста, присутствует только общая концепция, которая описана в базовых класс, имеет смыл создание пользовательских классов, которые будут наследоваться от базовых.

2.3.6 Программирование middleware

Иногда возникает необходимость написания дополнительной логики, связанный, например, определение функционала, который доступен пользователю с определенной ролью, правами. Так как поведение веб-сервера распределено между файлами маршрутов и так называемыми файлами видов, но для написания подобной логики существуют специальные файлы middleware, так как писать это в вышеперечисленные файлы не целесообразно.

В данном приложении присутствует специальный middleware, который не позволяет не авторизованным пользователям попадать на внутренние страницы приложения, даже если они знают точный адрес этой странички. Этот специальный обработчик будет перехватывать запрос и возвращать

форму регистрации, если пользователь, отправивший этот запрос не был авторизован.

Для этого имеется массив адресов страничек, которые неавторизованный пользователь может посещать. При получении сервером запроса, если указанный адрес не содержится в этом массиве, а отправивший его пользователь не был авторизован, то сервер, в качестве ответа присылает форму для входа или регистрации пользователя.

Массив доступных адресов приложения хранится в файле settings.py, представлен в листинге 2.4.

Листинг 2.4. массив доступных адресов приложения

```
LOGIN_EXEMPT_URLS = (
    r'^account/login/$',
    r'^account/logout/$',
    r'^account/registration/$',
    r'^account/reset-password/$',
    r'^reset-password/done/$',
    r'^reset-password/confirm/(?P<uidb64>[0-9A-Za-z]+)-
(?P<token>.+)/$',
    r'^reset-password/complete/$',
)
```

При каждом запросе проверяется это условие, код данной логики представлен в листинге 2.5.

Листинг 2.5 Обработка логики middleware

```
class LoginRequiredMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        return response
    def process_view(self, request, view_func, view_args,
view_kwargs):
        assert hasattr(request, 'user')
        path = request.path_info.lstrip('/')
```

```
        url_is_exempt = any(url.match(path) for url in
EXEMPT_URLS)
        if request.user.is_authenticated and url_is_exempt:
            return redirect(settings.LOGIN_REDIRECT_URL)
        elif request.user.is_authenticated or url_is_exempt:
            return None
        else:
            return redirect(settings.LOGIN_URL)
```

Наличие подобного механизма позволят быстро и удобно вносить новую логику в работу приложения, не нарушая структуры уже написанных файлов.

2.3.7 Создание панели администрирования

Панель администрирования является удобным инструментом для управления приложением. При её наличии это может делать даже человек, который не имеет никакого понятия о программировании веб-приложений. Используемый веб-фреймворк имеет встроенные механизмы быстрого создания панели администратора.

Доступ к этой панели имеют только пользователи с уровнем доступа superuser. Для создание такого пользователя необходимо, находясь в папке проекта запустить консоль, и ввести команду createsuperuser. Далее будет предложена стандартная форма регистрации, но в консоли. Этот пользователь также будет иметь доступ и к приложению, а также иметь возможность менять права доступа другим пользователям.

Панель администратора представляет из себя меню, в котором можно создавать, изменять и удалять любые объекты, которые были ранее описаны моделями. Чтобы в панели были доступны манипуляции с новыми объектами моделей, необходимо зарегистрировать файлы моделей в файле admin.py.

В конечном итоге, веб-приложение состоит из трех приложений, так называемых модулей (см. рис. 2.3).

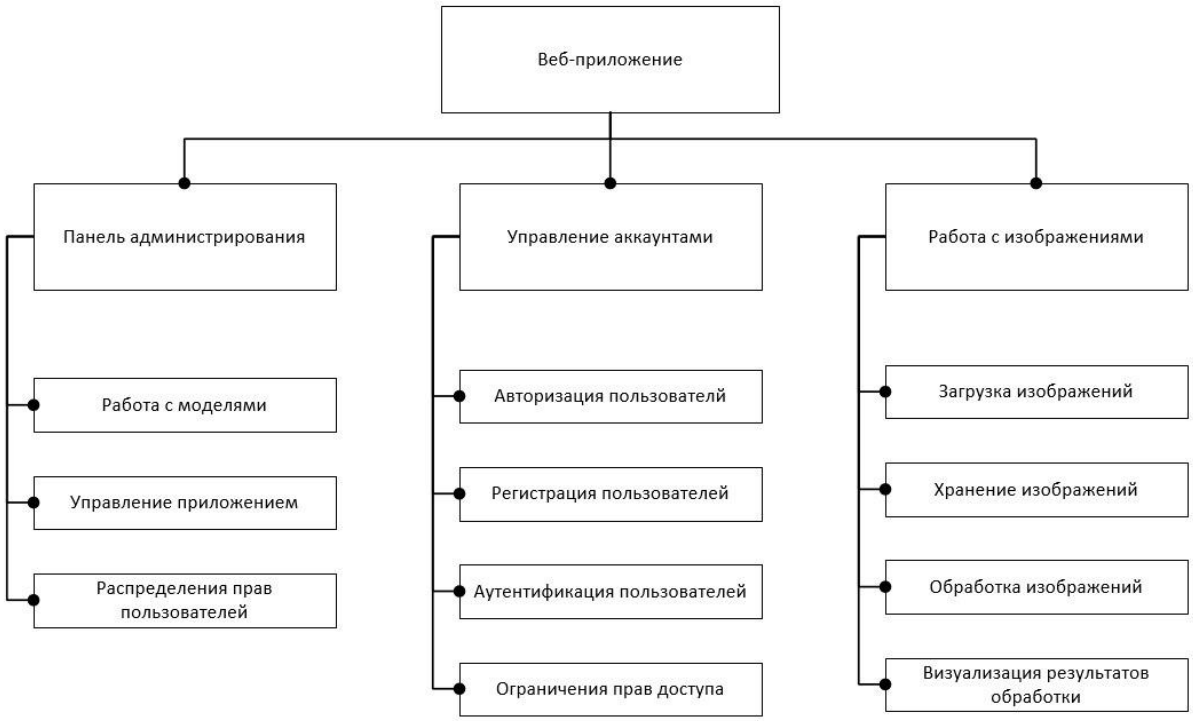

Рис. 2.3. Модули веб-приложения

Приложение admin panel отвечает за панель администрирования веб-приложения.

Приложение account отвечает за работу приложения с пользователями, система учета и аутентификации.

Приложение i2d (two dimensional images) отвечает учет, хранение и обработку приложений.

ГЛАВА 3. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ВЕБ-ПРИЛОЖЕНИЯ

После окончания разработки программы следует период её эксплуатации с целью выявления дефектов, этот процесс называется тестирование.

Для обнаружения определенных дефектов существуют свои тестовые работы, определенные методики тестирования. Для определения этих методов необходимо определится с общей стратегией тестирования.

Существуют несколько подходов к формулированию стратегии тестирования. Одни из них – тестирование требований с высоким приоритетом и тестирование конечной конфигурации продукта.

3.1 Определение необходимых тестовых условий

Так как одним из важнейших критериев, при работе с “облаком”, является безопасность, то особую часть в тестировании необходимо уделить для выявления дефектов безопасности.

Помимо этого, следует уделить внимание загрузке и хранению пользовательских данных.

Исходя их вышеперечисленного видно, какие требования нужно подвергать тестированию и можно выделить основные тест-кейсы:

— хранение сессий;
— разделение прав пользователей;
— аутентификация пользователей;
— переадресация;
— защита от SQL-инъекций;
— целостность загружаемых данных;
— целостность хранимых данных.

Чтобы каждый раз пользователю не приходилось вводить свои данные для аутентификации используются сессии, которые проверяются каждый раз, когда пользователь возвращается в приложение. Так как сессии хранятся на стороне клиента, то они должны хранится в зашифрованном виде, чтобы никто не мог использовать эти данные. Обычно данные сессии хранятся в виде кэшированных данных, таким образом, только сервер приложения сможет проверить их подлинность.

В разработанном приложении присутствует панель администрирования, к которой имеют доступ ограниченное число пользователей, по умолчанию – только один пользователь, который создается в процессе разработки. Этот пользователь, позже, сможет раздать особые права и остальным пользователям, используя эту самую панель.

Для того чтобы пользоваться функциями разработанного “облачного” сервиса пользователь должен авторизоваться или зарегистрироваться в системе. Изначально гостю доступны только страницы авторизации и регистрации.

Также нужно исключить возможность отправки несанкционированных запросов к серверу неавторизованных пользователей или пользователей без специальных прав доступа. SQL-инъекция – один из самых доступных способов взлома веб-приложений. Суть таких инъекций – внедрение в данные (передаваемые через GET, POST запросы или значения Cookie) произвольного SQL кода.

При загрузке данных на сервере необходимо убедится, что файлы действительно находятся в файловой системе удаленного сервера и соответствующая запись в базе данных корректно возвращает путь к файлу для его отображения и работы с ним. А при удалении файла должно происходить удаление файла из файловой системы, а не только записи из базы данных.

3.2 Тестирование системы аутентификации и хранения сессий

Цель данного теста определить исправность работы системы аутентификацию пользователей в веб-приложении. В данном кейсе будет протестированы:

— регистрация пользователей;

— авторизация пользователей;

— восстановления пароля пользователя;

— хранение сессий;

— допуск пользователей к определенному функционалу, в зависимости от их прав доступа.

При переходе по адресу расположения веб приложения неавторизованный пользователь попадает на страницу с формой для входа в систему (см. рис. 3.1).

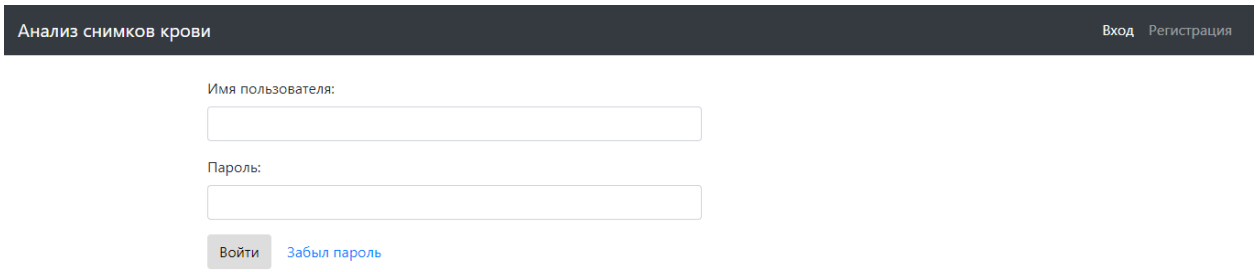

Рис. 3.1. Форма для входа в систему

Как на стороне клиента, так и на стороне сервера установлена валидация полей. Оба поля являются обязательными и поэтому отправка формы не произойдет благодаря валидации HTML5. Но валидация обязательности полей также присутствует и на стороне сервера, если каким-то образом получится отправить запрос минуя форму. Далее контроллер проверяет подлинность

данных, сравнивая с теми, что хранятся в базе данных. Если проверка пройдена успешно, то создается сессия пользователя и его пускает к функционалу приложения.

Если пользователь не имеет аккаунта в приложении, то он может со стартовой станицы перейти на форму регистрации (см. рис. 3.2).

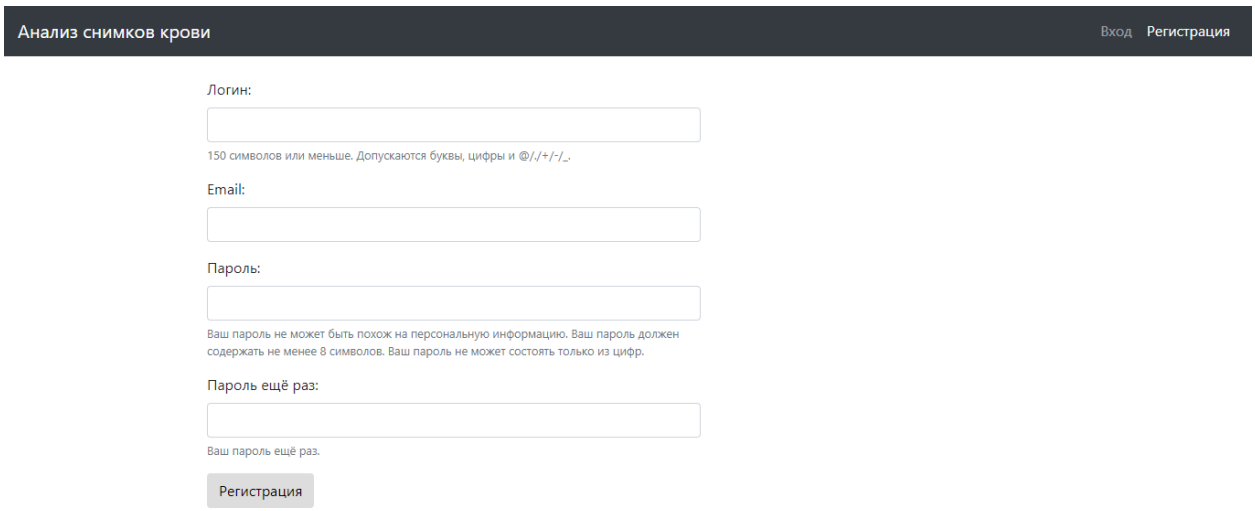

Рис. 3.2. Форма регистрации пользователя

Так эта данные с этой форму будут сохранены в базу данных приложения, то для полей формы регистрации задано больше параметров валидации. Информация о том, какой формат дынных ожидает каждое их полей, указанно под полями, в качестве подсказки, для удобства пользователя.

Если все данные введены пользователем верно, то происходит их добавление в базу данных, а также создается сессия пользователя, который только что был зарегистрирован. И он автоматически входит в систему.

Сессия сохраняется в файлах cookie и предоставляет возможность вернутся в систему пользователю, если пользователь закрыл вкладку веб-приложения, но не завершал сеанс работы, нажав кнопку выход. Данные для его авторизации хранятся в памяти браузера в качестве хэш-функции (см. рис. 3.3).

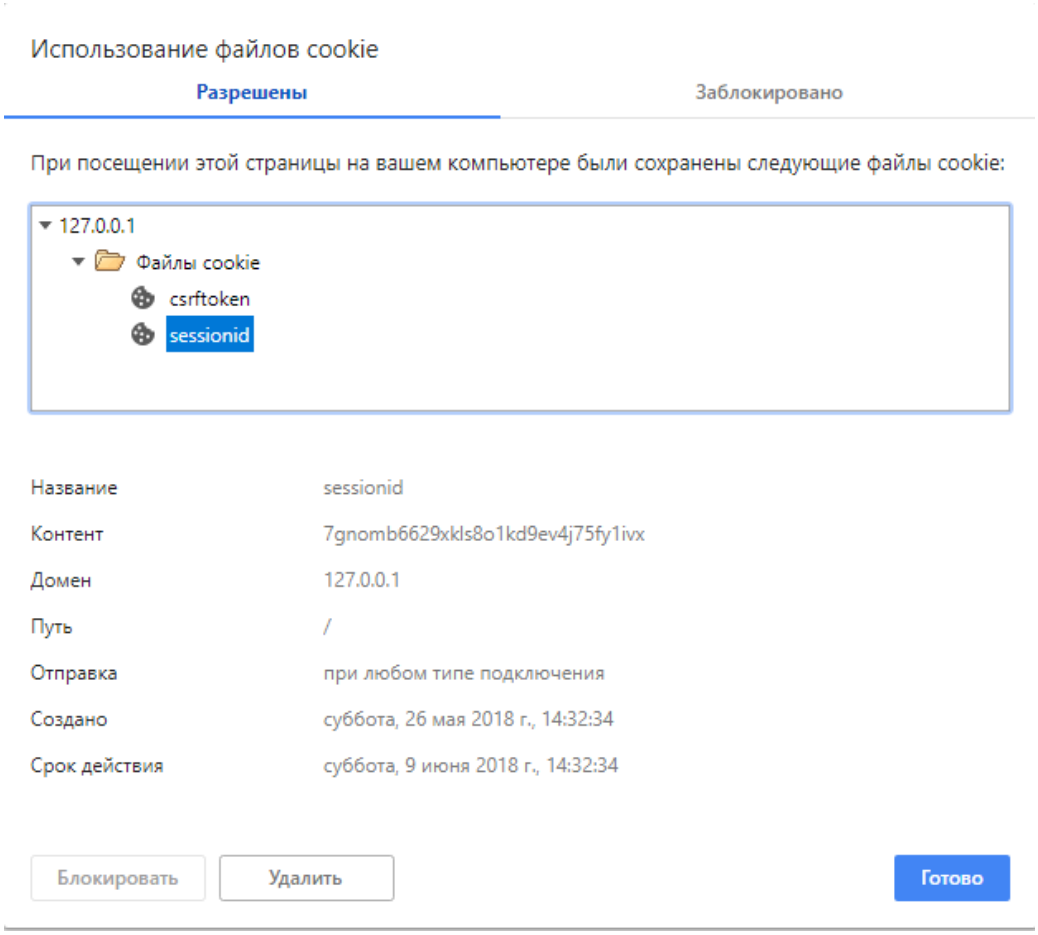

Рис. 3.3. Файлы cookie

На рисунке 3.3 видно, что файлы cookie, созданные приложением, имеют срок действия в 14 дней, если не обновлять сессию, то по истечению 14 дней пользователю нужно будет еще раз вводить данные для авторизации.

Кроме сессии, в файлах cookie хранится csrftoken, который подтверждает подлинность запросов клиента к серверу.

Если пользователь забыл свой пароль, то есть возможность восстановить его, используя адрес электронной почты, к которому привязан аккаунт. Пользователю предлагается ввести адрес электронной почты и если есть аккаунт, связанный с введенной почтой, то на нее высылается ссылка, которая ведет на форму восстановления пароля.

3.3 Тестирование загрузки, хранения и обработки изображений

После аутентификации пользователь попадает в рабочую область, где отображаются все загруженные изображения (см. рис. 3.4).

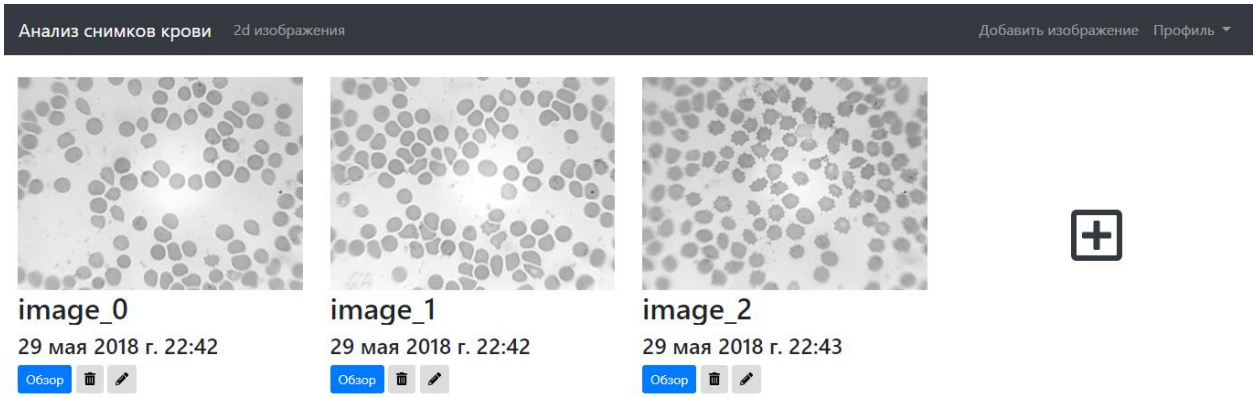

Рис. 3.4. Рабочая область. Галерея загруженных изображений

При загрузке нового изображения на сервере сохраняется файл и соответствующая запись в базе данных. Этот функционал работает верно и все корректно сохраняется. Также каждое изображение доступно для удаления и редактирования.

При нажатии кнопки “Обзор”, каждое изображение доступно для более детального просмотра. На рисунке 3.4 слева расположен снимок крови с микроскопа, а справа обработанное изображение. Контроллер возвращает файл изображения и сохраняет его в файловой системе сервера, соответствующая запись добавляется в базу данных. Помимо изображения сохраняются мета данные, которые хранят процентное соотношение эритроцитов в соответствии с их диаметром.

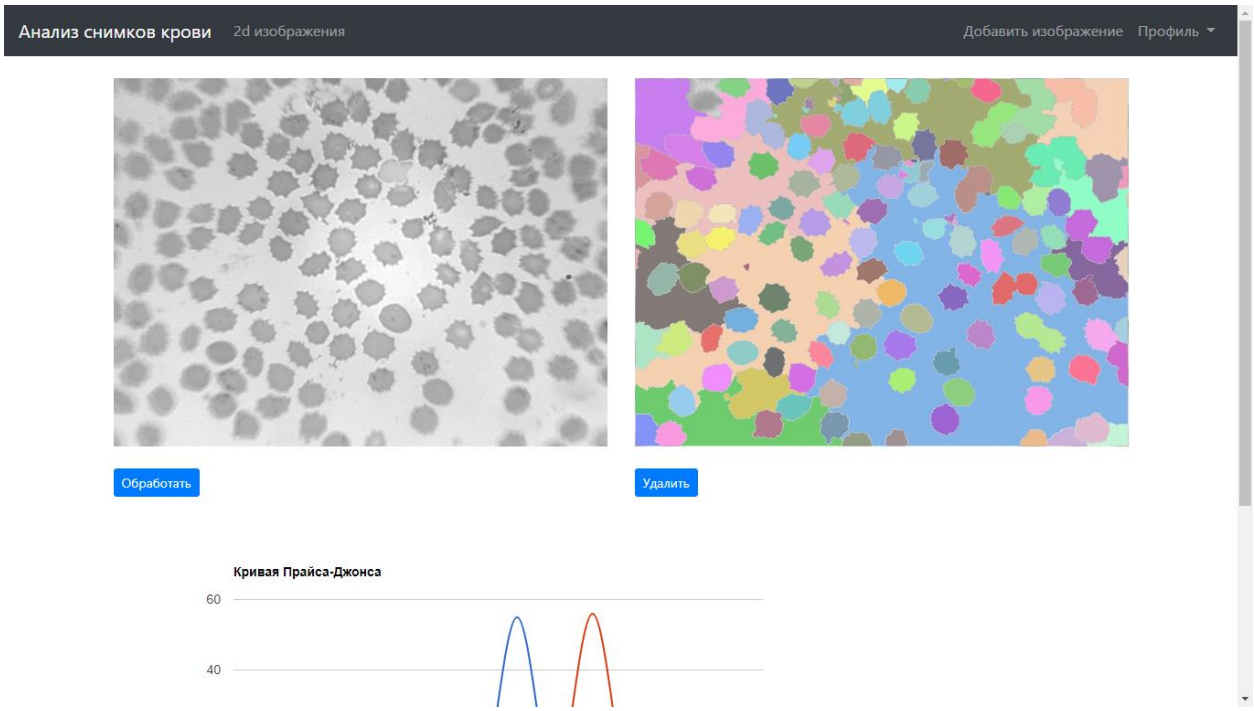

Рис. 3.4. Обработка изображения

На рисунке 3.5 изображена визуализация метаданных обработанного изображения, по средствам графика.

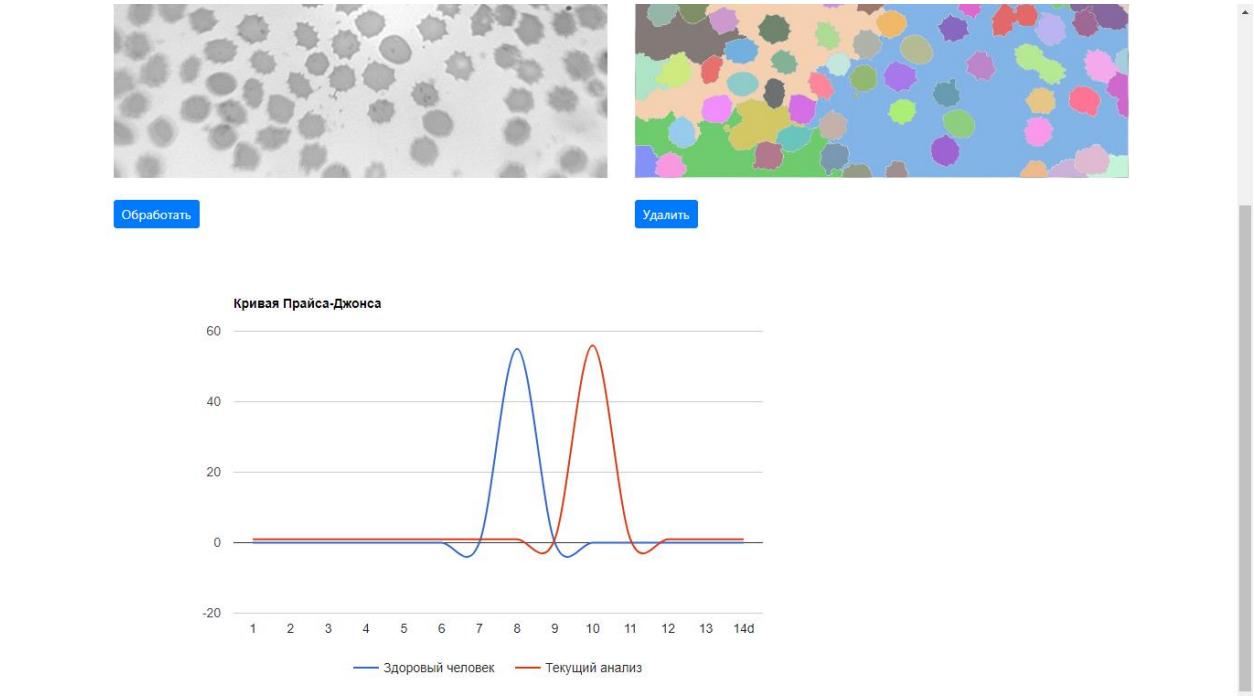

Рис. 3.5. Визуализация метаданных снимка крови

С помощью API Google Charts визуализирована кривая Прайса-Джонса, которая показывает соотношения содержания в крови эритроцитов с различными диаметрами, где по оси абсцисс откладывают величину диаметра эритроцитов (в мкм), а по оси ординат — процентное содержание эритроцитов соответствующей величины.

Синим графиком отображено эталонное значение здорового человека, а красным – показатели текущего анализа. Эти данные получены в результате обработки изображений.

Результаты теста показали, что все функции приложения работаю исправно.

ЗАКЛЮЧЕНИЕ

В ходе работы описаны теоретические сведенья об сильных и слабых сторонах облачных сервисов, а также о построении веб-приложений с использованием веб-фреймворка Django.

В рамках дипломной работы, поставленная цель достигнута, разработка и реализация информационной инфраструктуры “облачного” сервиса для хранения и обработки медицинских данных, а также реализация веб-интерфейса веб-приложения.

Результатом работы является разработанное веб-приложение, которое решает следующие поставленные задачи:

— хранение изображений;

— помощь в анализе изображений;

— унификация способа доступа к инструменту анализа;

— автоматизация процесса анализа.

Разработанное приложение протестировано с учетом вопросов безопасности данных пользователей. И является готовым программным продуктом, пригодным для использования в качестве инструмента для анализа медицинских изображений крови.

Таким образом, поставленную цель можно считать достигнутой.

В качестве возможных путей дальнейшего развития проекта и дальнейших исследований можно предложить:

— масштабирование и оптимизация приложения для работы с большими объемами данных;

— использование распределенных баз данных для надежности хранения пользовательских данных;

— интеграция дополнительных новых модулей, для обработки медицинских данных.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Свободная энциклопедия Википедия, статья “Облачное хранилище данных”. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%87%D0%BD%D0%BE%D0%B5_%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85 (дата обращения: 08.04.2018)
2. Свободная энциклопедия Википедия, статья “Веб-приложение”. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5 (дата обращения: 14.04.2018)
3. Свободная энциклопедия Википедия, статья “Масштабируемость”. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C (дата обращения: 14.04.2018)
4. Масштабирование нагрузки веб-приложений. [Электронный ресурс]. Режим доступа: https://habr.com/post/113992/ (Дата обращения: 20.04.2018)
5. 7 этапов построения масштабируемых веб-приложений. Стратегии для системных архитекторов. [Электронный ресурс]. Режим доступа: http://webcrunch.ru/library/development/highload-basics/seven-steps-of-scaling/ (дата обращения: 22.03.2014)
6. Архитектура высоконагруженных систем. [Электронный ресурс]. Режим доступа: http://www.insight-it.ru/highload/ (дата обращения: 23.03.2014)

7. Свободная энциклопедия Википедия, статья “ORM”. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/ORM (дата обращения: 14.04.2018)
8. Django documentation | Django documentation | Django. [Электронный ресурс]. Режим доступа: https://docs.djangoproject.com/en/dev/ (дата обращения 20.04.2018)
9. Выгоняйло В.Р., Батищев Д.С., Михелев В.М. «Микросервисная архитектура сервиса персональной медицины анализа крови» Информатика: проблемы, методология, технологии материалы XVIII Международной научно-методической конференции, Воронеж 9-10 февраля 2018, в 7т. – Воронеж: Изд-во ООО «Вэлборн», 2018. Т. 5. 21 – 25 с.
10. Выгоняйло В.Р., Батищев Д.С., Михелев В.М. «Облачный вычислительный сервис гематологического анализа на основе медицинских изображений» Информатика: проблемы, методология, технологии сборник материалов XVII международной научно-методической конференции, Воронеж 9-10 февраля 2017, в 5т. – Воронеж: Изд-во ООО «Вэлборн», 2017. Т. 4. 187 – 190 с.
11. Первая страница: DjangoBook по-русски | Django на русском. [Электронный ресурс]. Режим доступа: http://djbook.ru/index.html (дата обращения 1.04.2014)

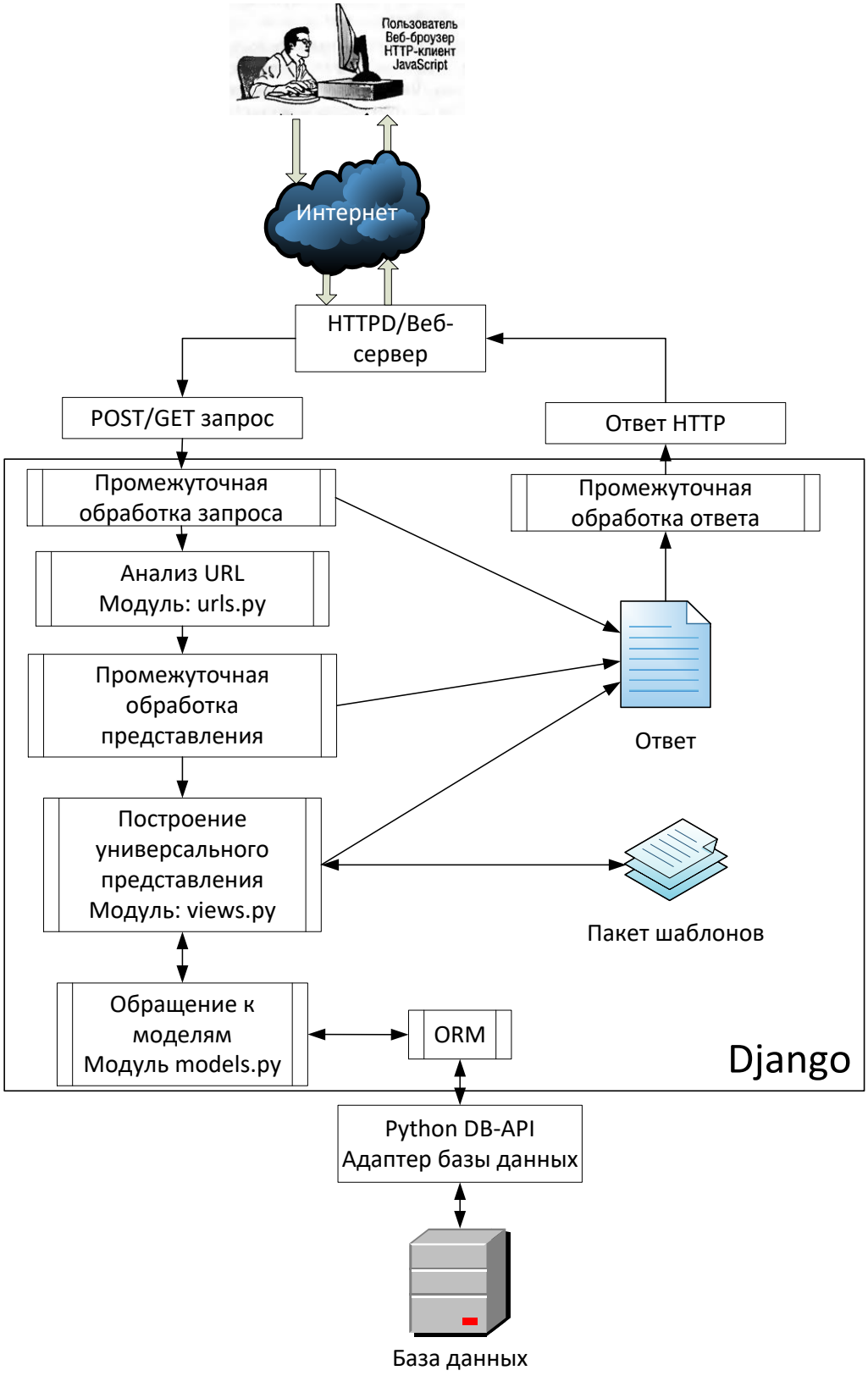

Рис. 1.1. Механизм обработки HTTP-запроса на платформе Django

ПРИЛОЖЕНИЕ 2

Листинг 2.1. Программный код моделей веб-приложения. Файл models.py

```
from django.db import models
from django.contrib.auth.models import User


class Image(models.Model):
    image_userID = models.ForeignKey(User,
on_delete=models.CASCADE)
    image_path = models.FileField()
    image_version = models.IntegerField(default=0, null=True)
    image_date = models.DateTimeField(auto_now_add=True)
    image_state = models.IntegerField(default=0, null=True)
    image_name = models.CharField(max_length=250,
default='image')
    image_prevImgID = models.IntegerField(default=None,
null=True)


class Result(models.Model):
    result_imageID = models.ForeignKey(Image,
on_delete=models.CASCADE)
    result_data = models.CharField(max_length=250)
```

Листинг 2.2. Программный код форм веб-приложения. Файл models.py

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm,
UserChangeForm
from django.utils.translation import ugettext_lazy as _


class RegistrationForm(UserCreationForm):
    email = forms.EmailField(required=True)

    def __init__(self, *args, **kwargs):
        super(RegistrationForm, self).__init__(*args, **kwargs)
        self.fields['email'].label = "Email"
        self.fields['password1'].label = "Пароль"
```

```
        self.fields['password2'].label = "Пароль ещё раз"
        self.fields['username'].help_text = "150 символов или
меньше. Допускаются буквы, цифры и @/./+/-/_."
        self.fields['password1'].help_text = "Ваш пароль не
может быть похож на персональную информацию. " \
                                             "Ваш пароль должен
содержать не менее 8 символов. Ваш пароль не может " \
                                             "состоять только из
цифр."
        self.fields['password2'].help_text = "Ваш пароль ещё
раз."

    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')
        labels = {
            'username': _('Логин'),
        }

        def save(self, commit=True):
            user = super(RegistrationForm,
self.save(commit=False))
            user.email = self.cleaned_data['email']

            if commit:
                user.save()
            return user


class EditProfileForm(UserChangeForm):

    def __init__(self, *args, **kwargs):
        super(EditProfileForm, self).__init__(*args, **kwargs)
        self.fields['email'].label = "Email"
        self.fields['first_name'].label = "Имя"
        self.fields['last_name'].label = "Фамилия"
        self.fields.pop('password')

    class Meta:
        model = User
        fields = ('email', 'first_name', 'last_name')
```

Листинг 2.3. Программный код видов веб-приложения. Файл views.py

```
from django.shortcuts import render, redirect
from django.urls import reverse

from account.forms import RegistrationForm, EditProfileForm
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth import update_session_auth_hash
```

```
def registration(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect(reverse('account:home'))
        else:
            context = "Форма заполнена неверно"
            args = {
                'context': context,
                'form': form
                }
            return render(request,
'account/registration_form.html', args)
    else:
        form = RegistrationForm()

        args = {'form': form}
        return render(request, 'account/registration_form.html',
args)


def profile(request):
    args = {'user': request.user}
    return render(request, 'account/profile.html', args)


def edit_profile(request):
    if request.method == 'POST':
        form = EditProfileForm(request.POST,
instance=request.user)

        if form.is_valid():
            form.save()
            return render(request, 'account/profile.html')
        else:
            return redirect(reverse('account:edit_profile'))
    else:
        form = EditProfileForm(instance=request.user)
        args = {'form': form}
        return render(request, 'account/edit_profile.html',
args)


def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.POST,
user=request.user)

        if form.is_valid():
```

```
            form.save()
            update_session_auth_hash(request, form.user)
            return redirect(reverse('account:profile'))
        else:
            return redirect(reverse('account:change_password'))

    else:
        form = PasswordChangeForm(user=request.user)
        args = {'form': form}
        return render(request, 'account/change_password.html',
args)


def home(request):
    return render(request, 'i2d/home.html')


class CreateImage(generic.CreateView):
    initial = {'image_version': 0, 'image_stat': 0,
'image_prevImgID': 0}
    model = Image
    fields = ['image_path', 'image_name']
    success_url = '/i2d/images/'

    def form_valid(self, form):
        form.instance.image_userID = self.request.user
        return super(CreateImage, self).form_valid(form)


class ShowImages(generic.ListView):
    template_name = 'i2d/images.html'
    context_object_name = 'all_images'

    def get_queryset(self):
        return
Image.objects.filter(image_userID=self.request.user,
image_state=0)


class ShowImage(generic.DetailView):
    model = Image
    template_name = 'i2d/detail.html'


class UpdateImage(generic.UpdateView):
    model = Image
    fields = ['image_name']
    template_name = 'i2d/image_update_form.html'
    success_url = '/i2d/images/'


class DeleteImage(generic.DeleteView):
```

```
    model = Image
    success_url = '/i2d/images'

    def delete(self, request, *args, **kwargs):
        obj = self.get_object(self.get_queryset())
        imagePath = "{}{}{}".format(settings.MEDIA_ROOT,
os.path.sep, obj.image_path)

        obj.delete()
        os.remove(imagePath)

        return redirect('i2d:images')
```

Листинг 2.4. Код разметки базового шаблона. Файл base.html

```
{% load widget_tweaks %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    {% load static %}
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootst
rap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm
" crossorigin="anonymous">
    <link type="text/css" rel="stylesheet" href="{% static
'account/css/style.css' %}" />
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <a class="navbar-brand" href="{% url 'i2d:home'
%}">Анализ снимков крови</a>
        <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse"
id="navbarSupportedContent">
            <!-- Items left -->
            {%  if user.is_authenticated %}
            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link" href="{% url
'i2d:images' %}">2d изображения</a>
```

```
                </li>
            </ul>

                <!-- Items right -->
            <ul class="navbar-nav ml-auto">
                <li class="nav-item">
                    <a class="nav-link" href="{% url
'i2d:add_image' %}">Добавить изображение</a>
                </li>
                <li class="nav-item dropdown {% block
activeProfile %} {% endblock %}">
                    <a class="nav-link dropdown-toggle" href="{%
url 'account:profile' %}" id="navbarDropdown" role="button"
data-toggle="dropdown" aria-haspopup="true" aria-
expanded="false">
                        Профиль
                    </a>
                    <div class="dropdown-menu dropdown-menu-
right" aria-labelledby="navbarDropdown">
                        <a class="dropdown-item" href="{% url
'account:edit_profile' %}">Редактировать профиль</a>
                        <a class="dropdown-item" href="{% url
'account:change_password' %}">Смена пароля</a>
                        <div class="dropdown-divider"></div>
                        <a class="dropdown-item" href="{% url
'account:logout' %}">Выйти</a>
                    </div>
                </li>
            </ul>
            {% else %}

            <!-- Items right -->
            <ul class="navbar-nav ml-auto">
                <li class="nav-item {% block activeLogin %} {%
endblock %}">
                    <a class="nav-link" href="{% url
'account:login' %}">Вход</a>
                </li>
                <li class="nav-item {% block activeRegistration
%} {% endblock %}">
                    <a class="nav-link" href="{% url
'account:registration' %}">Регистрация</a>
                </li>
            </ul>
            {% endif %}
        </div>
    </nav>

    {% block body %}
    {% endblock %}

    <script src="https://code.jquery.com/jquery-
```

```
3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN
" crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd
/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q
" crossorigin="anonymous"></script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstra
p.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl
" crossorigin="anonymous"></script>
    <script defer
src="https://use.fontawesome.com/releases/v5.0.8/js/all.js"></sc
ript>
    {% block js %}
    {% endblock %}
</body>
</html>
```

Листинг 2.5. Программный код менеджера URL. Файл urls.py

```
from django.conf.urls import url
from account import views
from django.contrib.auth.views import login, logout,
password_reset, password_reset_done,\
    password_reset_confirm, password_reset_complete

app_name = 'account'

urlpatterns = [
    url(r'^login/$', login, {'template_name':
'account/login.html'}, name='login'),

    url(r'^logout', logout, {'template_name':
'account/logout.html'}, name='logout'),

    url(r'^registration/$', views.registration,
name='registration'),

    url(r'^profile/$', views.profile, name='profile'),

    url(r'^profile/edit', views.edit_profile,
name='edit_profile'),

    url(r'^change-password/$', views.change_password,
name='change_password'),

    url(r'^reset-password/$', password_reset,
```

```
        {'template_name': 'account/reset_password.html',
            'post_reset_redirect':
'account:password_reset_done',
         'email_template_name':
'account/reset_password_email.html'},
        name='reset_password'),

    url(r'^reset-password/done/$', password_reset_done,
{'template_name': 'account/reset_password_done.html'},
        name='password_reset_done'),

    url(r'^reset-password/confirm/(?P<uidb64>[0-9A-Za-z]+)-
(?P<token>.+)/$',
        password_reset_confirm, {'template_name':
'account/reset_password_confirm.html',
                                 'post_reset_redirect':
'account:password_reset_complete'},
        name='password_reset_confirm'),

    url(r'^reset-password/complete/$', password_reset_complete,
{'template_name': 'account/reset_password_complete'},
        name='password_reset_complete'),

]
```

Листинг 2.6. Код разметки главной страницы. Файл images.html

```
{% extends 'base.html' %}

{% block title %}
    2d изображения
{% endblock %}

{% block body %}
    <div class="container-fluid"
xmlns="http://www.w3.org/1999/html">
        <div class="row">
            <br>
        </div>
        <div class="row">
            {% if all_images %}
                {% for image in all_images %}
                    <div class="col-sm-4 col-lg-3">
                        <div class="thumbnail">
                            <a href="{% url 'i2d:detail'
image.id %}">
                                <img src="{{
image.image_path.url }}" class="img-fluid">
                            </a>
                            <div class="caption">
```

```
                                <h2 class="text-truncate">{{
image.image_name }}</h2>
                                <h4>{{ image.image_date }}</h4>

                                <!-- View Details -->
                                <a href="{% url 'i2d:detail'
image.id %}" class="btn btn-primary btn-sm"
role="button">Обзор</a>

                                <!-- Delete Image -->
                                <form action="{% url
'i2d:delete_image' image.id %}" method="post" style="display:
inline;">
                                    {% csrf_token %}
                                    <input type="hidden"
name="image_id" value="{{ image.id }}" />
                                    <button type="submit"
class="btn btn-default btn-sm">
                                        <i class="fas fa-trash-
alt"></i>
                                    </button>
                                </form>

                                <!-- Update Image -->
                                <form action="{% url
'i2d:update_image' image.id %}" method="post" style="display:
inline;">
                                    {% csrf_token %}
                                    <input type="hidden"
name="image_id" value="{{ image.id }}" />
                                    <button type="submit"
class="btn btn-default btn-sm">
                                        <i class="fas fa-pencil-
alt"></i>
                                    </button>
                                </form>

                            </div>
                        </div>
                    </div>
                {% endfor %}
            {% else %}
                <p>Вы пока не загружали изображений.</p>
            {% endif %}

            <div class="col-sm-4 col-lg-3 align-self-center ">
                <div class="thumbnail text-center ">
                    <a href="{% url 'i2d:add_image' %}"
class="addBtn">
                        <i class="far fa-plus-square fa-4x"></i>
                    </a>
                </div>
```

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

{% load static %}
{% block js %}
    <script defer src="{% static 'i2d/js/gears.js' %}"></script>
{% endblock %}
```