**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Институт\_\_\_\_\_№ 3 «Системы управления, информатика и электроэнергетика»\_\_\_\_\_**

**Кафедра307«Цифровые технологии и информационные системы»\_Группа\_М30-412Б**

**Направление подготовки\_\_\_09.03.02\_\_«Информационные системы и технологии»\_\_**

**Квалификация (степень)\_\_\_\_\_\_\_\_\_\_\_\_Бакалавр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

На тему: «Разработка CRM системы на Python/Django.»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Автор квалификационной работы \_\_\_\_Столбунов Ярослав Андреевич\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_Склеймин Юрий Борисович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_Склеймин Юрий Борисович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

**К з а щ и т е д о п у с т и т ь**

Зав. кафедрой \_\_\_\_\_\_\_\_307 Васильев Фёдор Владимирович\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, И, О)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021г.

Москва 2021 г.

**УКАЗАТЕЛЬ ОПРЕДЕЛЕНИЙ, ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ**

**CRM** (Customer Relationship Management) — модель взаимодействия, основанная на теории, что центром всей философии бизнеса является клиент, а главными направлениями деятельности компании являются меры по обеспечению эффективного маркетинга, продаж и обслуживания клиентов.

**CRM**-**система** —прикладное программное обеспечение, предназначенное для автоматизации стратегий взаимодействия с заказчиками (клиентами).

**ER-диаграмма** — схема «сущность-связь».

**Git** — распределённая система управления версиями.

**PEP8** — руководство по написанию кода на языке Python.

**SQL** — язык программирования структурированных запросов к базе данных.

**БД** — база данных.

**ПО** — программное обеспечение.

**ТЗ** — техническое задание.

**Фреймворк** — программная платформа, определяющая структуру программной системы.

**Python** — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью.

**MVC (**Model View Controller**)** — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер.

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

**ORM (**Object-Relational Mapper**)** — технология, предоставляющая возможность связи объекта в объектно-ориентированном языке программирования с сущностями, представленными в базе данных, такими как таблица, запись, запросы, функции.

ОГЛАВЛЕНИЕ

Введение…………………………………………………..……………..5

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ……………………………..……..8

* 1. Тестирование программного обеспечения………………..………..8
     1. Тестирование «стеклянного ящика» на стадии кодирования……..8
     2. Программные ошибки. Документирование и анализ ошибок…....8
  2. Обзор систем отслеживания ошибок………………...…………....11
  3. Разработка функциональных требований к системе………..……16
     1. Функциональные возможности…………………………...…….…16
     2. Контекстная диаграмма процесса………………..………………..17
     3. Диаграмма прецедентов……………………………………..……..18
     4. Обзор фреймворков Python для веб-разработки………………….19
     5. Концептуальная модель………………………………………..…..21

Глава 2. ПРАКТИЧЕСКАЯ ЧАСТЬ……………………………..……..23

* 1. Разработка структуры БД …………………………………...….…23
     1. Логическая модель данных………………………………………...23
     2. Физическая модель данных…………………………….………….24
  2. Разработка дизайн-макета сайта…………………………………..27
     1. Структурная схема…………………………………………...……..27
     2. Библиотека шаблонов CSS – Bootstrap………………..…………..38
     3. Описание программных модулей………………………...……….40
  3. Размещение веб-портала на хостинге……………………………..45
  4. Тестирование……………………………………………..……...…47

Заключение………………………………………...…………...……..51

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ……………………….52

Приложение А. Презентация………………..……………..……………..53

**ВВЕДЕНИЕ**

В настоящее время во всех сферах жизни широкое распространение получили системы управления взаимоотношениями клиента с организацией, так называемые системы CRM.

По сути, они представляют собой прикладное программное обеспечение для автоматизации избранной стратегии взаимодействия с клиентами.

Термин «клиент» в этих системах может принимать самую широкую трактовку, учитывающую сферу использования этих систем. Независимо от области применения системы CRM, понятие «клиент» является центральным в её создании и применении.

К числу основных преимуществ систем CRM следует отнеси:

1. Возможность организации широкомасштабной клиентской базы

с сегментацией по различным параметрам, не допускающей дублирование данных.

1. Автоматизацию работы персонала с клиентами, исключающую ошибки.
2. Реализацию унифицированной постановки задач сотрудникам

с персональным распределением и контролем их выполнения.

1. Организацию управления взаимодействием с клиентом на конечном этапе – например, на этапе продаж.
2. Обязательное включение в структуру CRM многовекторной базы (баз)

данных, обеспечивающей мониторинг процессов взаимодействия с клиентом и их аналитический анализ.

Учитывая выше изложенное, задание на выпускную квалификационную

работу сформулировано следующим образом:

«Разработать автоматизированную систему управления взаимоотношениями с клиентом (CRM) c использованием технологий Python и Django.»

**Объектом** исследования данной бакалаврской работы является процесс

взаимодействия оператора с клиентом по средствам CRM-системы.

**Предметом исследования** представленной бакалаврской работы является CRM-система.

**Целью** работы является разработка программного обеспечения,

реализующего стратегию CRM.

Для достижения цели, поставленной в работе, необходимо решить

следующие задачи:

− провести анализ предметной области с целью выявления процессов,

требующих автоматизации;

− изучить существующие программные средства и

технологии, обеспечивающие реализацию CRM стратегий;

− описать функциональные требования к проектируемой системе;

− построить модель данных на логическом и физическом уровнях;

− разработать автоматизированную систему (АС) и описать принципы

работы с ней;

− провести тестирование разработанной системы.

Структура дипломной работы представлена введением, двумя главами,

заключением, списком использованной литературы и приложением.

В первой главе дано описание процесса взаимодействия система-

оператор-клиент. Проведен обзор существующих CRM систем. Разработана

модель CRM системы и сформулированы требования к проектируемой АС по

видам обеспечения.

Во второй главе разработаны логическая и физическая модели данных

исследуемой предметной области. Разработано приложение АС и описана его

функциональность. Проведено тестирование приложения и описан порядок

запуска программы.

**Глава 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

* 1. **CRM-стратегия**

**1.1.1 Основные принципы**

CRM-стратегия – это представление компании о том, как работать с существующими клиентами и привлекать новых, принимая во внимание факт, что клиент является центральным элементом бизнеса. В частном виде это инструкция, которая объясняет, что и как должна делать CRM-система в рамках бизнеса.

В настоящее время для успешного создания и поддержки бизнеса все больше компаний используют CRM-стратегию.

Не важно какой продукт или услугу предоставляет компания. Не имеет значения и статус клиента, - он может быть, как физическим лицом, так и юридическим лицом (другой компанией). Ключевым, является то, что клиент – центральный элемент системы.

Для реализации указанной стратегии применяются следующие принципы:

1. Клиент в центре внимания.
2. Все данные о клиенте, его намерения, нынешний статус для компании и этапы взаимодействия должны быть сохранены и использованы для увеличения эффективности продаж и бизнес-процессов.
3. Компания непрерывно работает над созданием наилучших условий для клиента.
4. Постоянное упрощение действий клиента. Он должен быть уверен в качестве, своевременности и простоте получении услуги или продукта компании.

**1.1.2 Реализация CRM-стратегии через CRM-систему**

Для реализации вышеизложенных принципов CRM, используется специально программное обеспечение. CRM-система отвечает всем потребностям бизнеса за счёт продуманности и простоты использования.

Для успешного решения поставленных бизнесом задач в CRM-системе предусмотрены различные механизмы и алгоритмы:

1. Важнейшим элементом системы являются «воронки продаж». Воронка продаж — маркетинговая модель, описывающая предполагаемое «путешествие» нынешнего или будущего клиента от первого знакомства с предложением или товаром до реальной покупки.
2. Сводная таблица клиентов бизнеса, позволяющая в любой момент времени получать полную информацию о состоянии клиентской базы компании.
3. Декомпозиция задач менеджеров по сделкам и клиентам, реализующая их удобное представление для оператора.
4. Гибкая ролевая модель сотрудников, обеспечивающая раздельный доступ к действиям с данными в зависимости от обязанностей сотрудника (оператора) в рамках его должностных обязанностей.
5. Аналитическая составляющая, благодаря которой руководство может оперативно оценить и скорректировать бизнес-модель.
6. Бесшовное «движение» клиента вдоль воронки продаж, что позволяет операторам (менеджерам) эффективнее осуществлять свою профессиональную деятельность, затрачивая меньше времени и сил на документирование собственных действий.

**1.2 Обзор существующих CRM-систем.**

Для обзора существующих решений было выбрано 3 CRM-систем, предоставляемых на рынке. Все они по-разному осуществляют реализацию CRM-стратегии.

*amoCRM*

Это удобная система с простым и понятным интерфейсом. Ее функционал позволяет строить взаимодействие с клиентом практически на всех этапах продаж. Программа сама формирует воронку заказов, систематизирует и упорядочивает все заявки от клиентов, уведомляя вас об этом. А еще можно создавать календарный план для отдела продаж, а также ставить цели и задачи его сотрудникам.

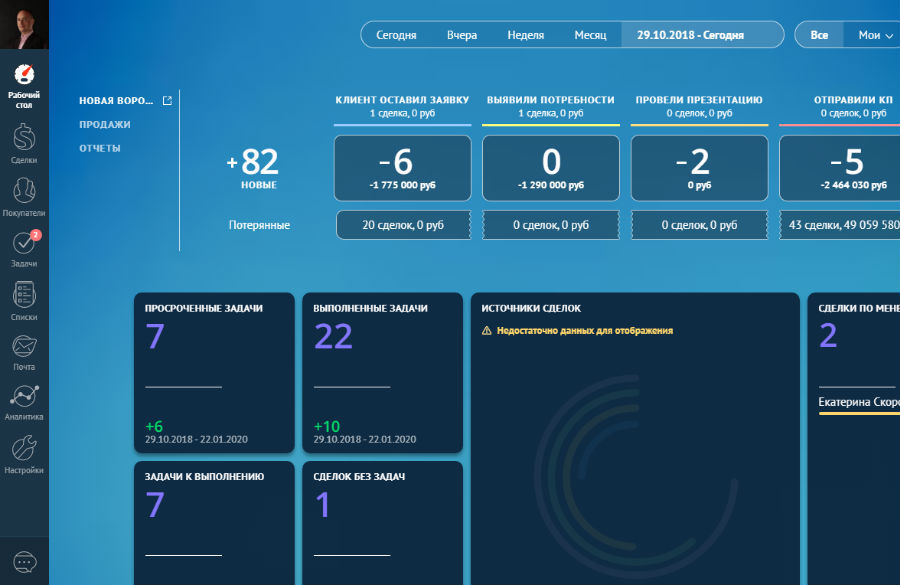


Рисунок 1.2.1.1 — Интерфейс amoCRM

Достоинством является то, что интерфейс адаптирован для смартфона. Это значит, что можно работать везде. amoCRM сохраняет данные каждого клиента, так что можно создавать e-mail рассылку и звонить прямо через мобильное приложение.

Кроме этого, программа предлагает дополнительный функционал — общий чат сотрудников, интеграцию службы поддержки в социальные сети.

На данный момент услугами этой CRM пользуются более 15 тысяч предприятий.

Для желающих протестировать программу существует пробный пакет, рассчитанный на 14 дней.

*Битрикс24*

Система нацелена на оптимизацию работы внутри коллектива компании. Ее функционал напоминает корпоративную социальную сеть — программа формирует "живую ленту" бизнеса, на которую стекаются все важные события, от задач руководителя до последних заказов клиента и его e-mail сообщений. Сотрудники могут присылать друг другу сообщения в чате.



Рисунок 1.2.1.2 — Интерфейс Битрикс24

Все это помогает наладить связь внутри команды. Благодаря системе Битрикс24 сотрудники более оперативно реагируют на разные задачи и вопросы клиентов. Становится не обязательным перемещение по офису для сбора и обмена информацией — Битрикс24 делает это за тебя. Существует мобильная версия, что является преимуществом этой системы.

*Salesforce*

Американская компания Salesforce считается заслуженным лидером в сфере CRM-систем. На нее приходится самая большая доля рынка — более 14%.

Основным преимуществом CRM-систем этой компании является учет отраслевой специфики клиента. Кроме того имеется удобный набор аналитических инструментов. Это позволяет отслеживать трафик потенциальных клиентов и анализировать эффективность продаж.

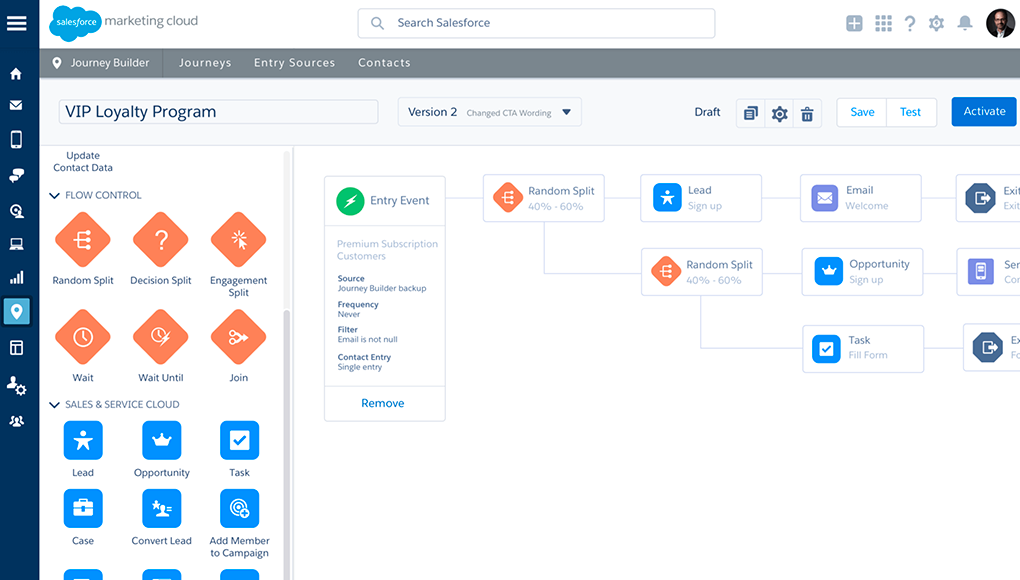
В дополнение к указанному система способна оценить маркетинговую стратегию бизнеса, а затем дать несколько советов по ее совершенствованию. В итоге предприниматель получает полный контроль над циклом работы. 

Рисунок 1.2.1.3 — Интерфейс Salesforce

Недостатком данных систем является то, что на текущий момент не существует русской локализации, а значит порог вхождения – знание английского на хорошем уровне, что подойдет очень немногим российским компаниям.

Таблица 1.1 — Сравнение CRM-систем

|  |  |  |  |
| --- | --- | --- | --- |
|  | amoCRM | Битрикс24 | Salesforce |
| Открытый исходный код | **˗** | **-** | **˗** |
| Бесплатное использование | **-** | **˗** | **˗** |
| Интуитивно понятный интерфейс | **+** | **-** | **-** |
| Формирование отчетов | **+** | **+** | **+** |
| Возможность кастомизации | **-** | **-** | **+** |
| Продукт предоставляется в исходном состоянии | **+** | **+** | **+** |
| Возможность изменение системы на этапе закупки и внедрения | **-** | **-** | **-** |

Основными недостатками всех рассмотренных систем является их сложность для потенциальных операторов (менеджеров). Для работы в выше перечисленных системах требуется приобретение дополнительных навыков по работе с конкретной системой, что невозможно без дополнительного обучения. Помимо этого, не исключены сложности в настройке системы и ее администрирования в рамках конкретных компаний.

Также, следует отметить, что все системы поставляются «как есть», что заставляет компании не редко изменять привычные способы работы и взаимодействия внутри команд. В системах отсутствует прямой доступ к исходному коду и базам данных, что уменьшает гибкость системы и усложняет процесс анализа работы сотрудников и бизнес-процессов для руководства.

Ещё одним из недостатков рассмотренных систем, является отсутствие возможности локального развёртывания (CRM после установки не требует подключения к глобальной Ethernet сети).

CRM-система, разработанная в рамках данной бакалаврской работы, направлена на решение проблем использования в этих системах классических решений. Преимуществом разработанного программного обеспечения, по сравнению с вышеперечисленными является:

- открытый исходный код (что предоставляет возможность изменения, доработки и улучшения непосредственно для нужд отдельных компаний);

- интуитивно понятный интерфейс, который позволяет не загромождать дополнительными функциями работу системы;

- возможность формировать отчеты в формате .xlsx;

- открытый доступ к базе данных.

- локальная установка системы внутри компании.

**1.3 Разработка функциональных требований к системе**

**1.3.1 Функциональные возможности**

Вход в систему могут иметь только зарегистрированные пользователи, прошедшие идентификацию по логину и паролю, которые, в свою очередь, задаются при первом входе в систему в процессе регистрации. Роли и права пользователей определяются на этапе конфигурирования и установки, учитывая пожелания заказчика. Такой принцип работы системы позволяет каждому пользователю иметь доступ к тем элементам системы, которые определены в рамках его должностных обязанностей в компании. Система реализует простой способ удаления, изменения и создания новых операторов (сотрудников).

Перед определением функциональных требований, целесообразно ввести понятие **CRUD**- создание (create), чтение (read), модификация (update), удаление (delete). Данному принципу должны соответствовать все сущности в системе. Это позволит гибко управлять CRM в любом сценарии использования.

Разработанная в рамках данной бакалаврской работы система, предполагает реализацию следующих основных функциональных возможностей:

– Цикл работы с клиентом (CRUD);

– Удобная, визуальная реализация «воронки продаж»;

– Гибкий способ декомпозиции задач по каждому клиенту (CRUD);

– Поддержка гибкой аналитики на этапе внедрения и использования.

**1.3.2 Контекстная диаграмма процесса**

Для описания основных функций системы и ее взаимодействия с внешней средой предлагается следующая структурно–функциональная диаграмма изучаемого процесса, проектирование которой производится на основе стандарта IDEF0 [7]. Схема контекстной диаграммы процесса «Реализация CRM-стратегии» представлена на рисунке 1.3.1.

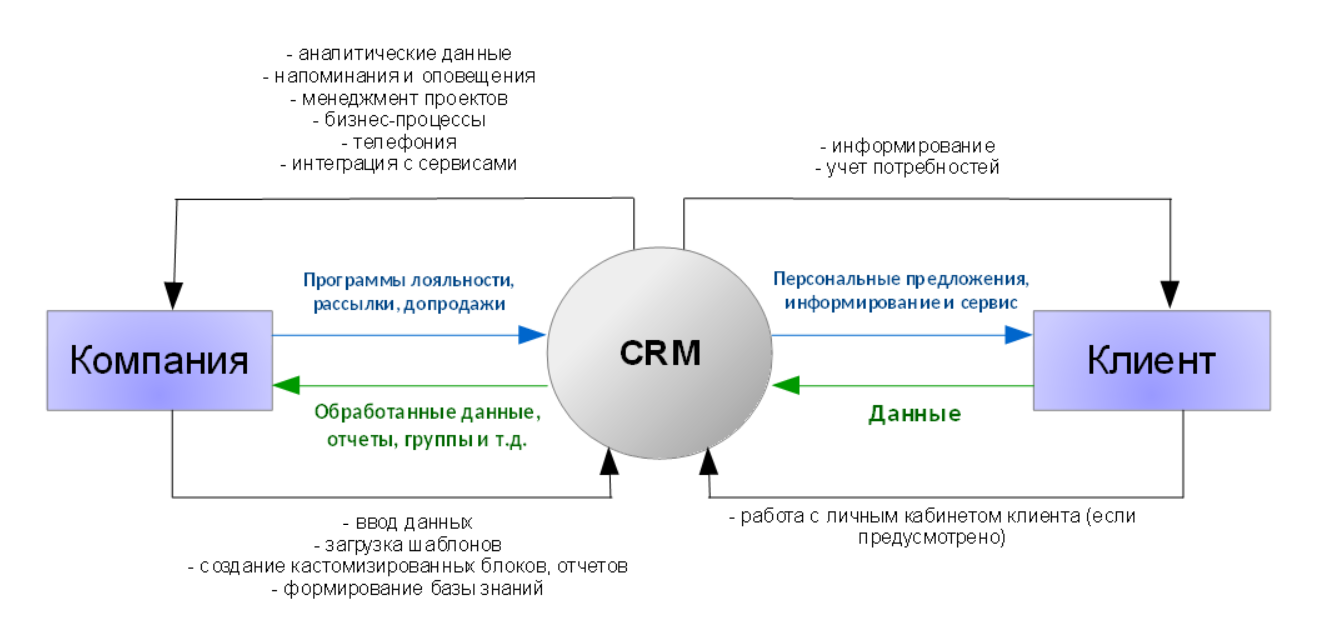


Рисунок 1.3.2.1 Контекстная диаграмма процесса «Реализация CRM-стратегии»

На диаграмме дается общее представление изучаемого процесса. Результат внедрения и использования CRM-стратегии считается успешным, если был достигнут статистически заметный рост объёмов продаж, а также уменьшено время обработки обращений клиентов при неизменном качестве.

Поясняя рисунок, стоит отметить, что процесс взаимодействия с клиентом может быть реализован с помощью любых каналов связи в удобной клиенту и компании форме. Также, в различных компаниях, процесс анализа и отчётности строится по-разному, на основании многих факторов, что предусматривается системой на этапе персонализации под конкретного заказчика.

**1.4 Используемые подходы и технологии.**

В данном разделе будут рассмотрены различные технологии и принципы их взаимодействия. С их помощью в дальнейшем будет реализована описанная выше функциональность CRM-системы.

**1.4.1 Модель взаимодействия «клиент-сервер».**

Для организации возможности простого и понятного взаимодействия человека, не имеющего специальной подготовки, с информационными системами, существует специальная модель взаимодействия. Клиент и сервер физически представляют собой программы. Например, классическим клиентом может являться браузер, а сервером – специализированное программное обеспечение, обеспечивающее функционирование сайта.

На рисунке 1.4.1 представлена классическая схема клиент-серверного взаимодействия.

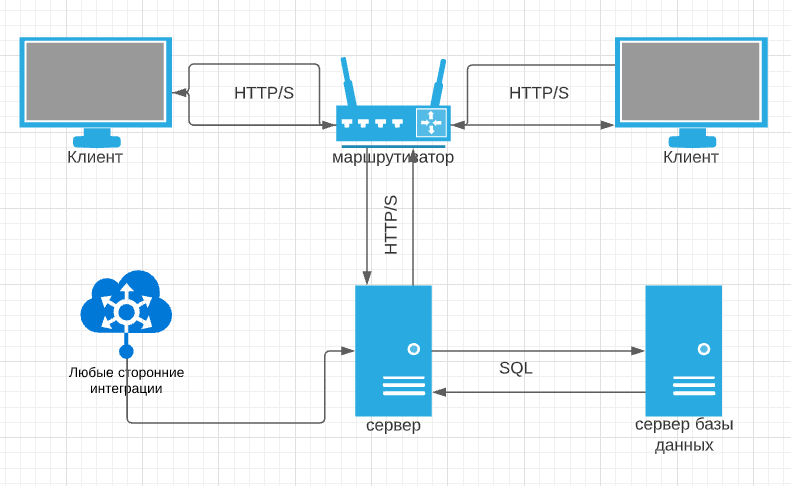


Рисунок 1.4.1.1 Модель клиент-сервер

Данная модель взаимодействия позволяет оградить пользователя от взаимодействия с внутренним устройством продукта, предоставляя ему лишь удобный пользовательский интерфейс.

Представленная архитектура является хорошо масштабируемой и не зависит от наличия подключения клиентов и сервера к глобальной сети «Интернет» и способна функционировать в замкнутых локальных сетях. Её обновление или внесение правок (изменений) не представляет большой сложности для разработчиков.

**1.4.2 Компьютерная сеть и протоколы.**

****

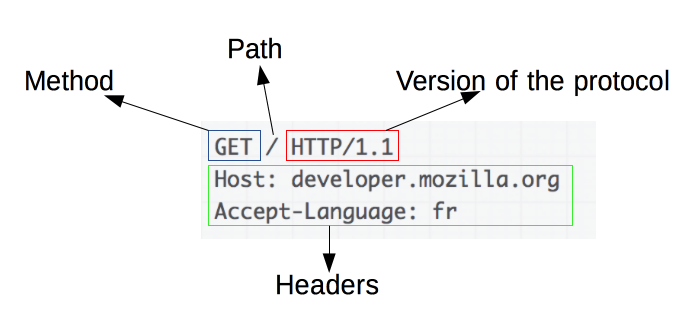
Рисунок 1.4.2.1 «Модель OSI»

Сетевая модель OSI (The Open Systems Interconnection model) —модель набора сетевых протоколов. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Модель определяет различные уровни взаимодействия систем. Каждый уровень выполняет свои определённые функции.

В настоящий момент, для разработки собственных программных продуктов, программистам не обязательно знать, как реализовано взаимодействие на уровни ниже четвёртого (Транспортный). Это позволяет уделить большее внимание конечному продукту, - его удобству применения, надёжности и безопасности.

В качестве примера, рассмотрим самый важный для разработчиков уровень – седьмой. Он обеспечивает пользовательской прикладной программе доступ к сетевым ресурсам. Примерами задач, решаемых прикладным уровнем могут послужить: передача файлов, электронная почта, управление сетью.

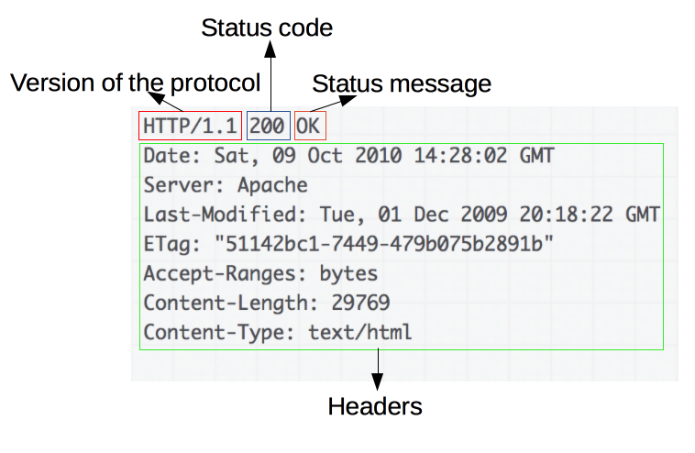
Самым популярным протоколом взаимодействия на высоком уровне является HTTP (HyperText Transfer Protocol) — протокол трансфера данных, изначально предназначенный для передачи гипертекстовых документов (HTML - HyperText Markup Language). В процессе его использования предполагается реализация клиент-серверной структуры обмена информацией. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает входные данные, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом. Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. Благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

*Структура запроса*:

1. Метод (GET, POST, PUT, DELETE...).
2. Путь, нужен для вызова определённого поведения веб-сервера.
3. Версия протокола.

Рисунок 1.4.2.2 «HTTP запрос»

1. Заголовки - опциональные параметры, предназначенные для передачи различной информации о клиенте.



*Структура ответа*:

1. Версия протокола.
2. HTTP код состояния, сообщающий об успешности запроса или причине неудачи.

Рисунок 1.4.2.1 «HTTP ответ»

1. Сообщение состояния - краткое описание кода состояния.
2. HTTP заголовки, подобно заголовкам в запросах.

Опционально ответ может содержать тело, представляющее пересылаемый ресурс.

Далее в данной работе будет представлен программный продукт построенный на архитектуре клиент-сервер с использованием протокола HTTP.

**1.4.3. Python как средство разработки программного обеспечения.**

В конце 1989 года Гвидо Ван Россум (голландский программист) создал Python – новый интерпретируемый язык программирования, который очень быстро стал популярен и востребован у программистов. Его используют крупнейшие мировые компании такие как: Google, Microsoft, Facebook, Yandex и многие другие.

*Возможности языка:*

1. Объектно-ориентированное программирование (ООП). Всё в Python является объектом: класс, функция, строка, чисто, список и т.д.
2. Обобщенное и функциональное программирование.
3. Модули и пакеты. Программное обеспечение (ПО) на Python оформляется в виде модулей, которые могут быть собраны в пакеты.
4. Интроспекция. Это возможность запросить тип и структуру объекта во время выполнения программы. В Python для любого объекта можно узнать всю информацию о его внутренней структуре.
5. Обработка исключений, итераторы и генераторы, управление контекстом выполнения, декораторы, регулярные выражения.

*Преимущества:*

1. Язык характеризуется логичным синтаксисом, вследствие чего исходный код программ, написанных на Python легко читается и воспринимается.
2. Гибкость и масштабируемость. Он позволяет разработчикам адаптировать высокоуровневую логику приложения, что позволяет легко расширять сложные приложения по мере необходимости.
3. Разработка идет быстрее, чем на большинстве других языков
4. Python является интерпретируемым языком программирования. Это значит, что до запуска он представляет собой обычный текстовый файл. Соответственно, поддерживается кроссплатформенность.
5. Удобный пакетный менеджер «PIP».

Благодаря вышеперечисленным свойствам, этот язык можно использовать для решения широкого спектра задач, от создания служебных приложений и веб-серверов до разработки игр и глубокого обучения нейронных сетей.

Для реализации задач, поставленных на этапе разработки функциональных требований, предлагается использовать язык Python и фреймворк Django, с целью создания серверной части CRM-системы.

**1.4.4. Веб-Фреймворк Django.**

Веб-фреймворк — это каркас для написания веб-приложений. Он определяет структуру, задаёт правила и предоставляет необходимый набор инструментов для разработки.

*Архитектура веб-фреймворков.*

Архитектура почти всех популярных веб-фреймворков основана на декомпозиции нескольких отдельных слоёв (приложения, модули и т.д.). Это позволяет расширять функциональность, исходя из потребностей.

Для обеспечения масштабируемости, безопасности, удобства, надёжности и увеличения скорости развёртывания в 1988 году была сформирована модель MVC.

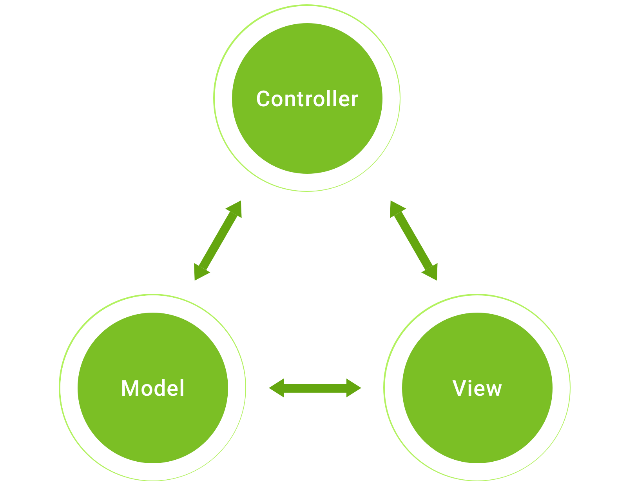
MVC (Model-View-Controller) — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Рисунок 1.4.4.1 «MVC»

* Модель содержит все данные и уровни бизнес-логики, её правила и функции.
* Представление отвечает за визуальное отображение данных.
* Контроллер преобразует входные данные в команды для Модели и Представления.

Придерживаясь этой схемы, легко вносить изменения в одну конкретную часть приложения без ущерба для остальных.

*Django*

Django - это высокоуровневый веб-фреймворк Python, который способствует быстрой разработке и чистому дизайну веб-приложений. Его исходный код открыт, а использование бесплатно, что открывает широкие возможности для разработки.

Преимущества данного фреймворка:

1. Принцип «Всё включено» - большинство инструментов для создания приложения — часть фреймворка, а не поставляются в виде отдельных библиотек.
2. Стандартизированная структура - Django задаёт структуру проекта. Она помогает разработчикам понимать, где и как добавлять новую функциональность.
3. Приложения (apps) - подход, позволяющий легко интегрировать готовые решения добавлением их в структуру проекта и в список settings.INSTALLED\_APPS.
4. Безопасность по умолчанию - включает механизмы предотвращения распространенных атак вроде.
5. Легко дополнить – существует большая кодовая база, созданная сторонними разработчиками, облегчающая создание продукта.

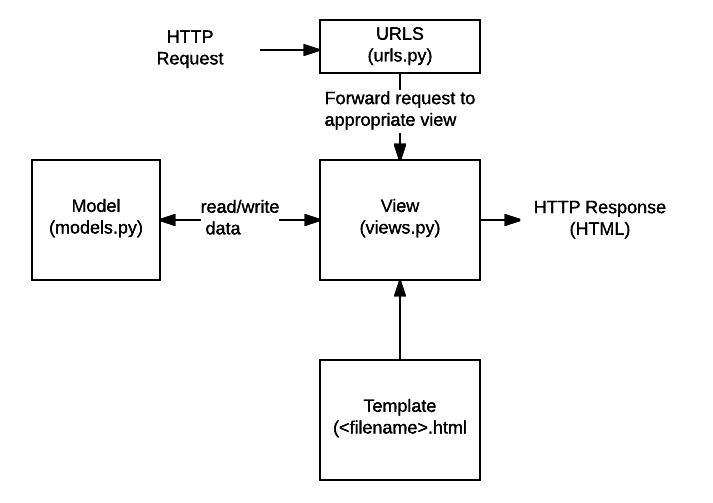


Рисунок 1.4.4.2 Структура веб-приложения, написанного на Django

*Django-ORM*

При создании веб-приложений нельзя обойтись без какой-либо базы данных. С её помощью можно хранить человеко-читаемые данные и одновременно взаимодействовать с ними с помощью программы. Обычно это означает, что разработчику системы необходим средний или выше уровень знаний в языке структурированных запросов (SQL), изучение которого требует большое количество человеко-часов. Для решения данной проблемы создан Django-ORM – часть общего фреймворка Django.

В общем случае ORM (Object-Relational Mapping) - технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Такое представление является удобным и безопасным методом взаимодействия программы с любой реляционной базой данных.

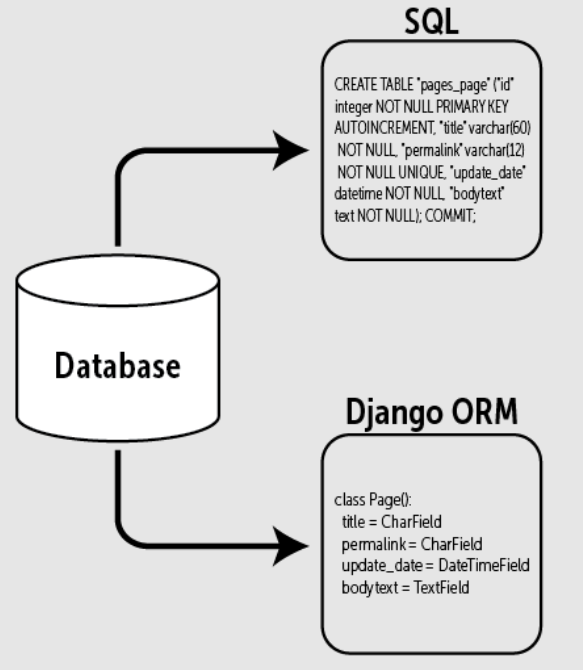
В Django-ORM это часть программного кода (модуль на языке Python), предоставляющий возможность доступа к базе данных при помощи специализированных моделей. Моделью является отображение определённой таблицы в базе данных, включающее в себя все поля этой таблицы и их тип. В языке Python модель будет представлена конструкцией «Класс» (class).

Рисунок 1.4.4.3 «ORM»

Для реализации взаимодействия с данным внутри таблицы внутри базового класса модели определены следующие методы:

1. .all()- Все записи в таблице (SELECT \* FROM TABLE)
2. .get(column=parameter)- Одна запись (SELECT \* FROM TABLE WHERE column= parameter)
3. .filter(column1=parameter1, column2=parameter2, …)- все записи соответствующие запросу (SELECT \* FROM TABLE WHERE column1=parameter1 AND column2=parameter2 AND ...)
4. Model(column1=parameter1, column2=parameter2, …).save() – сохранение изменений в записи или её создание в случае отсутствия.
5. Model(column1=parameter1, column2=parameter2, …).delete() – удаление конкретной записи.

Используя эти методы легко работать с данными из создаваемого приложения.

На момент написания данной дипломной работы, для разработчиков, использующих Django-ORM доступны следующие базы данных: PostgreSQL, MySQL, SQLite, а также Microsoft® SQL. В процессе разработки будет использована база PostgreSQL.

**1.4.5 База данных PostgreSQL.**

PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД).

Преимущества:

1. поддержка БД неограниченного размера;
2. мощные и надёжные механизмы транзакций и репликации;
3. Широкий набор поддерживаемых типов полей;
4. Легкая расширяемость.

Данное программное обеспечение поддерживает большинство операционных систем, обеспечивает простую установку и настройку. Для развёртывания данной СУБД не требуются глубокие знания администрирования ОС. Также следует отметить, что начиная с версии 9.6. PostgreSQL поддерживает тип данных JSON (JavaScript Object Notation - формат для хранения и обмена информацией, доступной для чтения человеком). Этот факт позволяет разработчика более гибко взаимодействовать с данными.

**1.4.6 Клиент приложения.**

Для того что бы человек мог без профессиональной подготовки использовать информационные системы, в клиент-серверной модели взаимодействия предусмотрено специальное программное обеспечение – клиент. В различных сценариях использование им может являться что угодно. Для CRM-систем характерно применение Веб-браузера в качестве клиентской стороны. Для отображения человеко-читаемой информации применяются различные технологии. Среди них:

1. HTML – Гипертекстовый документ имеющий разметку.
2. CSS (bootstrap) – Язык создания стилей для отображения HTML. Bootstrap- библиотека стилей.
3. JavaScript (jQuery) – Интерпретируемый язык программирования, использующийся в браузерах (и не только) для описания логики работы клиентской части приложения, например, создание анимации, изменение страницы без перезагрузки, отправка данных на сервер. jQuery – библиотека облегчающая работу с языком.

При создании клиента для пользователя, данный набор технологий позволяет дать ему максимально удобный и интуитивно понятный интерфейс.

**Глава 2. ПРАКТИЧЕСКАЯ ЧАСТЬ**

**2.1 Разработка структуры базы данных**

**2.1.1 Логическая модель данных**

Схема «сущность-связь» (также ERD или ER-диаграмма) — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграмма представлена на рисунке 2.1.1.1

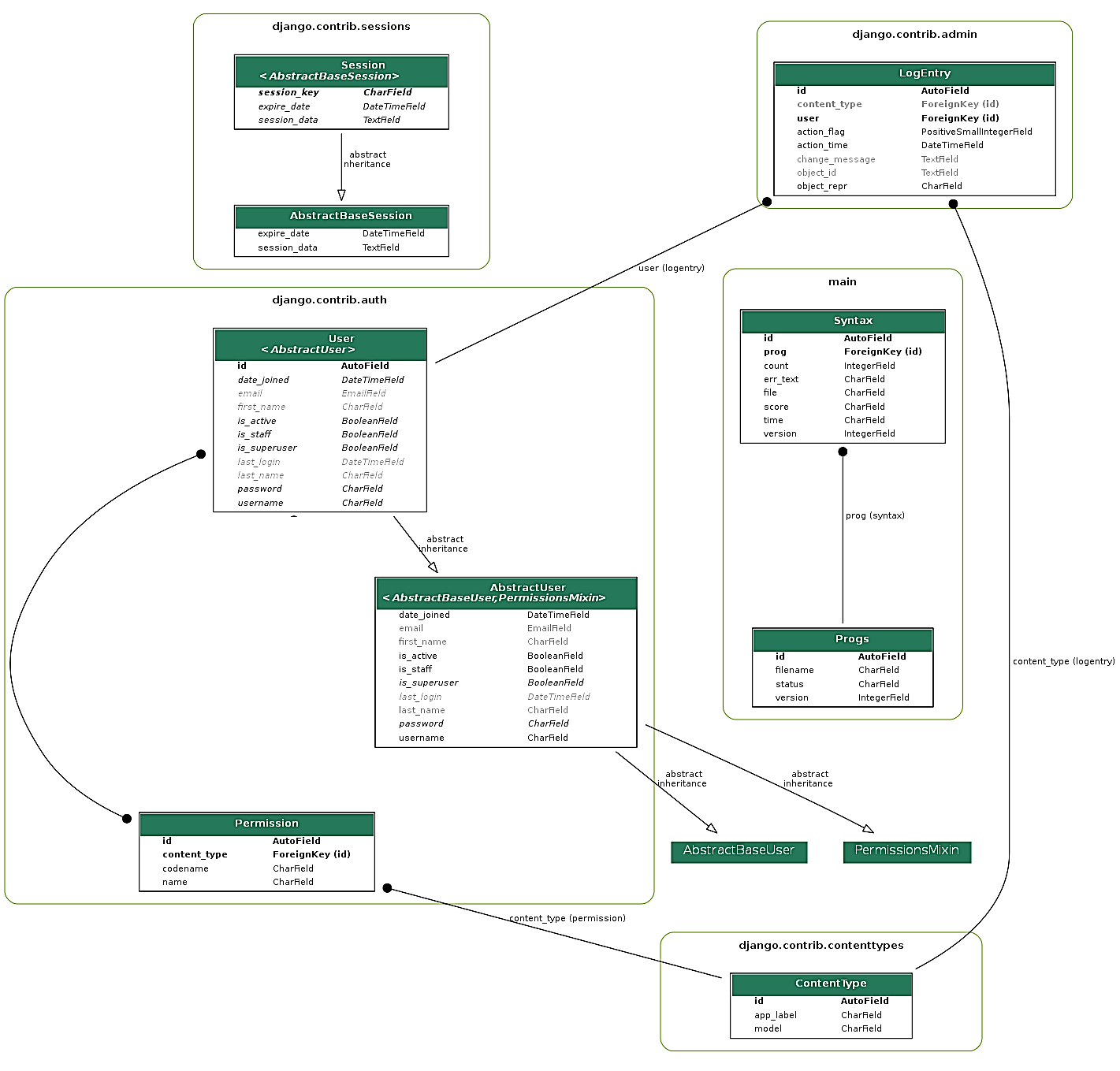


Рисунок 2.1.1.1 ER-диаграмма

**2.1.2 Физическая модель данных**

При создании физической модели данных отношения, описанные в логической модели, преобразуются в таблицы, атрибуты становятся столбцами, каждому атрибуту присваивается тип данных, а для ключевых столбцов создаются уникальные индексы.

Как видно из рисунка 2.1.1.1, проектируемая БД состоит из 7 таблиц, перечень и структура которых представлены ниже.

Сущности, хранящие загруженные программы «main\_progs» и найденные в них ошибки «main\_syntax».

Таблица 2.1. Атрибуты сущности «main\_progs»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| id | int(11) | Код программы |
| filename | varchar(200) | Имя файла программы |
| status | varchar(200) | Статус проверки |
| version | int(11) | Версия программы |

Таблица 2.2. Атрибуты сущности «main\_syntax»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| id | int(11) | Код ошибки |
| time | varchar(200) | Время обнаружения ошибки |
| version | int(11) | Версия программы, в которой обнаружена ошибка |
| err\_text | varchar(1000) | Описание ошибки |
| score | varchar(200) | Чистота кода |
| count | int(11) | Количество ошибок |
| prog\_id | int(11) | Код программы. в которой обнаружена ошибка |
| file | varchar(200) | Имя файла, в котором обнаружена ошибка |
| test\_passed | tinyint(1) | Пройден ли тест |

Создание настраиваемого механизма сеанса, управляемого базой данных, на основе тех, что включены в Django (т.е. db и cached\_db ), может быть выполнено путем наследования AbstractBaseSession от одного из классов. SessionStore,

Таблица 2.3. Атрибуты сущности «django\_session»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| session\_key | varchar(40) | Первичный ключ. Текущая реализация генерирует 32-символьную строку (случайную последовательность строчных цифр и букв ASCII). |
| session\_data | longtext | Строка, содержащая закодированный и сериализованный словарь сеанса. |
| expire\_date | datetime(6) | Дата / время, указывающие дату окончания сеанса. |

Миграции - это способ Django распространять изменения, которые вносятся в модели (добавление поля, удаление модели и т. д.) в схему базы данных.

Таблица 2.4. Атрибуты сущности «django\_migrations»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| id | int(11) | Код миграции |
| app | varchar(255) | Кем проводится миграция |
| name | varchar(255) | Имя миграции |
| applied | datetime(6) | Дата / время, принятия миграции |

Для авторизации пользователей используются следующие сущности:

Таблица 2.5. Атрибуты сущности «auth\_user\_user\_permissions»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| id | int(11) | Первичный ключ. |
| user\_id | int(11) | Код пользователя |
| permission\_id | int(11) | Код операции |

Таблица 2.6. Атрибуты сущности « auth\_user»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| id | int(11) | Код пользователя |
| password | varchar(128) | Пароль пользователя |
| last\_login | datetime(6) | Дата / время последнего входа в систему |
| is\_superuser | tinyint(1) | Имеет ли привилегии суперполльзователя |
| username | varchar(150) | Имя пользователя |
| first\_name | varchar(150) | Имя |
| last\_name | varchar(150) | Фамилия |
| email | varchar(254) | Почта |
| is\_staff | tinyint(1) | Является ли админом |
| is\_active | tinyint(1) | Находится ли в сети |
| date\_joined | datetime(6) | Дата / время регистрации |

Таблица 2.7. Атрибуты сущности «auth\_permission»

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| id | int(11) | Код разрешения |
| name | varchar(255) | Описание операции |
| content\_type\_id | int(11) | Кол операции |
| codename | varchar(100) | Название операции |

**2.2 Разработка дизайн-макета сайта**

**2.2.1 Структурная схема**

Возможная реализация основных функциональных возможностей системы представлена на рисунке 2.2.1 в виде структурной схемы.

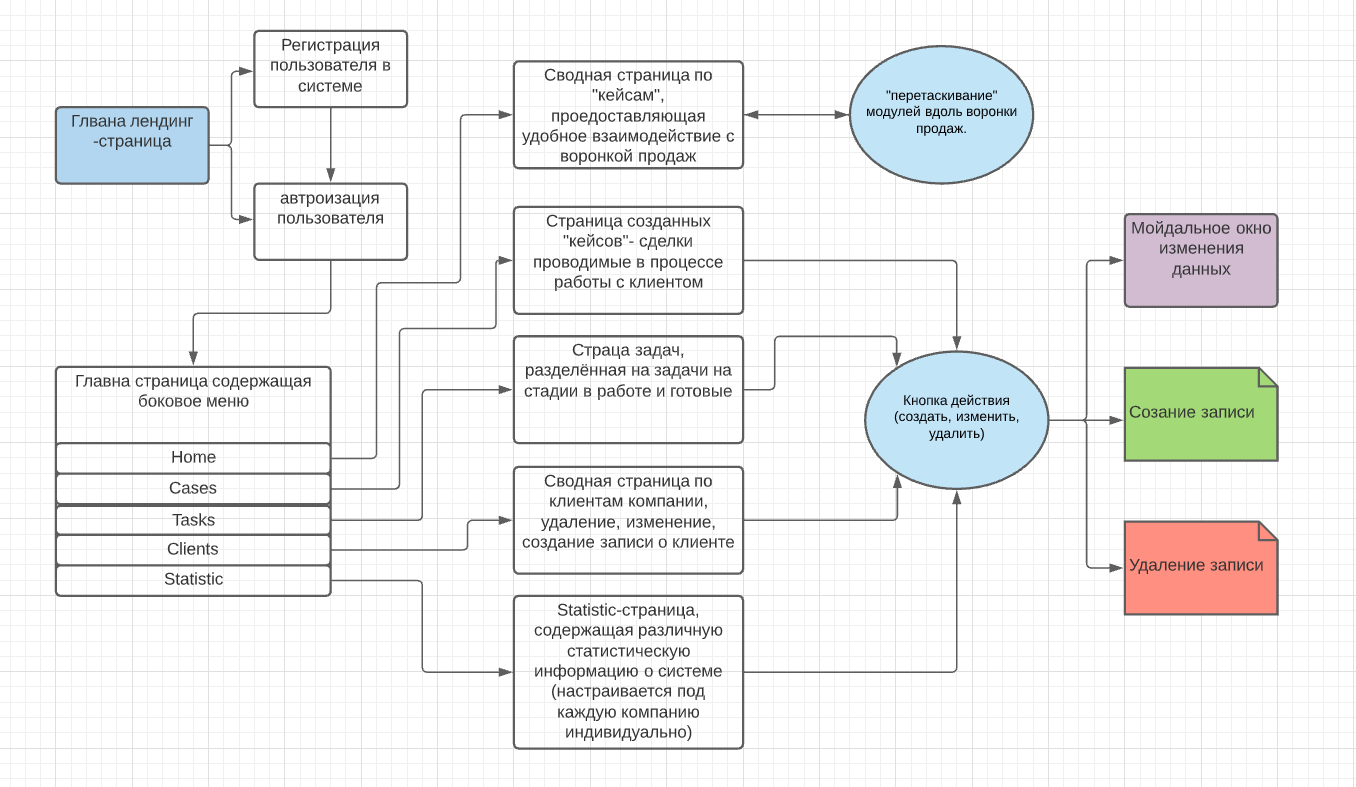


Рисунок 2.2.1.1 Структурная схема

Django основан на архитектуре MVT (Model-View-Template). MVT — это шаблон проектирования программного обеспечения для разработки веб-приложения. Структура MVT состоит из следующих трех частей:

- Модель: выступает в качестве интерфейса логической структуры данных.

- Представление: пользовательский интерфейс.

- Шаблон: статические части желаемого вывода HTML.

Таким образом, для реализации системы с представленной выше структурой и используя фреймворк Django, необходимо создать следующий проект (рис.2.2.2).

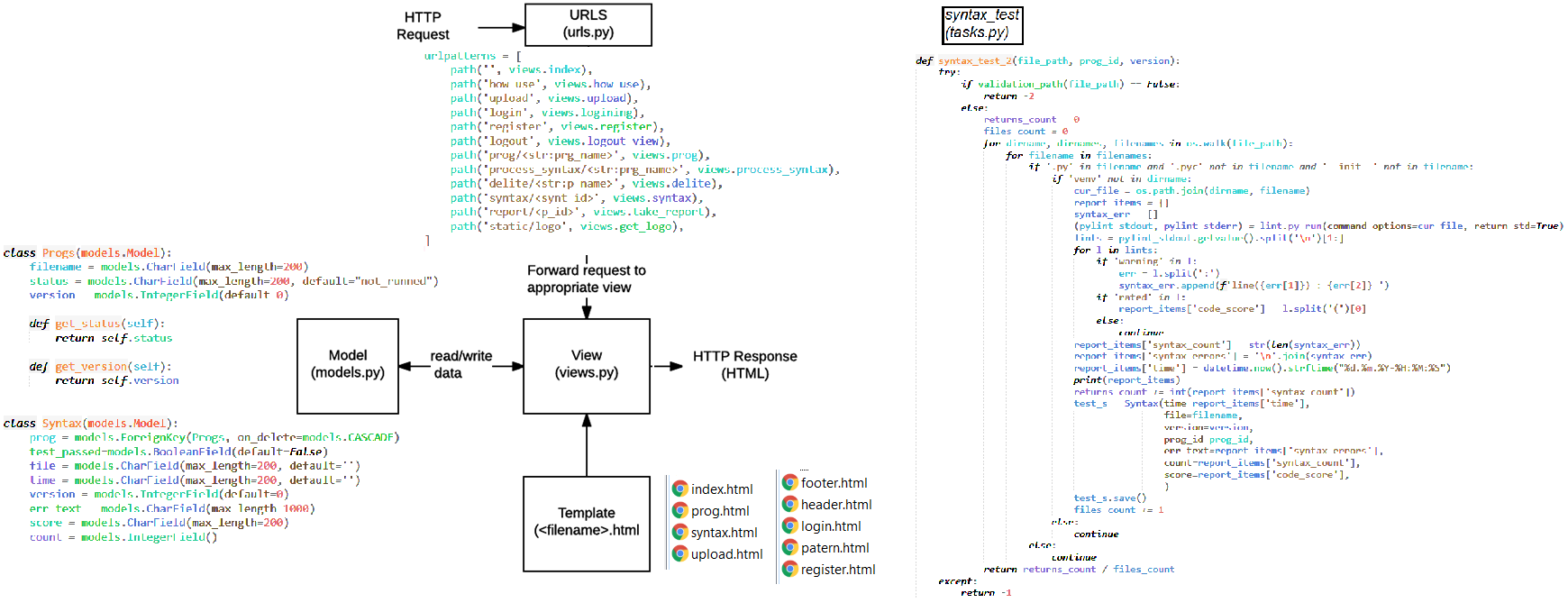


Рисунок 2.2.2 Схема проекта

Верхняя структура проекта будет следующей:

vkrlinter.ru/ # Папка сайта

manage.py # Скрипт для управления проектов (создан manage.py)

linter/ # Папка сайта/проекта (создана manage.py)

main/ # Папка приложения (также создана manage.py)

Для создания проекта в папке сайта vkrlinter.ru создаем папку linter, используя команду django-admin startproject linter. Команда django-admin создаст следующую файловую структуру:

vkrlinter.ru/

manage.py

linter/

settings.py

urls.py

wsgi.py

Подпапка проекта linter это ключевая директория нашего проекта:

*settings.py* содержит в себе все настройки проекта. Здесь мы регистрируем приложения, задаём размещение статичных файлов и настройки базы данных.

*urls.py* задаёт ассоциации url адресов с представлениями.

*wsgi.py* используется для налаживания связи между Django приложением и веб-сервером.

Скрипт *manage.py* используется для создания приложений, работы с базами данных и для запуска отладочного сервера.

Команда python3 manage.py startapp main создаст новую папку и наполнит её файлами различных частей приложения.

Большинство файлов названы, исходя из их назначения (контроллеры(views) должны находится во views.py, модели в models.py, тесты в tests.py, настройки административной части в admin.py, регистрация приложения в apps.py) и уже содержат некоторый шаблонный код для работы с вышеназванными объектами. Папка migrations используется, чтобы хранить «миграции» — файлы, которые позволяют автоматически обновлять базу данных по мере изменения моделей. \_\_init\_\_.py — пустой файл для того, чтобы Django и Python распознавали папку как Python модуль и позволяет использовать его объекты внутри других частей проекта.

Обновлённая директория выглядит следующим образом:

vkrlinter.ru/

manage.py

linter/

main/

admin.py

apps.py

models.py

tests.py

views.py

\_\_init\_\_.py

migrations/

После создания приложения, нужно зарегистрировать его в проекте, чтобы различные утилиты затрагивали его своим действием (например при добавлении моделей в базу данных). Приложения регистрируются добавлением их названий в список INSTALLED\_APPS в настройках проекта(settings.py).

В файл vkrlinter.ru/linter/settings.py в списке INSTALLED\_APPS подключены приложения, позволяющие поддерживать админ-панели Django (включая сессии, аутентификацию и т. д.).

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'main'

]

Так как в проекте используется база данных SQLite, она автоматически настроена в settings.py:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'u1356936\_default',

'USER': 'u1356936\_default',

'PASSWORD': '\_dAcr67r',

'HOST': 'localhost',

}

}

Откроем файл файл vkrlinter.ru/linter/urls.py. URL соотношения хранятся в переменной urlpatterns, которая является списком функций path(). Каждая path() функция или ассоциирует шаблон URL с контроллером(views) или же его с другим таким списком (во втором случае, первый URL становится «базовым» для других, которые определяются в дочернем списке). Список urlpatterns инициализирует список функции, которая, например, соотносит admin/ с модулем admin.site.urls , который содержит собственный файл-соотноситель.

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('main.urls'))

]

Создаем файл urls.py внутри папки main, в нем создадим URL соотношения:

from django.urls import path

from . import views

urlpatterns = [

path('', views.index),

path('how\_use', views.how\_use),

path('upload', views.upload),

path('login', views.logining),

path('register', views.register),

path('logout', views.logout\_view),

path('prog/<str:prg\_name>', views.prog),

path('process\_syntax/<str:prg\_name>', views.process\_syntax),

path('delite/<str:p\_name>', views.delite),

path('syntax/<synt\_id>', views.syntax),

path('report/<p\_id>', views.take\_report),

path('static/logo', views.get\_logo),

]

Django использует Объектный Соотноситель Связей (ORM) чтобы соотносить определения моделей в Django приложении со структурами данных, которые используются базой данных. Когда мы меняем наши модели, Django отслеживает изменения и может создать файлы миграций (в папке /vkrlinter.ru/main/migrations/), чтобы применить соответствующие структуры данных к базе, чтобы та соответствовала модели.

При создании сайта, Django автоматически добавил несколько моделей, которые можно использовать в админ-панели. Следующие команды создадут таблицы в базе данных, соответствующие этим моделям:

python3 manage.py makemigrations

python3 manage.py migrate

Команда makemigrations создаёт миграции для всех приложений, которые установлены в проект. Команда migrate применяет созданные миграции к базе (Django отслеживает, какие миграции были созданы для данной базы).

После изменений файлов проекта для отображения изменений необходимо перезапустить проект. Для этого создаем файл .restart-app в корневой директории сайта. После перезапуска проекта файл удаляется автоматически.

Веб-приложения Django получают доступ и управляют данными через объекты Python, называемые моделями. Модели определяют структуру хранимых данных, включая типы полей и, возможно, их максимальный размер, значения по умолчанию, параметры списка выбора, текст справки для документации, текст меток для форм и т. д.

Django позволяет определять отношения, как один к одному (OneToOneField), один ко многим (ForeignKey) и многие ко многим (ManyToManyField).

Модели определяются в приложении models.py. Они реализуются как подклассы django.db.models.Model, и могут включать поля, методы и метаданные.

from django.db import models  
  
  
class Progs(models.Model):  
 filename = models.CharField(max\_length=200)  
 status = models.CharField(max\_length=200, default=**"not\_runned"**)  
 version = models.IntegerField(default=0)  
  
 def get\_status(self):  
 return self.status  
  
 def get\_version(self):  
 return self.version  
  
  
class Syntax(models.Model):  
 prog = models.ForeignKey(Progs, on\_delete=models.CASCADE)  
 test\_passed=models.BooleanField(default=False)  
 file = models.CharField(max\_length=200, default=**''**)  
 time = models.CharField(max\_length=200, default=**''**)  
 version = models.IntegerField(default=0)  
 err\_text = models.CharField(max\_length=1000)  
 score = models.CharField(max\_length=200)  
 count = models.IntegerField()

Шаблон в верхней части страницы импортирует модуль моделей, который содержит базовый класс модели models.Model, от которого наследуются наши модели.

Модель *Progs* используется для хранения информации о программе, которую загрузит пользователь. Модель имеет:

- CharField field (filename), которое используется для описания имени программы (оно ограничено 200 символами).

- CharField field (status), которое используется для описания статуса проверки программы (оно ограничено 200 символами) и по умолчанию не запущено («not\_runned»).

- IntegerField (version), которое используется для указания номера версии загруженной программы.

В конце модели мы объявляем метод get\_status(), который возвращает статус проверки каждой программы, и метод get\_version(), который возвращает номер версии.

Модель *Syntax* используется для хранения информации о найденных синтаксических ошибках в программе, которую загрузит пользователь. Так как в одной программе может быть несколько ошибок, используется отношение один ко многим (ForeignKey), причем указано, чтобы при удалении программы удалялись и все найденные в ней ошибки. Кроме того модель имеет:

- BooleanField field (test\_passed), которое используется для отображения пройден тест или нет (по умолчанию не пройден False).

- CharField field (file), которое используется для отображения имени программы, в которой обнаружена ошибка (оно ограничено 200 символами).

- CharField field (time), которое используется для отображения времени нахождения ошибки (оно ограничено 200 символами).

- IntegerField (version), которое используется для указания номера версии загруженной программы.

- CharField field (err\_text), которое используется для описания найденной ошибки (оно ограничено 1000 символами).

- CharField field (score), которое используется для отображения чистоты кода (оно ограничено 200 символами).

- IntegerField (count), которое используется для указания количества найденных ошибок.

Приложение Django admin может использовать модели для автоматического создания части сайта, предназначенной для создания, просмотра, обновления и удаления записей. Все необходимые настройки, которые необходимо включить в admin приложение вашего веб-сайта, были сделаны автоматически, когда был создан каркас проекта и для добавления модели в приложение admin необходимо только зарегистрировать их.

Файл /vkrlinter.ru/main/admin.py:

from django.contrib import admin  
from .models import Progs, Syntax  
# Register your models here.  
  
admin.site.register(Progs)  
admin.site.register(Syntax)

Этот код просто импортирует модели и затем вызывает admin.site.register для регистрации каждой из них.

Отображение является функцией, которая обрабатывает HTTP-запрос, получает данные из базы данных (при необходимости), которые применяются для генерации страницы HTML. Затем функция отображения возвращает сгенерированную страницу пользователю в виде HTTP-ответа. Отображения представлены в файле /vkrlinter.ru/main/views.py.

Главная страница получает информацию о том, вошел ли пользователь в систему, и если да, загружает из базы данных загруженные им ранее программы, и предоставляет доступ к загрузке новых:

def index(request):  
 user\_name = request.user.username  
 context = {**'prg\_names'**: valids\_progs(user\_name),  
 **'page\_flag'**: **''**,  
 **'logined'**: False,  
 **'prg\_data'**: **''**,  
 **'user\_name'**: user\_name,  
 **'dataset'**: {},  
 **'status'**: **''**}  
 if not request.user.is\_authenticated:  
 return render(request, **'main/index.html'**, context)  
 else:  
 context[**'logined'**] = True  
 return render(request, **'main/index.html'**, context)

Затем полученные записи передаются в шаблон страницы, генерирует страницу и передаёт её пользователю (клиенту пользователя, например браузеру). Если же пользователь не был ранее зарегистрирован в системе, его перенаправит на страницу регистрации.

Остановимся подробнее на функции поиска синтаксических ошибок.

def process\_syntax(request, prg\_name):  
 user\_name = request.user.username  
 cur\_prg = Progs.objects.get(filename=prg\_name)  
 prog\_id = cur\_prg.id  
 version = cur\_prg.version  
 cell\_dir = os.path.join(os.getcwd(), **'main'**, **'user\_files'**, user\_name, prg\_name)  
 s\_test\_result = syntax\_test\_2(cell\_dir, prog\_id, version)  
 print(s\_test\_result)  
 if s\_test\_result == -2:  
 messages.error(request, **"Your code have not .py files. Please retry do upload"**)  
 elif s\_test\_result == -1:  
 cur\_prg.status = **'test\_failed'** messages.error(request, **"Sory we can't test it, try reload file"**)  
 elif s\_test\_result >= 0:  
 if s\_test\_result <= 1.0:  
 cur\_prg.status = **'passed'** else:  
 cur\_prg.status = **'syntax\_errors'** cur\_prg.save()  
 return HttpResponseRedirect(**f'/prog/**{prg\_name}**'**)

Эта функция проверяет, загрузил ли пользователь файл с расширением .py, и меняет статус проверки программы на «пройден», если обнаружено не более одной синтаксической ошибки. Процесс поиска ошибок описан в файле /vkrlinter.ru/main/tasks.py и основан на применении метода pylint [5]:

def syntax\_test\_2(file\_path, prog\_id, version):  
 try:  
 if validation\_path(file\_path) == False:  
 return -2  
 else:  
 returns\_count = 0  
 files\_count = 0  
 for dirname, dirnames, filenames in os.walk(file\_path):  
 for filename in filenames:  
 if **'.py'** in filename and **'.pyc'** not in filename and **'\_\_init\_\_'** not in filename:  
 if **'venv'** not in dirname:  
 cur\_file = os.path.join(dirname, filename)  
 report\_items = {}  
 syntax\_err = []  
 (pylint\_stdout, pylint\_stderr) = lint.py\_run(command\_options=cur\_file, return\_std=True)  
 lints = pylint\_stdout.getvalue().split(**'**\n**'**)[1:]  
 for l in lints:  
 if **'warning'** in l:  
 err = l.split(**':'**)  
 syntax\_err.append(**f'line(**{err[1]}**) :** {err[2]} **'**)  
 if **'rated'** in l:  
 report\_items[**'code\_score'**] = l.split(**'('**)[0]  
 else:  
 continue  
 report\_items[**'syntax\_count'**] = str(len(syntax\_err))  
 report\_items[**'syntax\_errors'**] = **'**\n**'**.join(syntax\_err)  
 report\_items[**'time'**] = datetime.now().strftime(**"%d.%m.%Y-%H:%M:%S"**)  
 print(report\_items)  
 returns\_count += int(report\_items[**'syntax\_count'**])  
 test\_s = Syntax(time=report\_items[**'time'**],  
 file=filename,  
 version=version,  
 prog\_id=prog\_id,  
 err\_text=report\_items[**'syntax\_errors'**],  
 count=report\_items[**'syntax\_count'**],  
 score=report\_items[**'code\_score'**],  
 )  
 test\_s.save()  
 files\_count += 1  
 else:  
 continue  
 else:  
 continue  
 return returns\_count / files\_count  
 except:  
 return -1

Все взаимодействия между браузерами и серверами осуществляются при помощи протокола HTTP, который не сохраняет своё состояние (stateless). Данный факт означает, что сообщения между клиентом и сервером являются полностью независимыми один от другого — то есть не существует какого-либо представления «последовательности», или поведения в зависимости от предыдущих сообщений. В результате, если вы хотите создать сайт который будет отслеживать взаимодействие с клиентом (браузером), вам нужно реализовать это самостоятельно.

Сессии являются механизмом, который использует Django для отслеживания «состояния» между сайтом и каким-либо браузером. Сессии позволяют вам хранить произвольные данные браузера и получать их в тот момент, когда между данным браузером и сайтом устанавливается соединение. Данные получаются и сохраняются в сессии при помощи соответствующего «ключа».

Django использует куки (cookie), которые содержат специальный идентификатор сессии, который выделяет среди остальных, каждый браузер и соответствующую сессию. Реальные данные сессии, по умолчанию, хранятся в базе данных сайта (это более безопасно, чем сохранять данные в куки, где они могут быть уязвимы для злоумышленников).

Необходимые конфигурации выполняются в разделах INSTALLED\_APPS и MIDDLEWARE файла проекта (vkrlinter.ru/linter/settings.py), как показано ниже:

INSTALLED\_APPS = [  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**, # Фреймворк аутентификации и моделей по умолчанию.  
 **'django.contrib.contenttypes'**, # Django контент-типовая система (даёт разрешения, связанные с моделями).  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'main'**]

MIDDLEWARE = [  
 **'django.middleware.security.SecurityMiddleware'**,  
 **'django.contrib.sessions.middleware.SessionMiddleware'**, # Управление сессиями между запросами  
 **'django.middleware.common.CommonMiddleware'**,  
 **'django.middleware.csrf.CsrfViewMiddleware'**,  
 **'django.contrib.auth.middleware.AuthenticationMiddleware'**, # Связывает пользователей, использующих сессии, запросами.  
 **'django.contrib.messages.middleware.MessageMiddleware'**,  
 **'django.middleware.clickjacking.XFrameOptionsMiddleware'**,  
]

Django предоставляет систему аутентификации и авторизации («permission») пользователя, реализованную на основе фреймворка работы с сессиями. Система аутентификации и авторизации позволяет проверять учётные данные пользователей.

**2.2.2 Библиотека шаблонов CSS – Bootstrap**

Шаблон это текстовый файл, который определяет структуру и расположение данных в файле, кроме того, в нем размещают специальные метки (placeholders), которые используются для показа реального содержимого, то есть данных. По умолчанию Django ищет файлы шаблонов в директории с именем 'templates' внутри приложения.

Для отображения шаблонов используется библиотека шаблонов CSS – Bootstrap. «Стиль и макет Bootstrap можно применять к широкому спектру структур разметки. Эта документация направлена на то, чтобы предоставить разработчикам примеры передового опыта, чтобы продемонстрировать использование самого Bootstrap и проиллюстрировать соответствующую семантическую разметку, включая способы решения потенциальных проблем доступности». [1]. В качестве шаблона использовалась базовая панель администратора с фиксированной боковой панелью и навигационной панелью, представленная на рисунке 2.2.3.

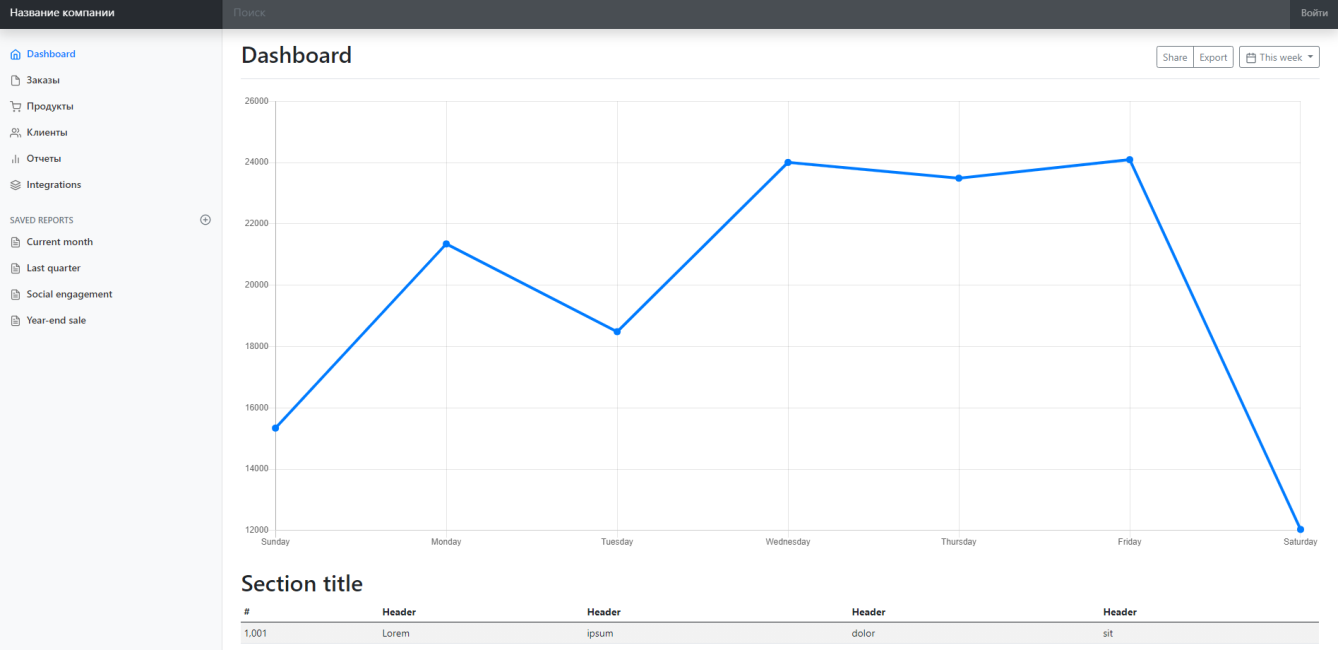


Рисунок 2.2.3 Шаблон панели администратора

Шаблон дизайна для формы входа представлен на рисунке 2.2.4.

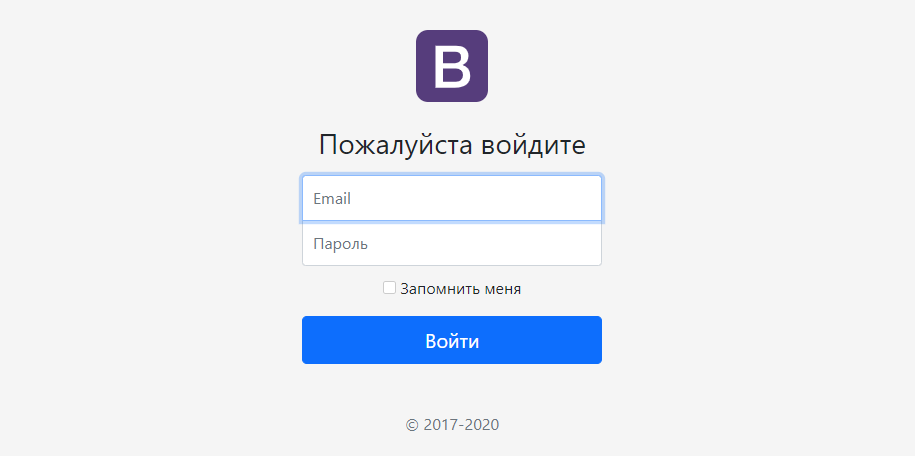


Рисунок 2.2.4 Шаблон формы входа

Кроме того, Bootstrap представляет библиотеку иконок с открытым исходным кодом, примеры которой представлены на рисунке 2.2.5.

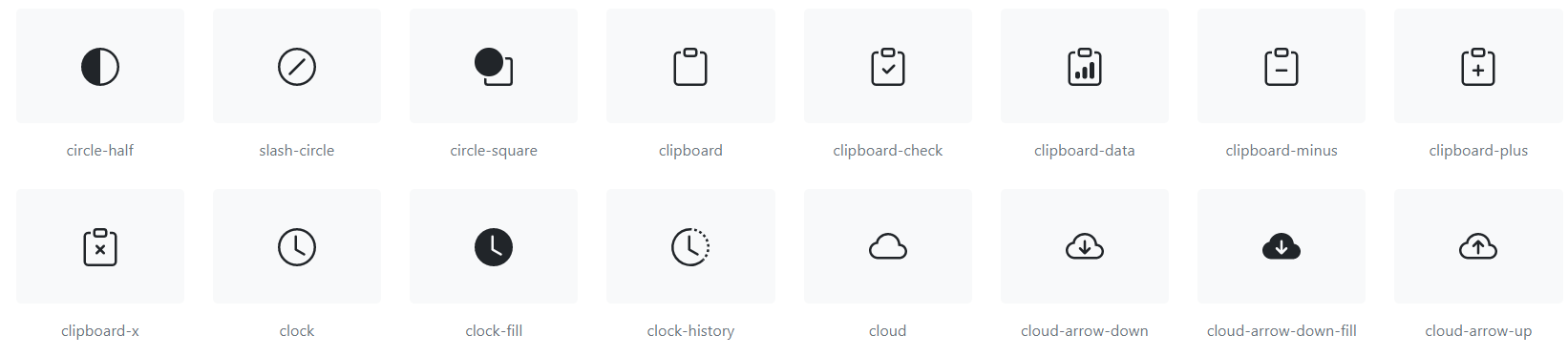


Рисунок 2.2.5 Библиотека иконок

**2.2.3 Описание программных модулей**

При запуске системы пользователю становится доступна главная страница (рис. 2.2.6), на которой отображается краткое руководство пользователя и кнопка логина/регистрации.

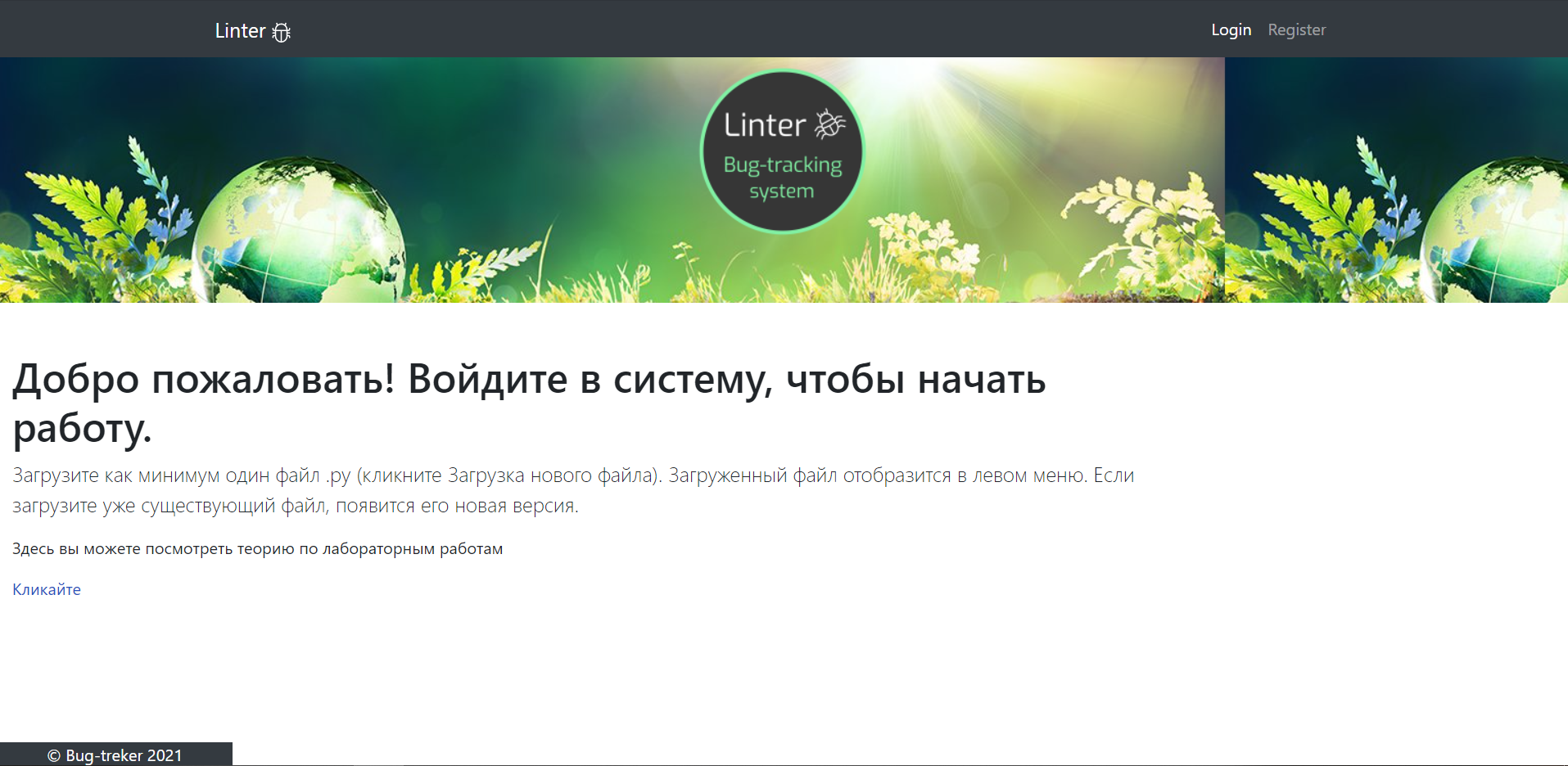


Рисунок 2.2.6 Главная страница

Логотип был разработан с помощью сайта FreeLogoDesign [3] и представлен на рисунке 2.2.7.

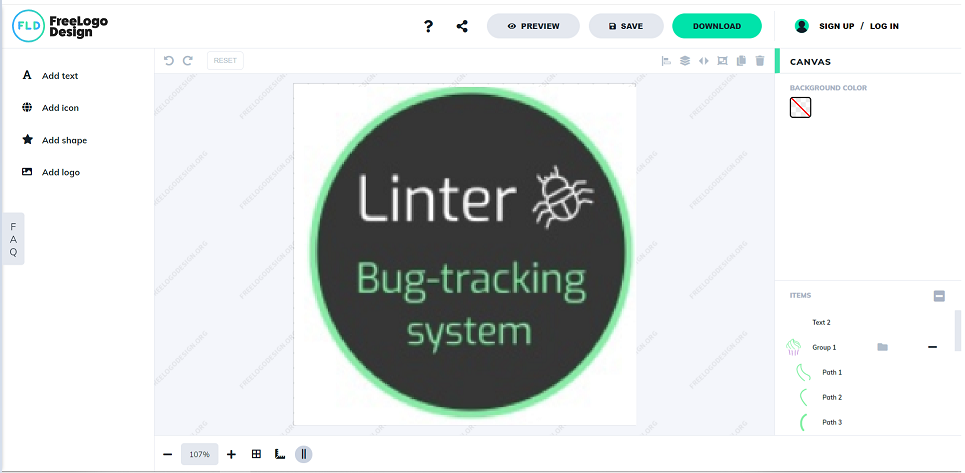
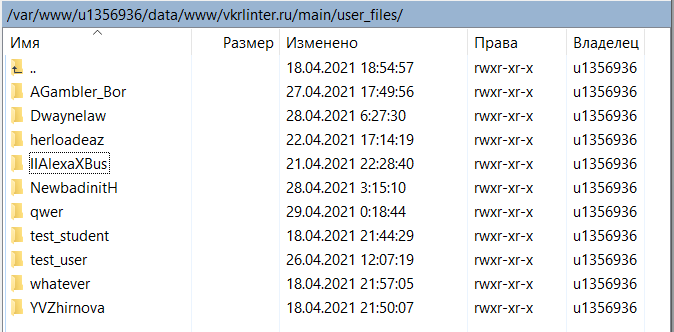


Рисунок 2.2.7 Логотип

Для дальнейшей работы с системой необходимо авторизоваться, что создаст таблицу пользователя в базе данных и позволит ему иметь доступ к загруженным файлам/проектам. Для регистрации необходимо ввести имя, почту и пароль (рис. 2.2.8, а), после этого пользователя перенаправит на страницу логина, где будет необходимо повторить имя пользователя и пароль (рис. 2.2.8, б)

|  |  |
| --- | --- |
| а) Регистрация | б) Логин |
| Рисунок 2.2.8 Авторизация пользователя | |

Зарегистрированных пользователей и загруженные файлы можно просмотреть в папке user\_files. (рис.2.2.9)



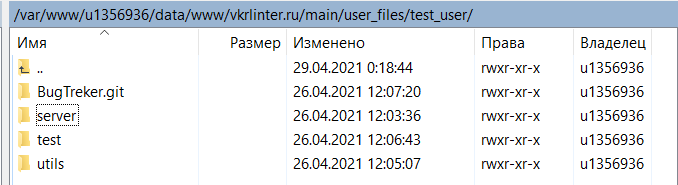


Рисунок 2.2.9 Примеры пользователей

После авторизации у пользователей появляется боковое меню с возможностью открыть страницу загрузки файла (рис. 2.2.10).

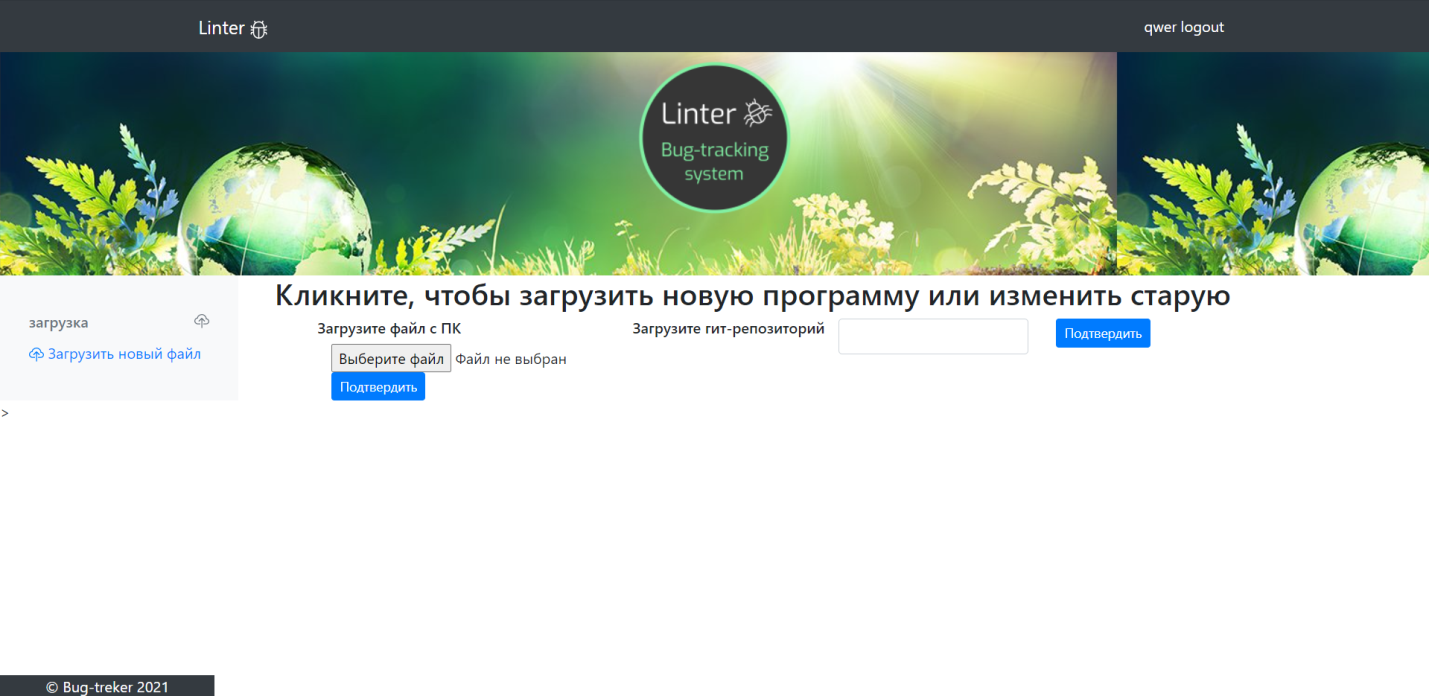
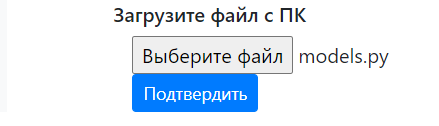
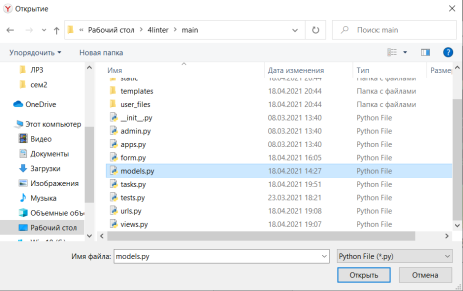


Рисунок 2.2.10 Страница загрузки файла

Реализована возможность загрузки файла с расширением ,py с ПК и клонирование файлов с гит-репозитория. При загрузке файла открывается окно загрузки файла с ПК, название выбранного файла отображается на странице, и если пользователя все устраивает, он нажимает кнопку «Подтвердить» и имя файла отображается в боковом меню, происходит автоматическое перенаправление на страницу тестирования файла. Пока пользователь не запустит тест отображается значок not runned (рис. 2.2.11)



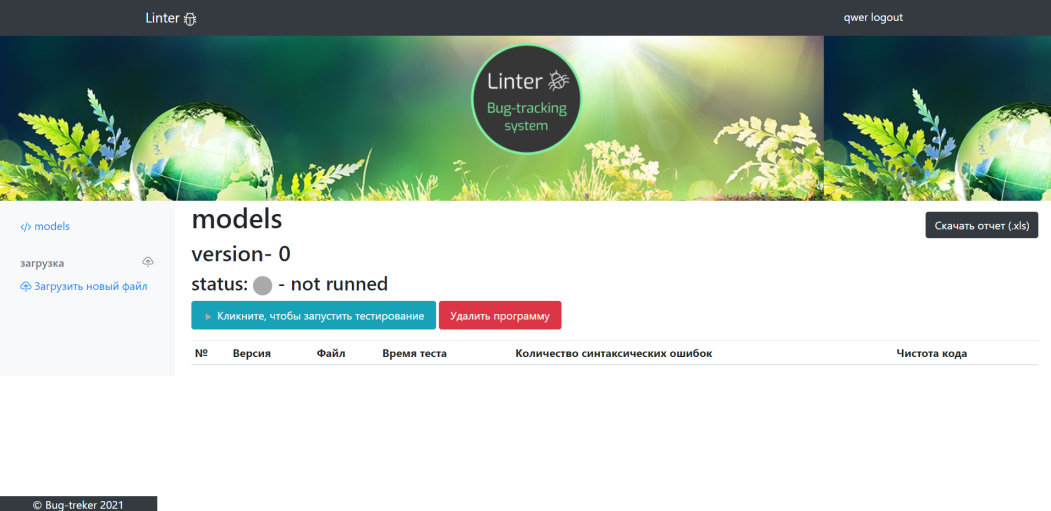
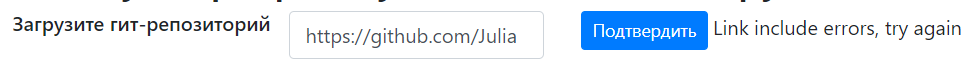
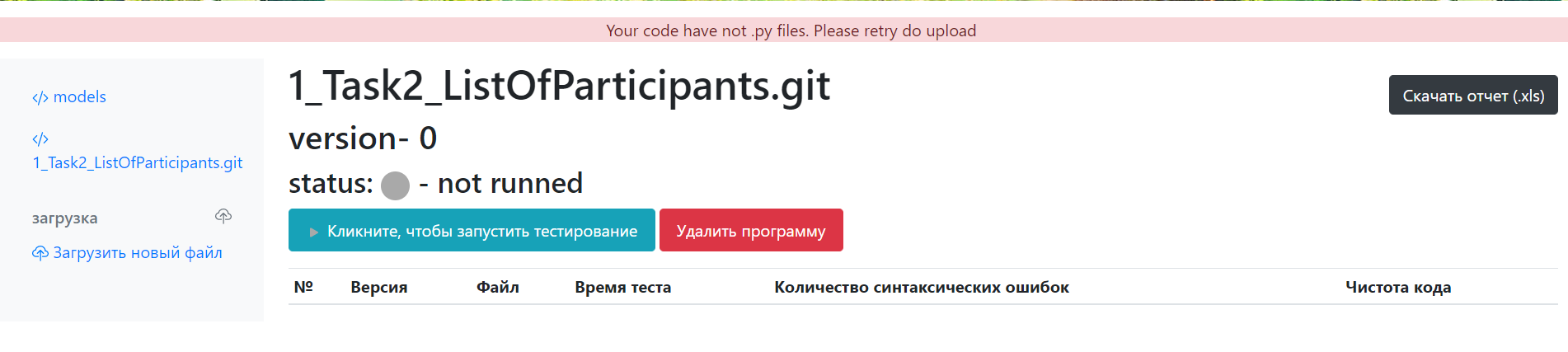


Рисунок 2.2.11 Пример загрузки файла с компьютера

При вводе ссылки на гит-репозиторий возможно, что пользователь введет неверную ссылку, это может произойти, если указан не существующий репозиторий, в этом случае отобразится ошибка «Link include errors, try again».(рис. 2.2.12 а) Или гит-репозиторий существует, но в нем отсутствуют файлы с расширением .py, в этом случае отобразится ошибка «Your code have not .py files. Please retry do upload». (рис. 2.2.12 б)



а) Неверная ссылка



б) Гит-репозиторий не содержит файлы с расширением .py

Рисунок 2.2.12 Возможные ошибки при загрузке гит-репозитория

Чтобы запустить тестирование необходимо в левом меню выбрать имя файла/проекта, отобразится страница тестирования. Для начала тестирования необходимо нажать кнопку «Кликните, чтобы начать тестирование». После этого обновится статус на syntax errors (если более 1 ошибки) или passed (если менее 1 ошибки) и отобразится версия программы. При выборе версии отображается количество ошибок и время теста (рис. 2.2.13 - 2.2.14). Если больше не нужна информация о данной программе ее можно удалить, нажав на кнопку «Удалить программу».

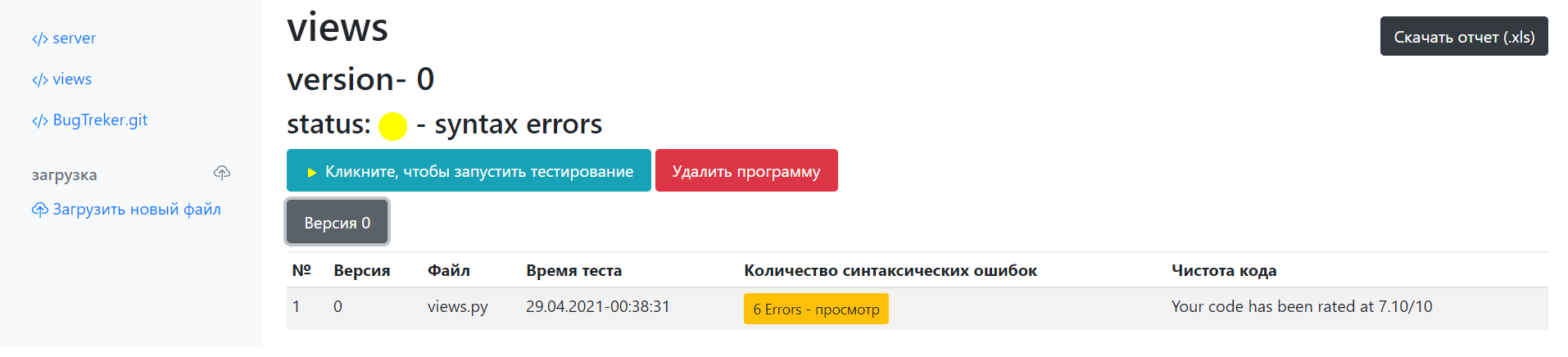


Рис. 2.2.13 Пример теста с ошибками

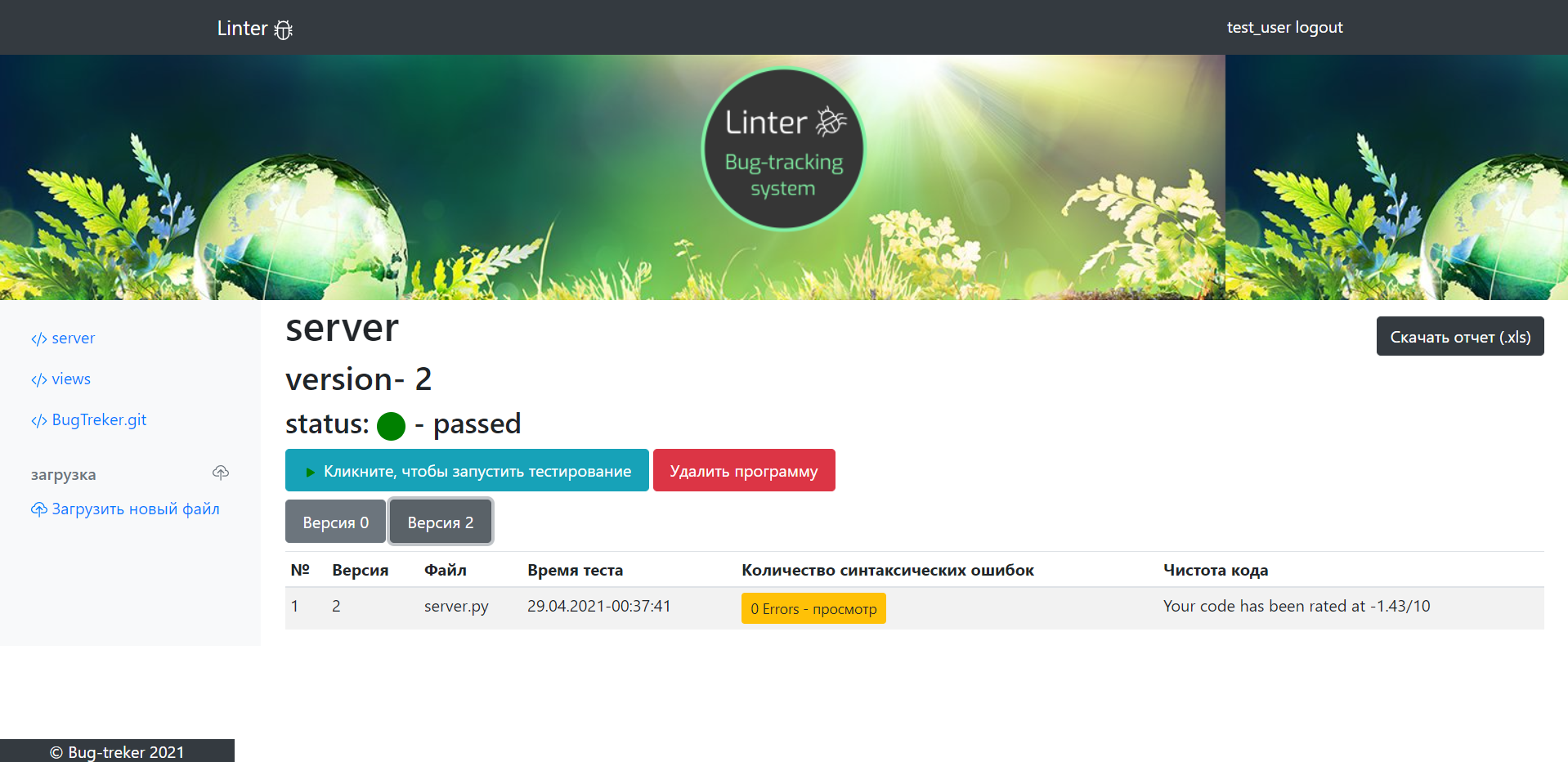


Рис. 2.2.14 Пример теста без ошибок

При загрузке проекта с репозитория отобразятся все найденные файлы с расширением .py и количество найденных ошибок. Как видно на рисунке 2.2.15 загрузка нескольких файлов занимает больше времени, к примеру, проверка 13 файлов заняла 20 секунд.

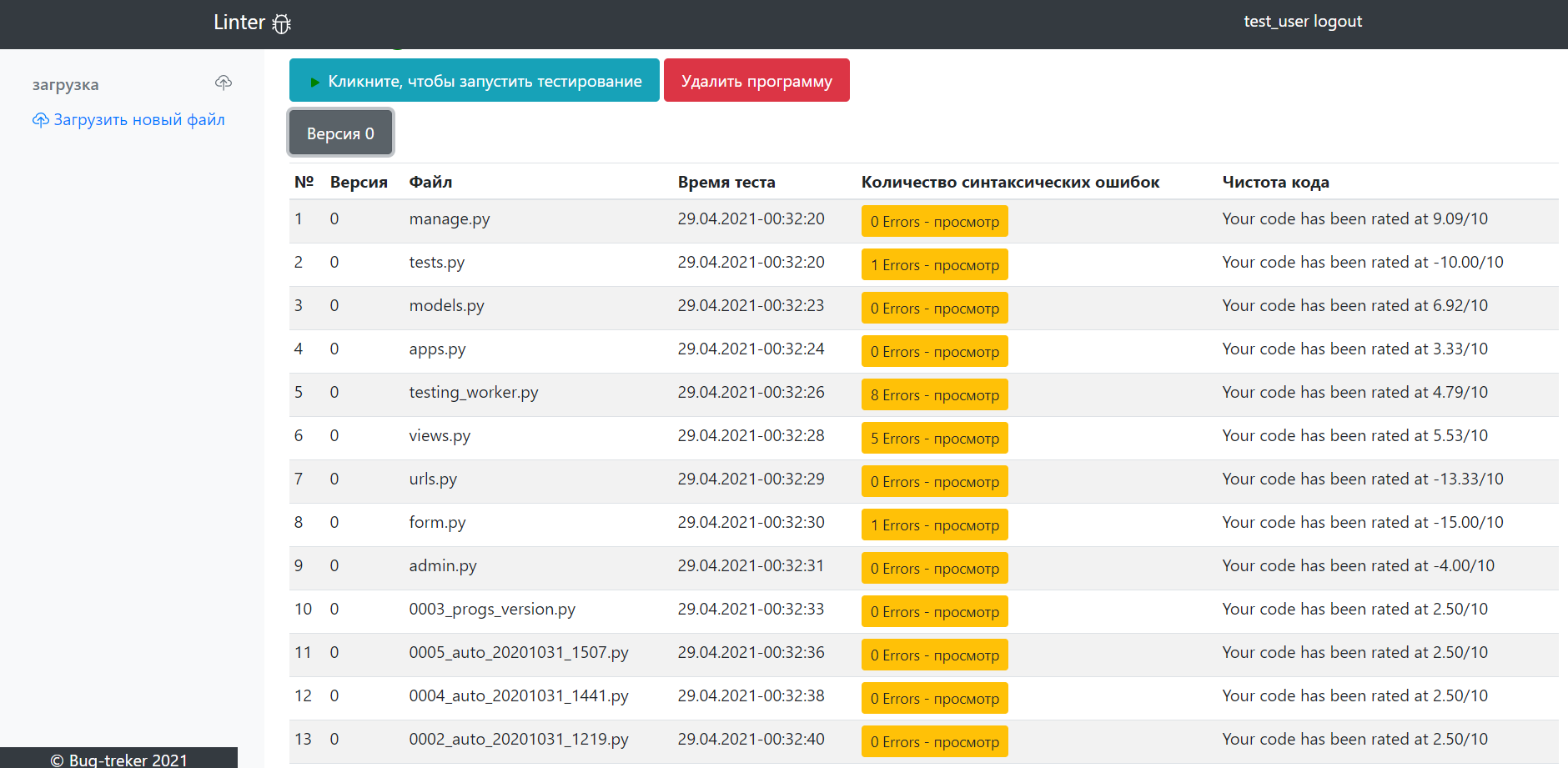


Рис. 2.2.15 Пример проверки проекта склонированного с репозитория

Для просмотра подробного описания текста ошибки можно нажать на кнопку «Просмотр» и отобразится страница просмотра ошибки (рис. 2.2.16).

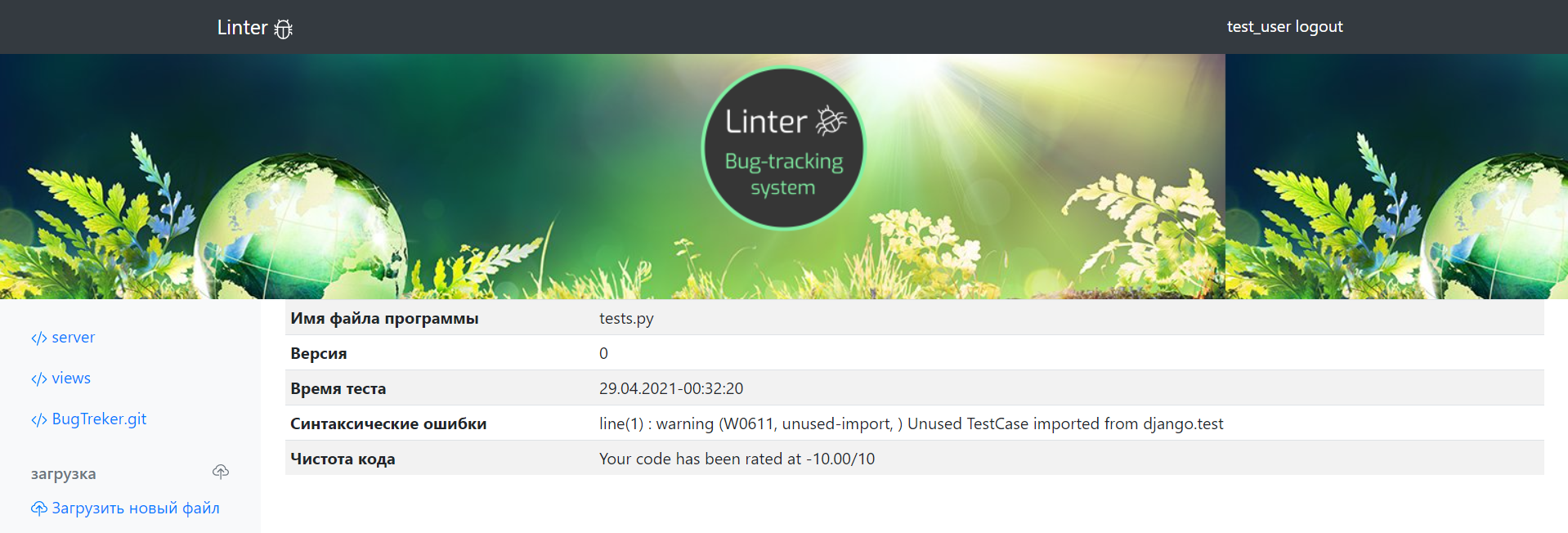


Рис. 2.2.16 Страница просмотра ошибки

Чтобы получить отчет о проведенных тестах необходимо нажать на кнопку «Скачать отчет». На компьютер загрузится файл формата .xls. Его содержание представлено на рисунке 2.17

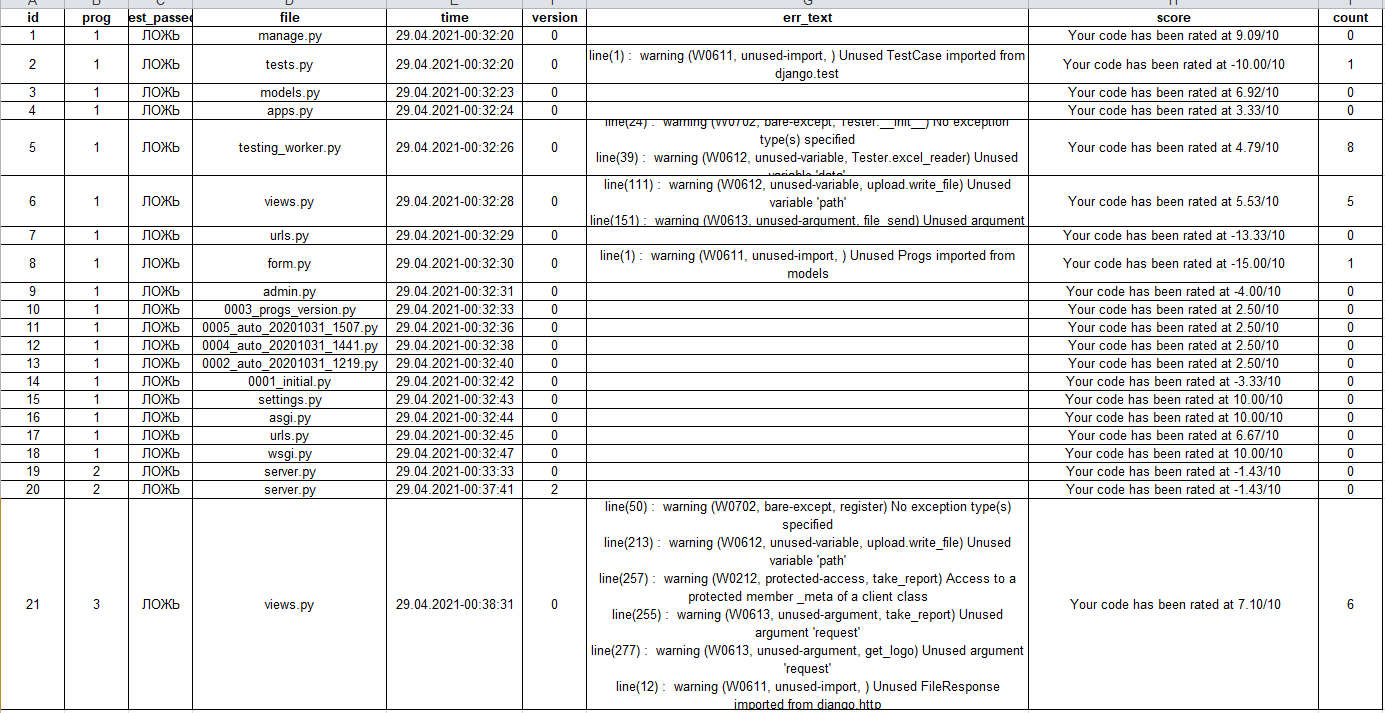


Рис. 2.17 Отчет о проведенных тестах

**2.3 Размещение веб-портала на хостинге**

Для размещения веб-портала на хостинге на сайте reg.ru был приобретен домен vkrlinter.ru и хостинг Host-A, в котором есть поддержка Python и который позволяет выложить 1 сайт (рис. 2.3.1). Кроме этого в комплекте шел SSl сертификат. [10]

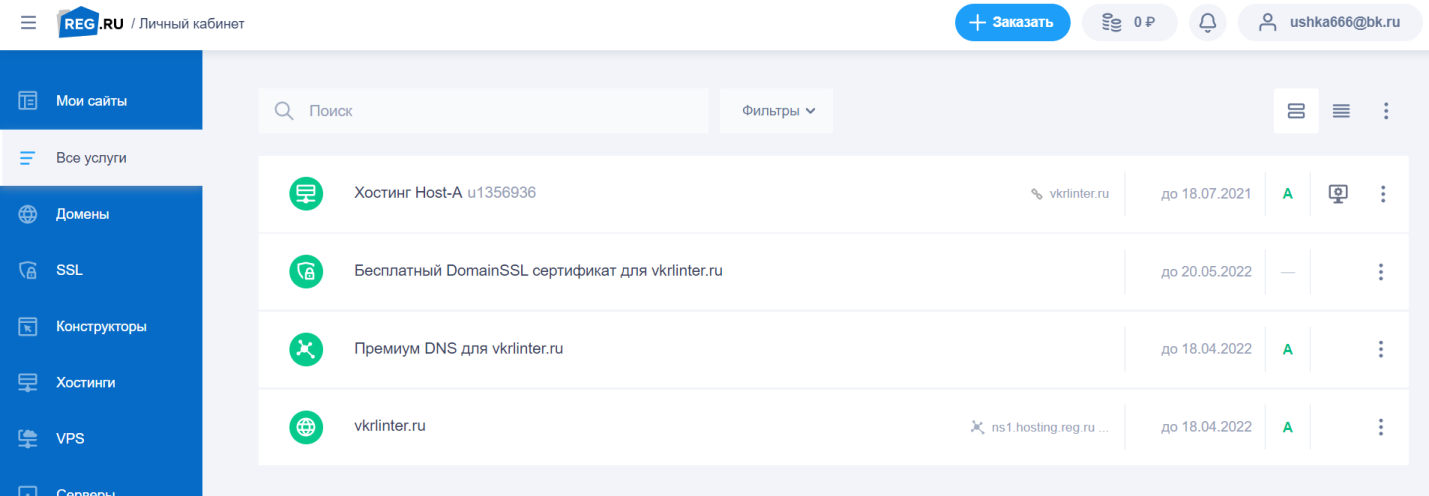
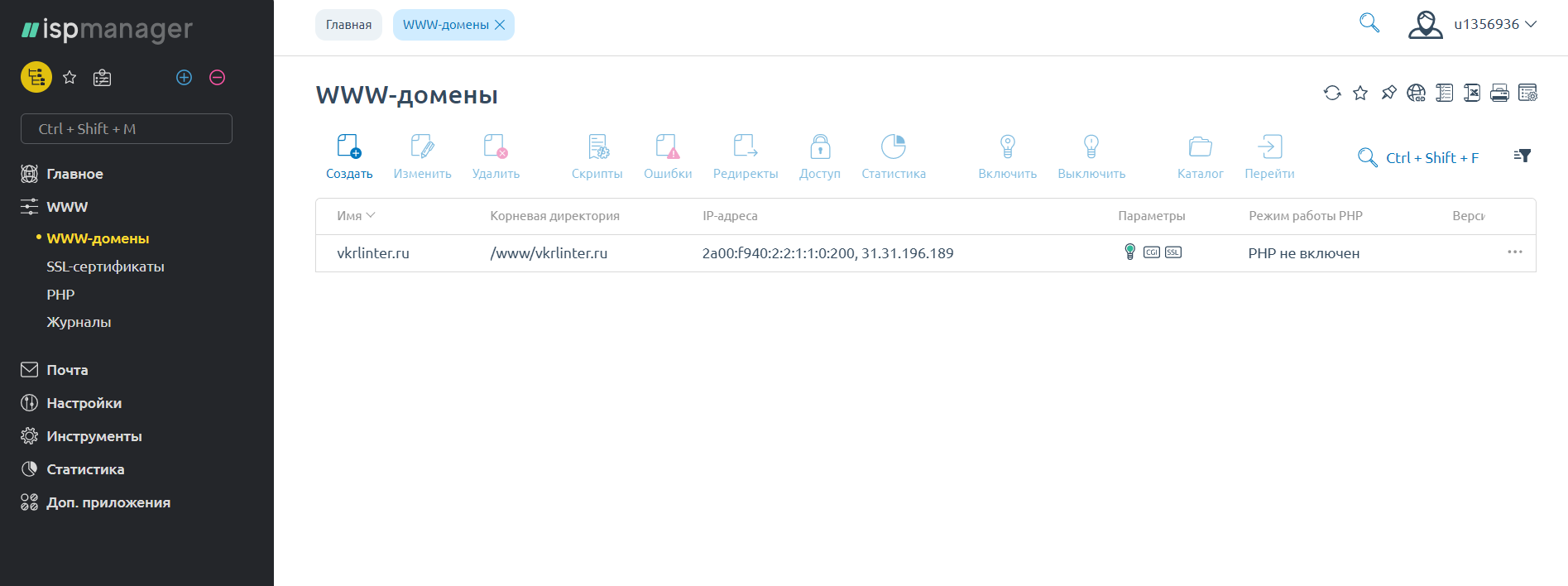


Рис. 2.3.1 Приобретенный домен и хостинг

Для установки Django на хостинг использовалась программа ispmanager (в ней выбирается домен) и WinSCP (создается виртуальное окружение) (рис.2.3.2). [14]



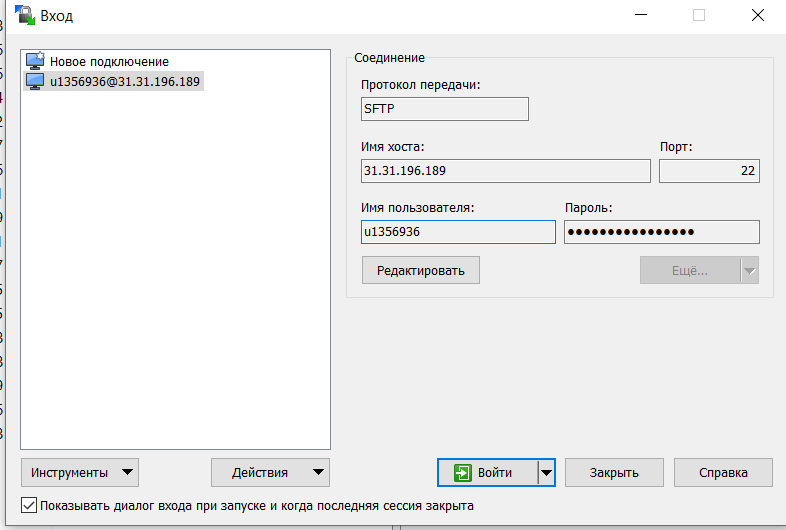


Рис. 2.3.2 Создание подключения

После выкладывания проекта на хостинг он становится доступен по адресу: <https://vkrlinter.ru/>

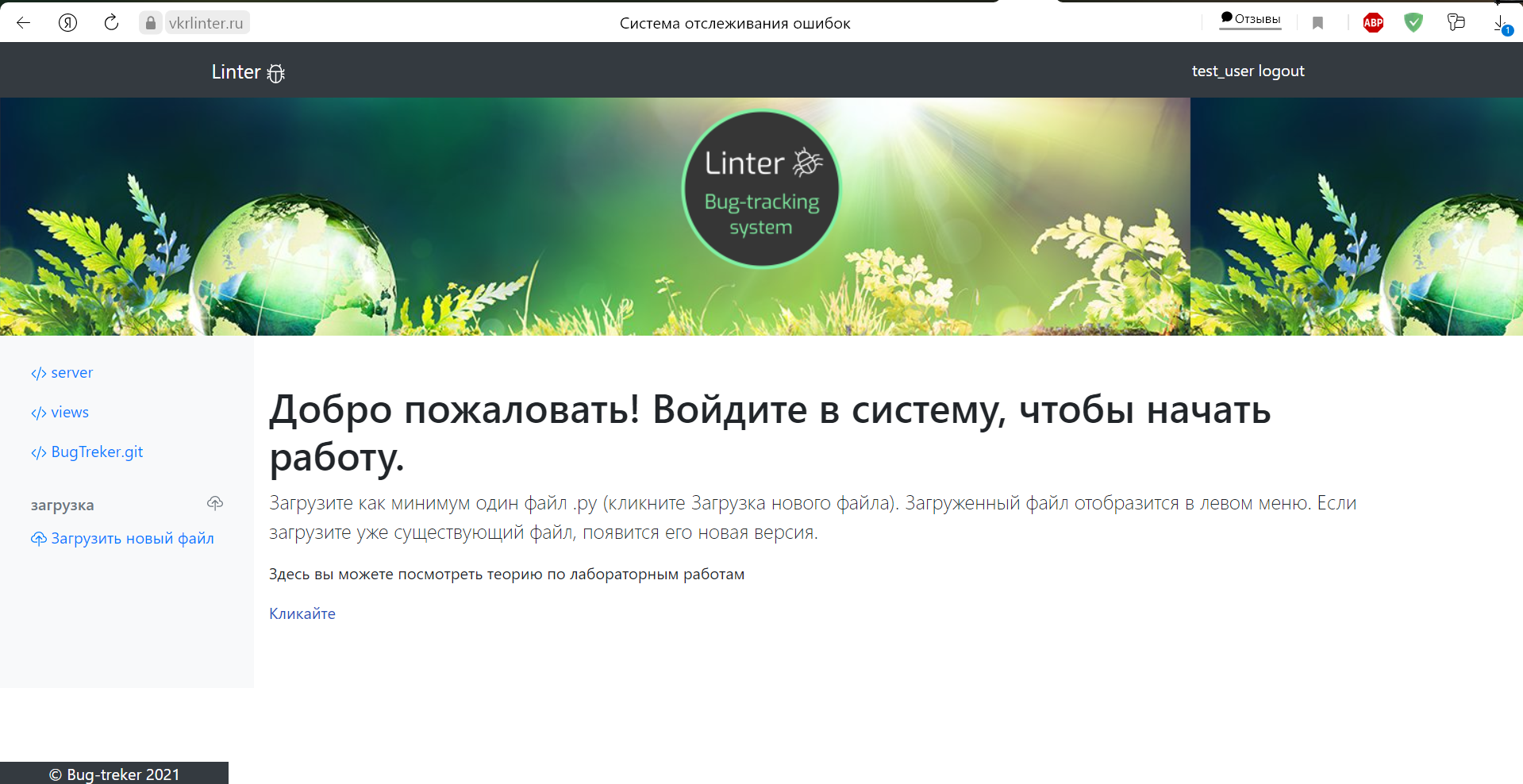


Рис. 2.3.3 Сайт выложен в интернет

**2.4 Тестирование**

После выкладывания системы в интернет, можно провести ее тестирование. Так, для мониторинга трафика, можно использовать сайт HOST-TRACKER website monitoring service. Результаты проверки приведены на рисунке 2.4.1

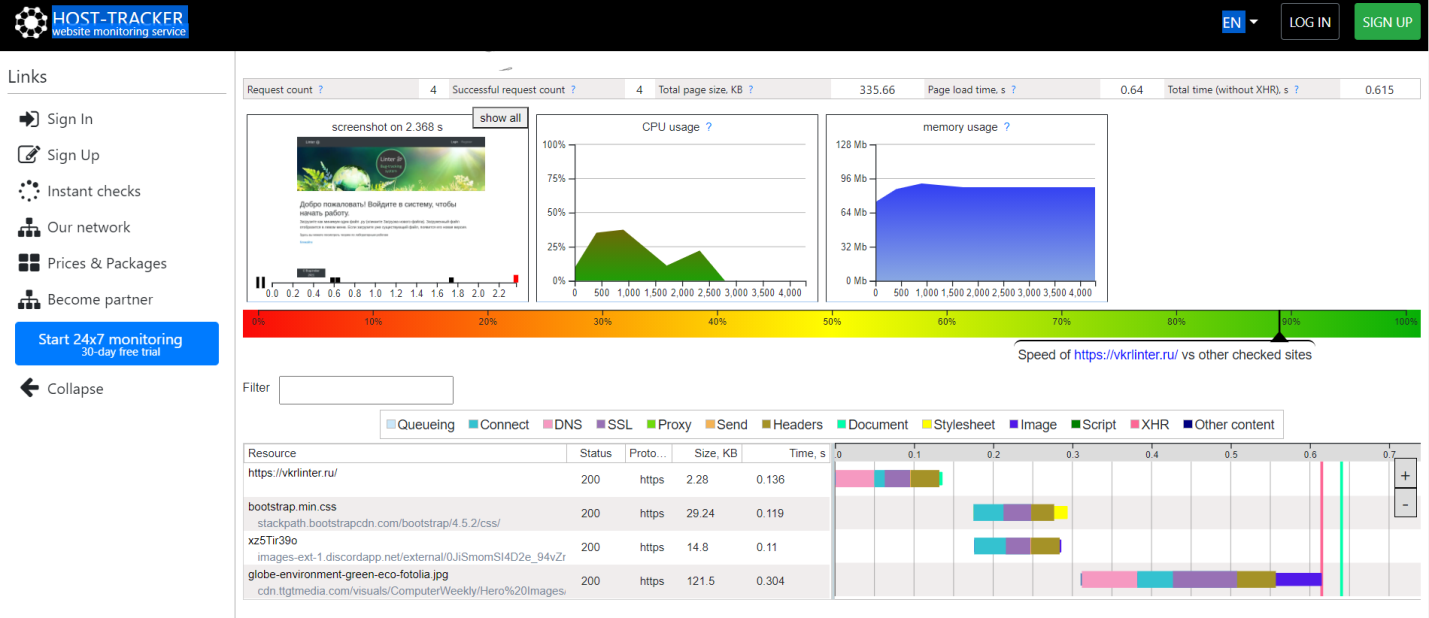


Рис.2.4.1 Мониторинг трафика

Для проверки структуры сайта, наличия ошибок в коде, можно использовать сайт wave.webaim.org. Результаты проверки приведены на рисунке 2.4.2

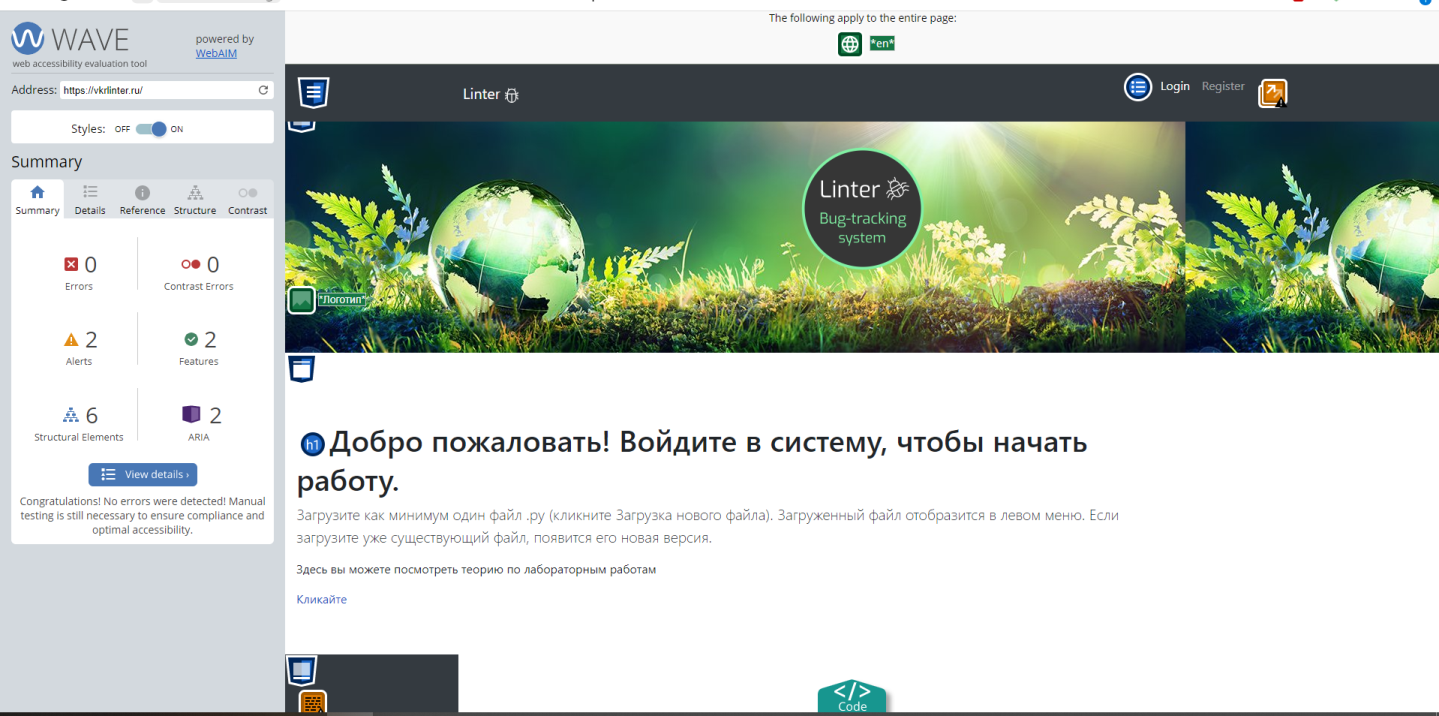


Рис.2.4.2 Проверка структуры

Для проведения нагрузочного тестирования использовался сайт loaddy.com. Результаты проверки приведены на рисунке 2.4.3.

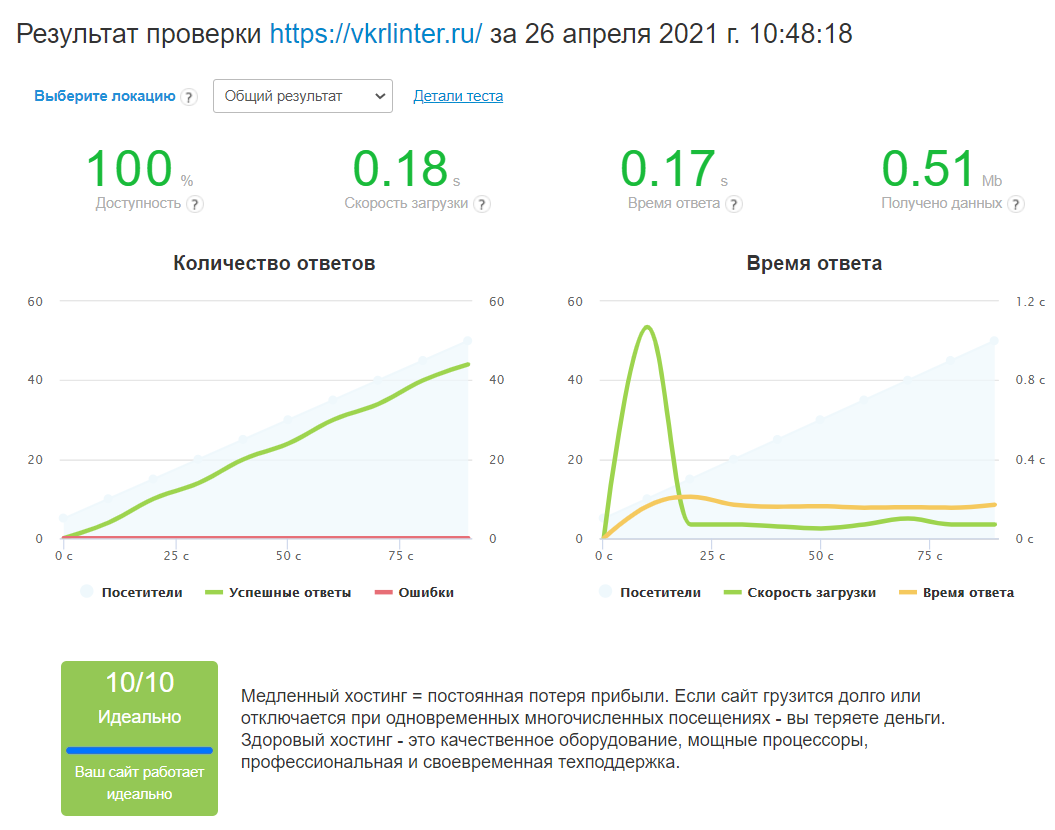
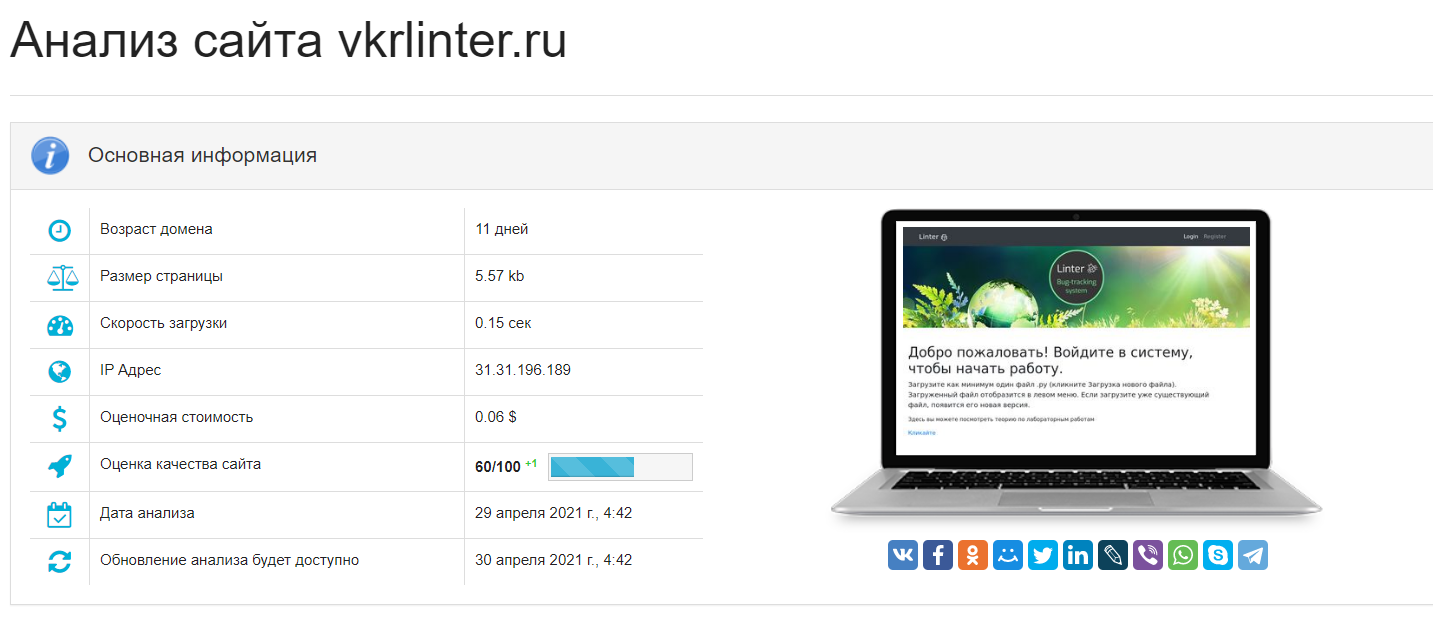
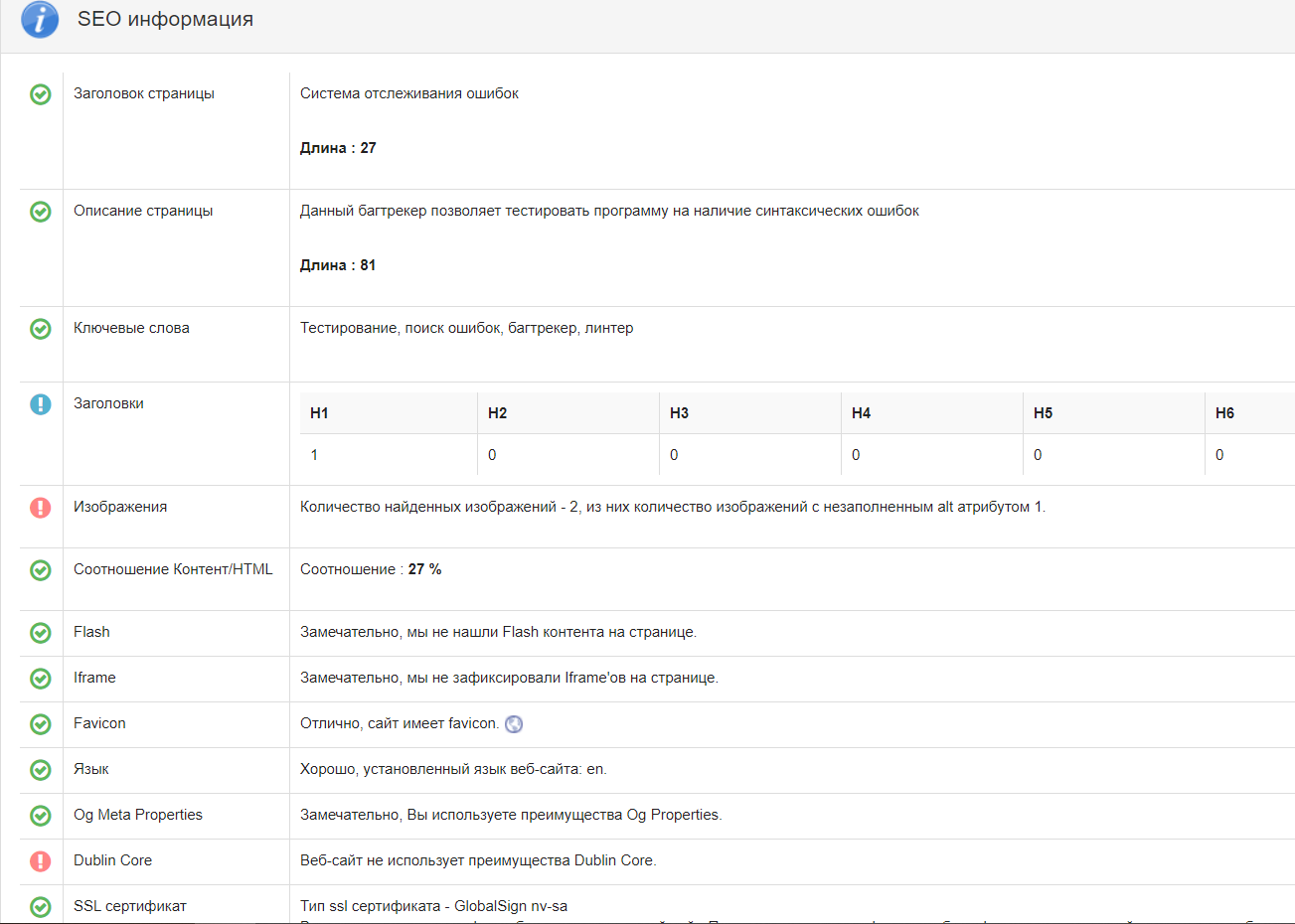
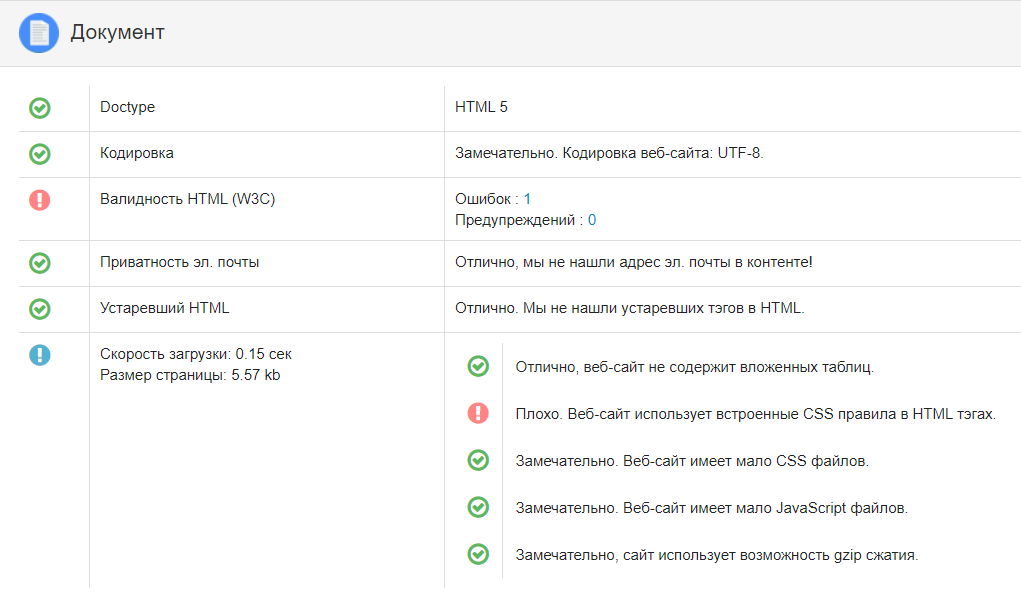


Рис.2.4.3 Нагрузочное тестирование

SEO-анализ текста — проверка на соответствие (предполагаемым) требованиям поисковых систем и, как правило, оценка с точки зрения читаемости. Позволяет проверить, как отображается сайт на различных устройствах. Он был проведен с помощью сайта parsesite.ru. Результаты представлены на рисунке 2.4.4.







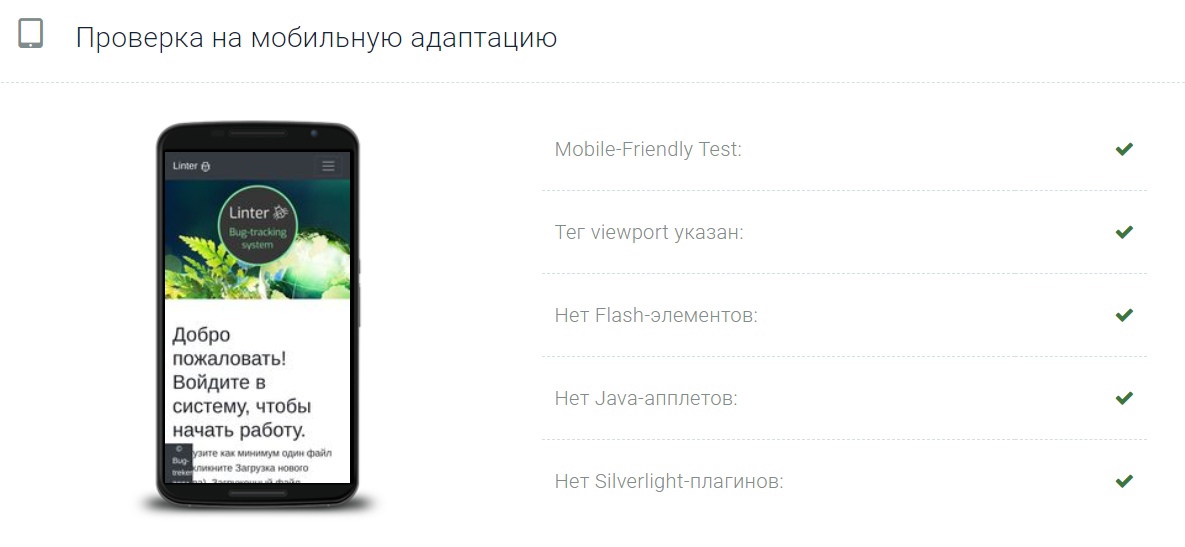


Рис.2.4.4 SEO-анализ

**Заключение**

Как уже было сказано ранее багтрекеры — это системы, обеспечивающие учёт и контроль ошибок, возникающих в процессе создания программного обеспечения и при его применении. Сервисы работают как с проблемами, обнаруженными разработчиками во время тестирований, так и с запросами пользователей. Главной функцией багтрекера является создание базы данных ошибок, содержащей подробные сведения о проблеме (версия продукта, раздел, дата и время обнаружения проблемы, ответственное лицо), а также приоритет и дедлайн. В отличие от обычного таск-менеджера системы отслеживания ошибок ориентированы на решение конкретных технических задач и сохранение данных о проведённых тестах для использования при необходимости их в будущем.

*Целью* бакалаврской работы являлась разработка автоматизированной системы отслеживания ошибок при разработке программного обеспечения.

Для достижения поставленной цели были решены следующие *задачи*:

− проведен анализ предметной области с целью выявления процессов учета, требующих автоматизации;

− проанализированы существующие программные средства и технологии для отслеживания ошибок;

− описаны функциональные требования к проектируемой системе;

− построена модель данных на логическом и физическом уровнях;

− разработана автоматизированную систему (АС) и описать принципы работы с ней;

- проведено тестирование разработанной системы.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Bootstrap: библиотека шаблонов CSS [Электронный ресурс] // URL: <https://bootstrap-4.ru/docs/4.5.2/getting-started/introduction/> (дата обращения: 10.03.2021)
2. Code review [Электронный ресурс] // URL: <https://habr.com/ru/post/340550/> (дата обращения 06.02.2021)
3. FreeLogoDesign [Электронный ресурс] // URL: <https://www.freelogodesign.org/> (дата обращения: 20.03.2021)
4. PEP 8 - руководство по написанию кода на Python [Электронный ресурс] // URL: <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html> (дата обращения: 05.02.2021)
5. Pylint изнутри. Как он это делает [Электронный ресурс] // URL: <https://habr.com/ru/company/oleg-bunin/blog/433474/> (дата обращения: 05.02.2021).
6. TOP 5 фреймворков Python для Web-разработки в 2020 году [Электронный ресурс] // URL: <https://waksoft.susu.ru/2020/01/14/top-5-frejmvorkov-python-dlya-web-razrabotki-v-2020-godu/> (дата обращения: 21.02.2021)
7. ГОСТ 34.601-90 Автоматизированные системы. Стадии создания: URL: <https://www.prj-exp.ru/gost/gost_34-601-90.php>
8. ГОСТ 7.32-2017 ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ. Структура и правила оформления [Электронный ресурс] // URL: <https://allgosts.ru/01/140/gost_7.32-2017.pdf>
9. Изучение веб-разработки. Серверное программирование веб-сайтов. Веб-фреймворк Django (Python). [Электронный ресурс] // URL: <https://developer.mozilla.org/ru/docs/Learn/Server-side/Django> (дата обращения: 23.02.2021)
10. Как установить Django на хостинг [Электронный ресурс] // URL: <https://www.reg.ru/support/hosting-i-servery/yazyki-programmirovaniya-i-skripty/kak-ustanovit-django-na-hosting> (дата обращения: 10.04.2021)
11. Мир Python: исключения. [Электронный ресурс] // URL: <https://ru.hexlet.io/courses/advanced_python/lessons/python_exceptions/theory_unit> (дата обращения: 05.02.2021)
12. Обзор популярных систем bug-трэкинговых систем. [Электронный ресурс] // URL: <https://training.qatestlab.com/blog/helpful-materials/overview-of-popular-bug-tracking-systems/> (дата обращения: 07.02.2021)
13. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ./Сэм Канер, Джек Фолк, Енг Кек Нгуен. — К.: Издательство «ДиаСофт», 2001. — 544 с.
14. Хостинг. PHP, ASP.NET и скрипты. Как установить Django на хостинг. [Электронный ресурс] // URL: <https://www.reg.ru/support/hosting-i-servery/yazyki-programmirovaniya-i-skripty/kak-ustanovit-django-na-hosting> (дата обращения: 10.04.2021)
15. Язык программирования Python: особенности и преимущества. [Электронный ресурс] // URL: <https://ipap.ru/poleznye-stati/4-useful/yazyk-programmirovaniya-python-osobennosti-i-preimushchestva> (дата обращения: 23.01.2021)