

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРОГРАММНОЕ ПРИЛОЖЕНИЕ

ВАЛИДАЦИИ ОБЪЕКТОВ

Студент

Клим 31.05.22 Климин Владислав Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель

Дуб 31.05.22 А.В. Дубравин
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

Быч 31.05.22 А.С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой

М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

21 июня 2022 № 18)

Секретарь ГЭК

Е.И. Гурий
(подпись) (фамилия, инициалы)

Пенза, 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«30» апреля 2020 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Климин Владислав Алексеевич, гр.18ВВ2 факультета вычислительной техники.

2. Тема работы: «Программное приложение валидации объектов»

Тема утверждена приказом ПГУ №357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Дубравин А.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать программное приложение валидации объектов, осуществляющее сбор информации объекта и прохождение валидации. Приложение должно быть спроектировано с применением клиент-серверной архитектуры. Клиентами должны выступать браузеры пользователей, которые получают заполненные html-шаблоны. Система должна включать в себя функционал сбора и сохранения данных в базу данных и их удаление оттуда. Приложение должно осуществлять валидацию объектов по установленным требованиям.

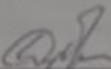
5. Перечень подлежащих разработке вопросов

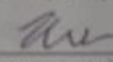
Требуется произвести анализ предметных областей: изучить аналоги существующих систем валидации, изучить основы разработки приложений, изучить способы проектирования клиент-серверного приложения. Также необходимо изучить функционал подходящих фреймворков клиент-серверных приложений.

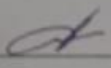
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	50	26.04.2022 – 30.04.2022	
2.	Постановка задачи	10	01.05.2022 – 02.05.2022	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2022 – 07.05.2022	
4.	Описание алгоритмов	40	08.05.2022 – 11.05.2022	
5.	Разработка программного приложения валидации объектов	150	12.05.2022 – 29.05.2022	
6.	Оформление пояснительной записки	24	30.05.2022 – 31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  26.04.22 Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент  26.04.22 Климин В.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  17.06.2022 Фионова Л.Р.

Реферат

Работа содержит 81 страницу формата А4, 25 рисунков, 2 таблицы, 6 источников, 3 приложения.

БАЗА ДАННЫХ, ВАЛИДАЦИЯ, ВЕБ-ИНТЕРФЕЙС, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА.

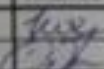
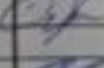
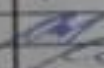
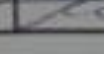
Объект разработки – Программное приложение валидации объектов.

Целью дипломной работы – разработать систему, позволяющую осуществлять валидацию объектов.

В результате проделанной работы разработано приложение, которое полностью соответствует поставленной задаче, разработана серверная часть приложения, реализованы шаблоны клиентской части.

В пояснительной записке приведен анализ предметной области, обзор технологии разработки и отладки программы.

Графическая часть содержит *UML*-диаграммы приложения.

					ПГУ 1.090301.16.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Создан		Кликин В.А.			Программное приложение валидации объектов Пояснительная записка	Лист	Листов
Провер.		Добровин А.В.				2	81
И. Контр.		Белкин А.С.			ФВТ, каф. ВТ гр. 18ВВ2		
Утверд.		Митрохин М.А.					

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1. Программное приложение валидации объектов	7
1.1 Анализ предметной области	7
1.1.1 Обзор существующих аналогов.....	8
1.1.2 Постановка задачи.....	9
1.1.3 Выбор языка программирования	Error! Bookmark not defined.
1.1.4 Выбор библиотеки для разработки серверного приложения	Error!
Bookmark not defined.	
1.1.5 Выбор СУБД.....	Error! Bookmark not defined.
1.2 Проектирование и разработка	16
1.2.1 Реализация БД.	17
1.2.2 Реализация хранения файлов и изображений.	20
1.2.3 Реализация серверной части	Error! Bookmark not defined.
1.2.4 Реализация шаблонов клиентской части	Error! Bookmark not defined.
1.3 Тестирование и отладка	25
1.3.1 Функционал веб-приложения.	28
1.3.2 Функционал панели администратора.....	35
2 Безопасность жизнедеятельности.....	39
2.1 Анализ опасных и вредных факторов, воздействующих на программиста при разработке приложения.....	39
2.2 Микроклимат рабочей зоны программиста	41
2.3 Окраска и коэффициенты отражения	42
2.4 Освещение рабочего места	43
2.5 Воздействие шума на программиста. Защита от шума.....	45
Заключение.	47
Список использованных источников.	48

Приложение А – Листинг программы.....	49
Приложение Б – <i>UML</i> -диаграммы.....	68
Приложение В – Презентация.....	73

					ПГУ 1.090301.16.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс приложения

CSS (Cascading Style Sheets) – формальный язык описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки

HTML (Hyper Text Markup Language) – язык гипертекстовой разметки

ID (data name, identifier) – опознаватель

MVC (Model-View-Controller) – «Модель-Представление-Контроллер» или «Модель-Вид-Контроллер»

ORM (Object-Relational Mapping) – объектно-реляционное отображение или преобразование

SQL (Structured query language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

URL (Uniform Resource Locator) – унифицированный указатель ресурса

БД – база данных

ВКР – выпускная квалификационная работа

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базами данных

Введение

Количество пользователей увеличилось во много раз с момента появления Интернета. Основная масса населения имеет у себя практически в любой точке мира выход к его ресурсам. В связи с этим, есть необходимость использования этого направления в информационных целях. С помощью Интернета можно фокусировать воздействие на определенную целевую аудиторию, где могут быть заинтересованы разные организации.

Неверные данные, не соответствующие определенным требованиям, часто приводят к сбою программы. Предположим, в программе появляется исключение при попытке какого-либо действия, которое нарушает требование валидации. Если эта ошибка так и не перехватилась, она, скорее всего, вызовет аварийное завершение программы. Конечно, это маловероятно. Скорее всего, пользователю выведется сообщение об ошибке в программе, далее программа попытается восстановиться.

Отсутствие проверки может вызвать всевозможные трудности. Наличие валидации предотвращает критические проблемы, но сразу возникает проблема с производительностью, так как большое количество проходов дают большую нагрузку на систему.

Актуальность выбранной темы обусловлена потребностью в возможности проверки объектов на установленные требования в целях повышения качества объекта и достоверности в том, что продуктом можно пользоваться по назначению.

Цель работы – разработка программного приложения для проверки объектов на установленные требования.

Необходимо разработать веб-приложение на основе МРА шаблона, модели, отвечающие за хранение файлов и данных объекта и реализовать функции для обработки конечных точек.

1 Программное приложение валидации объектов

1.1 Анализ предметной области

Валидация — это проверка того, что процесс, продукт полностью удовлетворяет заданным требованиям заказчика. К примеру, человек захотел купить планшет, он планирует играть в игры, сидеть в интернете, делать фотографии и т.д. Все это есть в инструкции, но если планшет не открывает приложения, а фотографии не появляются в галереи, то валидация не пройдена [1].

При обмене данными между различными системами необходимо проверять, соответствуют ли данные конкретным требованиям — такими требованиями может быть проверка диапазона значений, заполнения полей и т. д. Валидация проводится, если невозможно оценить, соответствует ли процесс, продукт или система критериям пользователя до того, как пользователь начнет пользоваться этим продуктом [1].

Задача проверки данных хорошо известна всем разработчикам. Будь то данные, введенные пользователем в форму, сохраненные в базе данных или отправленные во внешний сервис, — все они должны соответствовать определенным правилам, быть правильно отформатированы.

В разных частях приложения данные проверяются по-разному по одним и тем же правилам. Кроме того, ошибки обрабатываются различными способами, а в остальных случаях они просто блокируются.

В этом случае стандартной проблемой является дублирование знаний о правилах проверки, поскольку они встроены в сам механизм проверки. Решением этой проблемы является введение единого механизма описания правил, который затем применяется на всех уровнях. Например, на основе этих правил будут сгенерированы валидаторы для полей в веб-формах, которые будут проверять входные данные и выходные данные в методах веб-сервиса перед сохранением данных в базу данных.

1.1.1 Обзор существующих аналогов

Произведем разбор инструментов, используемых разработчиками для валидации.

Первым этапом валидации будет проверка *HTML*-кода на соответствие стандартам, отвечающим за правила размещения *HTML*-страниц в сети.

Если страница не проходит валидацию, то высок риск некорректного отображения контента во всех существующих браузерах. В большинстве случаев эта процедура не занимает много времени и редко приводит к появлению большого количества ошибок.

Проверить валидацию можно с помощью одного из специальных сервисов. Самый популярный – *Markup Validation Service* это валидатор, который позволяет пользователям Интернета проверять документы *HTML* до *HTML5* на наличие хорошо сформированной разметки по определению типа документа. Чтобы им воспользоваться нужно:

- открыть страницу сервиса;
- ввести адрес страницы, которую нужно проверить;
- нажать кнопку *Check*.

Через несколько секунд отобразится информация об успешном прохождении валидации либо появится список ошибок, которые нужно устранить.

Второй шаг на пути к успешной валидации страницы – проверка *CSS*-разметки. Правильное использование стилей, как и проверка *HTML*, гарантирует корректное отображение всех элементов сайта.

Валидатор *CSS* выполняет ту же функцию, что и валидатор *HTML*, проверяет *CSS*-код на соответствие стандартам.

Для валидации используется сервис *CSS Validation*, который гарантирует, что код *CSS* соответствует стандартам, установленным для языка.

Существуют отдельные сервисы, помогающие оценить корректность написанного кода. Здесь со стандартами сложнее, потому что основным мерилем качества кода является его работоспособность. Большая часть редакторов не станут ругаться на программу, если она функционирует.

Для валидации используются дополнительные плагины и сервисы со своим набором правил для оформления кода, например, линтеры. Они сообщают о наличии неиспользуемых переменных, о не указанных типах, об отсутствии необходимых атрибутов или некорректно используемых циклах.

Некоторые компании создают свои требования к коду и иногда прилагают собственные инструменты для проверки качества написанных приложений.

Например, бренд *AirBnb* создал свой свод правил для оформления *JavaScript*-кода. В нем запрещены многие функции, а разработчиков постоянно мотивируют использовать более современный синтаксис. Этот свод правил обрел популярность и превратился в стандарт во многих компаниях.

Чтобы тоже начать писать код в соответствии с заветами *AirBnb*, нужно обзавестись линтером и загрузить отдельный плагин с правилами оформления кода от *AirBnb*.

Еще один тип валидации, который стоило бы отметить – *Google Lighthouse*. Это комплекс мер по оценке качества созданного сайта или приложения. *Lighthouse* встроен в браузер *Google Chrome* и в автоматическом режиме показывает, что можно исправить, чтобы увеличить производительность и сделать работу ресурса эффективнее.

1.1.2 Постановка задачи

В данной работе реализуется программное приложение, осуществляющее сбор информации объекта и прохождение валидации.

К приложению предъявляются следующие требования:

- изучить основы разработки приложений;

- изучить архитектуру клиент-серверных приложений;
- изучить актуальные среды разработки;
- изучить подходящие фреймворки клиент-серверных приложений;
- изучить популярные СУБД;
- предусмотреть наличие системы авторизации пользователя;
- предусмотреть возможность добавления пользователей в систему;
- предусмотреть возможность редактирования объектов из админ-панели;
- реализовать общение базы данных с веб-интерфейсом;
- предусмотреть возможность создания новых объектов и хранение их в базе данных;
- реализовать валидацию объектов по установленным требованиям;
- разработать функционал сервера с клиентами.

1.1.3 Выбор языка программирования

Для разработки программного приложения существует широкий выбор языков программирования. Каждый из этих языков имеет преимущества и недостатки.

C# - объектно-ориентированный язык программирования. Основное отличие C# от других языков заключается в наличии быстрого и эффективного компилятора. Также C# имеет инструменты для работы с БД и позволяют эффективно создавать приложения.

Преимущества C#:

- простой и мощный язык, позволяющий создавать многофункциональные приложения;
- большое разнообразие синтаксических конструкций и возможность работать с платформой .Net;
- высокая надежность;
- наличие интегрированного отладчика, имеющего огромное количество полезных свойств.

Недостатки C#:

- сложный синтаксис;
- низкая производительность и скорость;
- готовые конструкции языка работают неоднозначно.

Язык программирования C++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день C++ является одним из самых популярных и распространенных языков программирования.

Преимущества C++:

- высокая скорость;
- универсальность;
- широкая сфера применения;
- высокий уровень востребованности.

Недостатки C++:

- сложный и большой язык программирования;
- неинтуитивные преобразования некоторых типов;
- плохая проверка модульности;
- трудный для понимания синтаксис.

Python - популярный высокоуровневый язык программирования, применение которого довольно широко, на нем можно разрабатывать множество приложений различных типов (веб-приложение, игры и т.д.). Язык обладает простым синтаксисом и огромным количеством библиотек под разные задачи [2].

Преимущества *Python*:

- простота изучения;
- высокая востребованность языка;
- использование в технологиях искусственного интеллекта и машинного обучения;

- написание программ для ПК и веб-приложений;
- открытый исходный код и бесплатная доступность языка на официальном сайте;
- независимость от ОС.

Недостатки *Python*:

- медленная работа;
- мало возможностей для процессорных оптимизаций.

Java - язык высокого уровня, имеющий простой синтаксис. Является компилируемым и интерпретируемым языком программирования. Поддерживает множество библиотек, имеет стандарт для корпоративных вычислительных систем.

Это простой и легкий язык для изучения. Если оценивать его по сложности, то он имеет среднюю сложность в освоении. В целом подходит для обучения даже новичкам, которые ранее никогда не сталкивались с программированием.

Как и все многие языки программирования, *Java* имеет свои преимущества и недостатки.

Преимущества *Java*:

- объектно-ориентированное программирование;
- язык высокого уровня с простым синтаксисом;
- стандарт для корпоративных вычислительных систем;
- надежность кода;
- независимость от платформы;
- многопоточность;
- автоматическое управление памятью.

Недостатки *Java*:

- платное коммерческое использование;
- низкая производительность;
- многослойный код.

Таким образом, учитывая все преимущества и недостатки вышеперечисленных языков программирования – был выбран язык *Python* 3.10.4.

1.1.4 Выбор библиотеки для разработки серверного приложения

Python имеет две основных библиотеки для разработки веб-приложений: *Flask* и *Django*.

Flask – это небольшой и легкий веб-фреймворк, написанный на языке *Python*, предлагающий полезные инструменты и функции для облегчения процесса создания веб-приложений с использованием *Python*. *Flask* можно использовать при разработке как тренировочных проектов или небольших сайтов, которым не нужен сложный бэкенд, так и сложных проектов для электронной коммерции. Он обеспечивает гибкость и является более доступным для новых разработчиков, так как позволяет создать веб-приложение быстро, используя только один файл *Python*. *Flask* используется в основном для построения минималистичных веб-проектов [3].

Преимущества *Flask*:

- простая и легкая архитектура;
- неплохие возможности работы с веб-приложениями;
- легкий тест блоков и функций;
- тщательная документация.

Недостатки *Flask*:

- единственный источник затрудняет масштабирование;
- сторонние модули могут представлять угрозу безопасности;
- ручная установка библиотек.

Наше приложение является нагруженным и должно иметь возможность дальнейшего расширения, поэтому рассмотрим *Django*.

Django — это свободный фреймворк для разработки быстрых и безопасных веб-приложений и сайтов на языке *Python*. Реализован на основе архитектуры *MVC*. Имеет множество библиотек и собственный *ORM*, с

помощью которого, можно работать со многими СУБД. Представления реализуются как функциональным подходом, так и объектно-ориентированным [4].

Преимущества *Django*:

- разносторонность и обширная документация;
- автоматически генерируемая админ-панель;
- обеспечивает лучшую безопасность веб-приложения;
- маленькая затрата времени на завершение крупных проектов;
- высокий уровень производительности;
- легок в обучении [5].

Недостатки *Django*:

- позволяет создавать только одноуровневые приложения;
- не подходит для небольших приложений;
- требует большего понимания структуры.

1.1.5 Выбор СУБД

Существует масса различных баз данных, которые можно использовать для нужд *Django*, и каждая из них требует наличия как самого программного обеспечения, реализующего базу данных, так и библиотеки для языка *Python*, которая обеспечит взаимодействие с ней [4].

PostgreSQL — это полноценный веб-сервер баз данных, обладающий широким спектром функциональных возможностей и давней историей развития, который является одним из ведущих приложений баз данных, распространяемых с открытыми исходными текстами. Эта база данных рекомендуется к использованию основными разработчиками *Django*, которые дают весьма высокие оценки ее качеству.

Преимущества *PostgreSQL*:

- широкий набор инструментов;
- надежность;
- объектно-ориентированность;

- расширяемость.

Недостатки *PostgreSQL*:

- производительность;
- непопулярный инструмент из-за своей сложности;
- низкая скорость при быстрых операциях.

MySQL- самая популярная система управления базами данных для *web*-приложений. Создатели *MySQL* распространяют ее в качестве свободного ПО. Установив систему управления базами данных *MySQL* бесплатно, можно использовать ее много раз.

Преимущества *MySQL*:

- простота;
- поддержка большей части функционала *SQL*;
- масштабируемость;
- синхронизация с другими БД.

Недостатки *MySQL*:

- присутствуют определенные ограничения функциональности;
- недостаточная надежность;
- застой в разработке.

SQLite — встраиваемая кроссплатформенная БД, которая поддерживает наиболее полный набор команд *SQL*.

В отличие от *PostgreSQL*, *MySQL* и многих других баз данных, *SQLite* не является самостоятельным сервером, это просто библиотека, реализующая интерфейс доступа к дисковым файлам базы данных.

SQLite прекрасно подходит для быстрого запуска процесса разработки или для маленьких сайтов, где нет необходимости устанавливать полноценный веб-сервер баз данных. Чтобы иметь возможность взаимодействовать с базой данных *SQLite*, потребуется библиотека для языка *Python*. Используя *Python*, имеется встроенный модуль *sqlite3*. *SQLite* не требует явного создания базы данных или механизма управления

пользователями. Вместо этого просто выбирается каталог в файловой системе для базы данных и его имя записывается в виде параметра *DATABASES* в файле *settings.py*. После этого в указанном каталоге будет создан файл базы данных при обычном использовании утилиты *manage.py* [3].

Преимущества *SQLite*:

- простота;
- высокая производительность;
- открытость;
- низкая нагрузка на сервер.

Недостатки *SQLite*:

- недостаточная функциональность;
- недостаточная надежность;
- отсутствие совместной работы с данными.

Желательно, чтобы СУБД поддерживалась *Django ORM*, что позволит работать с данными и системой управления максимально эффективно. Поэтому для реализации проекта была выбрана *sqlite3*, которая отлично подходит для наших задач.

1.2 Проектирование и разработка

Система состоит только из серверной части приложения. Предполагается, что клиентами будут выступать браузеры пользователей, которые будут получать заполненные *html*-шаблоны, а также отображать их содержимое на экране устройства. Приложение работает на основе *МРА* шаблона. Этот подход был выбран на основе нескольких причин:

- наиболее простая реализация средствами *Django*;
- конечное приложение не обладает большим количеством динамических страниц;

- конечное приложение не передает большое количество данных клиентам;

- конечное приложение не имеет собственного *API*.

Регистрация пользователей не подразумевается. Для доступа к веб-приложению пользователю необходимо обратиться к администратору с целью получения логина и пароля для авторизации в системе.

1.2.1 Реализация БД

В качестве СУБД была выбрана *sqlite3*, которая поддерживает все необходимые типы данных, позволяет сохранять ссылки на изображения и файлы. Этого достаточно для реализации хранения данных в рамках ВКР, а также для локальной разработки. Т.к. одним из главных преимуществ *Django* является встроенный *ORM*, то схема БД зависела от реализации моделей представления данных в проекте. Ниже приведены описания моделей, отвечающих за хранение файла и данных продукта:

```
class File(models.Model):
    name = models.CharField(null=True, blank=True,
max_length=250, verbose_name = 'Название')
    file = models.FileField(upload_to="files/%Y/%m/%d",
verbose_name = '')
    product = models.ForeignKey(Product,
on_delete=models.CASCADE, null=True, blank=True,
related_name='product_file', verbose_name = 'Товар')
    created = models.DateTimeField(auto_now_add = True,
verbose_name='Создано')
    uploaded = models.DateTimeField(auto_now=True,
verbose_name='Обновлено')

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

class Meta:
    verbose_name = "Файл"
```

```

verbose_name_plural = "Файлы"

class Product(models.Model):
    title = models.CharField(max_length=250, verbose_name =
'Название')# Название товара. Проверяется на длину. Максимум
250 символов. Не может быть пустым.
    body = models.TextField(blank=True,
verbose_name='Описание')# Описание товара. Может быть пустым
    slug = models.SlugField(max_length=150, unique=True,
verbose_name="URL")# Ссылка на товар. Не может быть больше 150
символов. Только латинские символы. Только уникальные(не
повторяющиеся) значения.
    product_category = models.ManyToManyField(Product_category,
verbose_name = 'Категория товара') # Не может быть пустой
    storage_type = models.ManyToManyField(Storage_type,
verbose_name = 'Вид хранения')# Не может быть пустой
    provider = models.ManyToManyField(Provider, verbose_name =
'Поставщик')# Не может быть пустой
    room_layout = models.ManyToManyField(Room_layout,
verbose_name = 'Склад хранения')# Не может быть пустой
    quantity = models.DecimalField(decimal_places=2,
max_digits=20, default=0, verbose_name = 'количество ед.
товара')# Количество знаков после запятой не более двух.
Количество символов не более 20.
    height = models.DecimalField(decimal_places=2,
max_digits=20, default=0, verbose_name = 'Высота в см. 1шт')#
Количество знаков после запятой не более двух. Количество
символов не более 20.
    width = models.DecimalField(decimal_places=2,
max_digits=20, default=0, verbose_name = 'Ширина в см. 1шт')#
Количество знаков после запятой не более двух. Количество
символов не более 20.
    depth = models.DecimalField(decimal_places=2,
max_digits=20, default=0, verbose_name = 'Глубина в см. 1шт')#

```

Количество знаков после запятой не более двух. Колличество символов не более 20.

```
weight = models.DecimalField(decimal_places=2,
max_digits=20, default=0, verbose_name = 'Вес в кг. 1шт')#
Количество знаков после запятой не более двух. Колличество
символов не более 20.
```

```
publication = models.BooleanField(default=False,
verbose_name = 'Публикация')
```

```
start_date = models.DateTimeField(verbose_name='Дата
поставок')# Не может быть пустой. Проверяется корректность
введенных данных - формат пример 2022-03-07 12:00
```

```
created = models.DateTimeField(auto_now_add = True,
verbose_name='Создано')# Создается автоматически
```

```
uploaded = models.DateTimeField(auto_now=True,
verbose_name='Обновлено')# Создается автоматически
```

```
product_icon =
models.ImageField(upload_to='img/Product_icon/%Y/%m/%d',
verbose_name="Фото товара")# Не может быть пустым.
```

```
def __str__(self):
return self.title
```

```
class Meta:
verbose_name = "Продукты"
verbose_name_plural = "Продукты"
```

Модель продукта является самой сложной моделью в системе, поскольку содержит в себе много полей с самыми разными типами хранимых данных, например, поле хранения *URL*, поля со связями «многие ко многим», поля хранения дат с автоматической простановкой значений. Немаловажной особенностью работы с моделями в *Django* является то, что по умолчанию автоматически создается поле «*id*», которое хранит в себе уникальный

идентификатор (первичный ключ), позволяющий идентифицировать каждую сущность по числовому значению.

На рисунке 1 представлена схема БД.

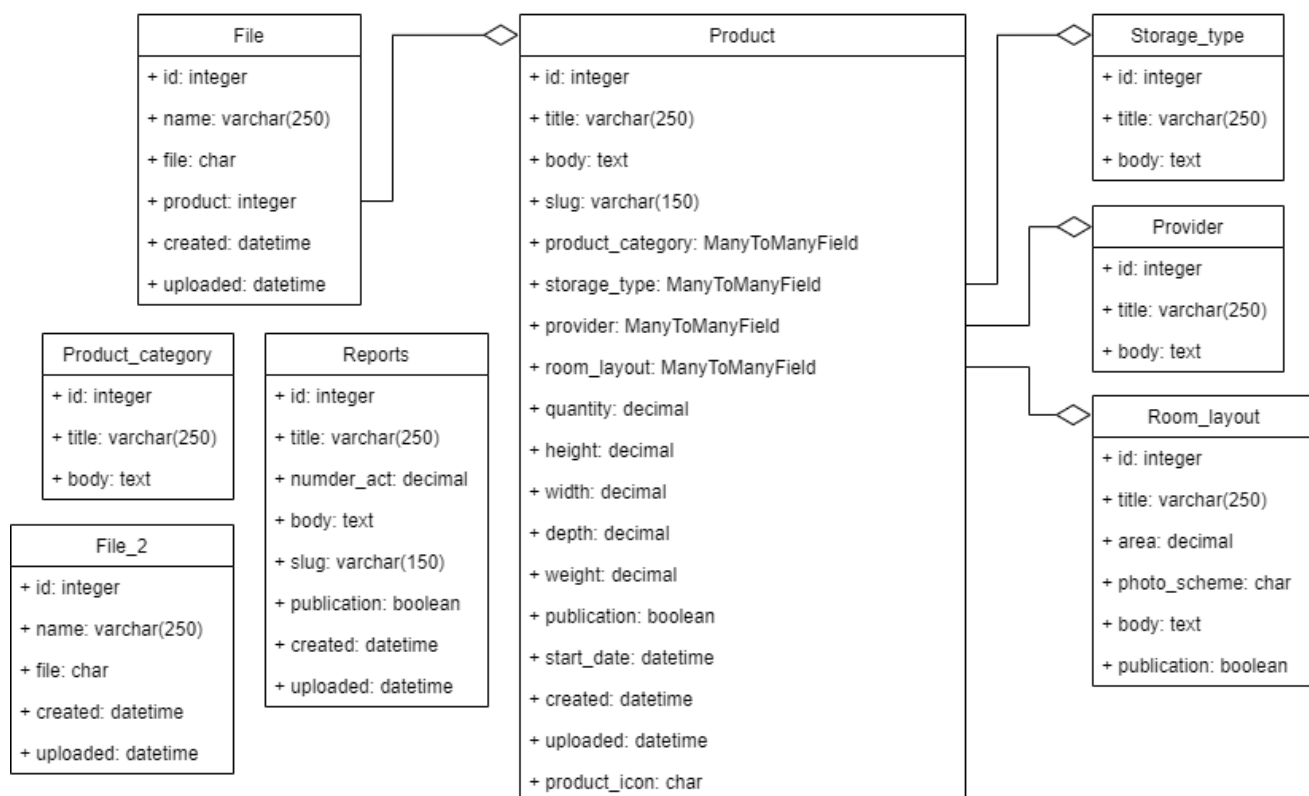


Рисунок 1 - Схема базы данных

На рисунке 2 представлена диаграмма классов моделей.

1.2.2 Реализация хранения файлов и изображений

Sqlite3 не поддерживает хранение данных непосредственно в базе, но это и не требуется т.к. хранение файлов крайне не рекомендуется реализовывать с применением СУБД, поскольку это может повлечь за собой проблемы с выборкой данных, нерациональной затратой дискового пространства, а также могут возникнуть затруднения касающиеся файловой системы сервера. Поэтому хранение файлов реализовано в виде хранения пути до файла в папке «*media*» и названия файла. Файлы распределяются в

папке по дате загрузки по годам, месяцам и дням, что облегчает их поиск и выборку.

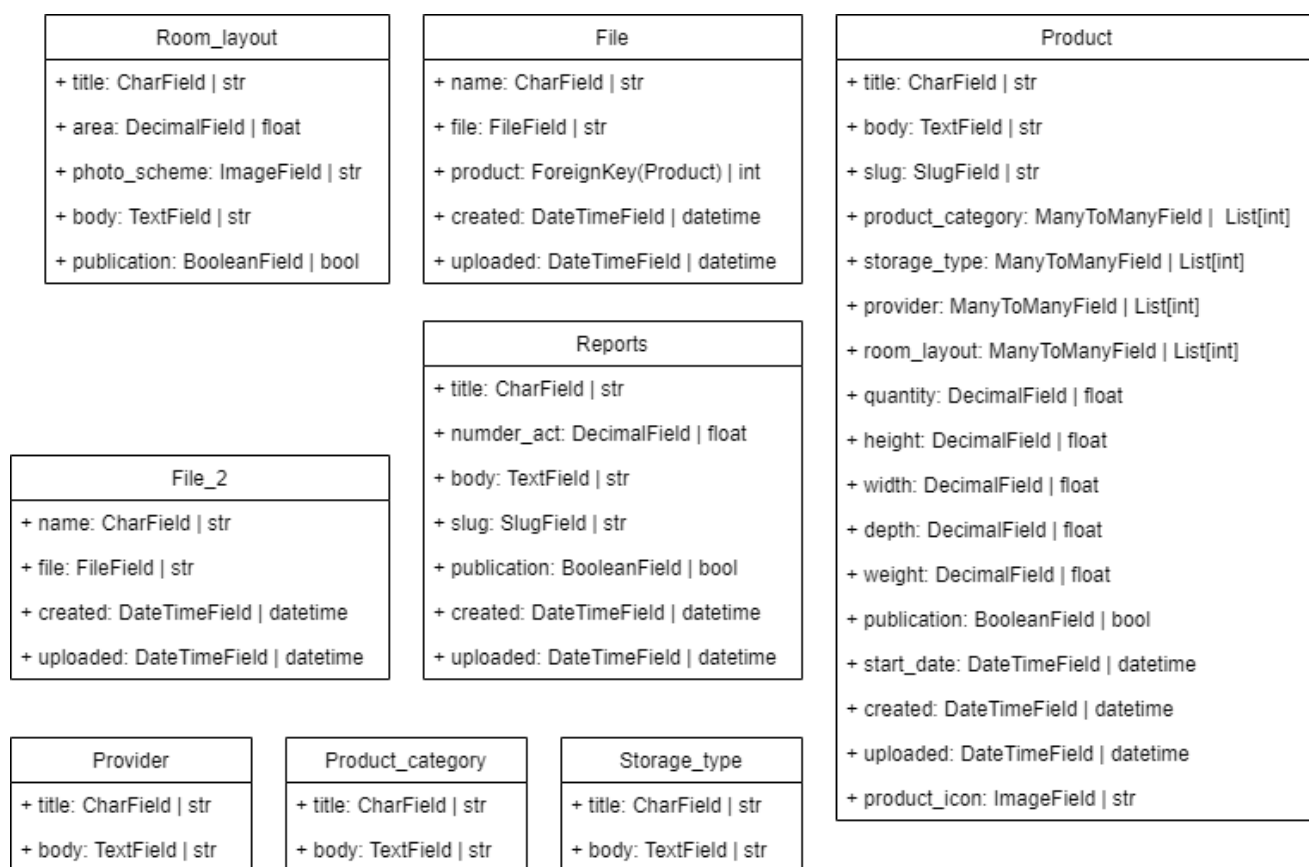


Рисунок 2 – UML-диаграмма классов моделей

Для включения поддержки файлов в файле «settings.py» были указаны следующие конфигурационные параметры:

```
MEDIA_URL = '/media/'
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Файлы хранятся в директории «*media/files*», а изображения «*media/img*». *Django* позволяет изменять и удалять изображение из панели администратора.

1.2.3 Реализация серверной части

При реализации серверной части сначала был создан проект консольной командой:

```
django-admin startproject museum
```

Django автоматически создаст проект с минимальным функционалом.

Затем были заданы настройки в файле «*settings.py*». Часть настроек приведена ниже:

```
LANGUAGE_CODE = 'ru'
```

```
TIME_ZONE = 'Europe/Moscow'
```

```
USE_I18N = True
```

```
USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
```

```
# https://docs.djangoproject.com/en/4.0/howto/static-files/
```

```
STATIC_URL = 'static/'
```

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

```
#STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

```
# Media files
```

```
MEDIA_URL = '/media/'
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

```
# Default primary key field type
```

```
#https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field
```

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Для реализации представлений в *Django* можно применить два подхода: функциональный и объектно-ориентированный. В данной реализации был применен функциональный подход, т.к. разработка с его применением занимает гораздо меньше времени и реализовывается в гораздо меньшее количество строк кода. Было реализовано восемь функций для обработки каждой конечной точки, присутствующей в проекте.

Ниже приведена таблица с названием функции, конечной точкой и предназначением конечной точки.

					ПГУ 1.090301.16.001 ПЗ	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1 – Описание функций представлений и привязки к конечным точкам

Название функции	Конечная точка	Предназначение
<i>login</i>	<i>/login/</i>	Авторизация пользователей
<i>product_show</i>	<i>/product/</i>	Отображение списка товаров
<i>product_create</i>	<i>/product/create/</i>	Создание новой сущности товара
<i>product_details_show</i>	<i>/product_details/<slug:slug></i>	Отображение данных о товаре
<i>documentation_create</i>	<i>/documentation/create/</i>	Создание документа
<i>documentation_show</i>	<i>/documentation/</i>	Отображение списка документов
<i>documentation_details_show</i>	<i>/documentation/<slug:slug>/</i>	Отображение документа
<i>room_layout_show</i>	<i>/room_layout/</i>	Отображение схем складов
ОТСУТСТВУЕТ	<i>logout</i>	Деавторизация пользователей

На рисунке 3 представлена диаграмма развертывания приложения.

1.2.4 Реализация шаблонов клиентской части

Шаблонное проектирование страниц в *Django* позволяет сократить количество необходимого кода. Существует два подхода, которые могут дополнять друг друга – проектирование на основе базовых шаблонов и

компонентов. Т.к. страницы единообразны и большинство элементов повторяются, то был выбран подход, основанный на базовом шаблоне.

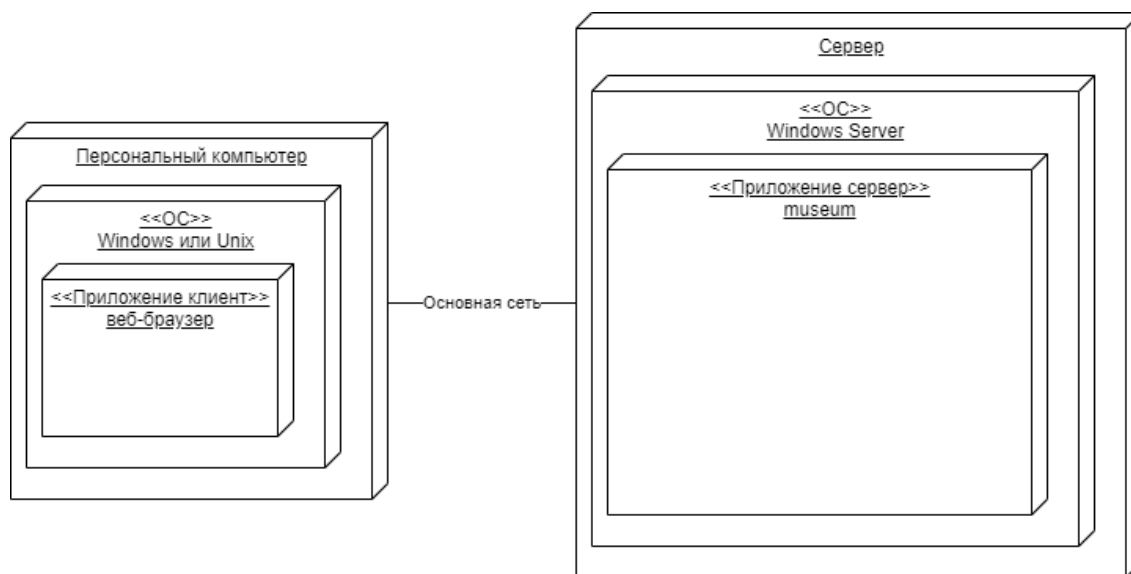


Рисунок 3 – Диаграмма развертывания приложения

Ниже приведен листинг базового шаблона основных страниц:

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Система учета товаров</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/semantic-
ui@2.4.2/dist/semantic.min.css">
    <script src="https://cdn.jsdelivr.net/npm/semantic-
ui@2.4.2/dist/semantic.min.js"></script>
    {% block css_js_content %}      {% endblock %}
</head>
<body>
```

```

<div class="ui fluid container">
    <div class="ui large fluid pointing menu">
        <h1><a style="color: black;" href={% url
'product_show' %}> <i class="cart arrow down icon"></i>
Система товаров</a></h1>
        <div class="right item">
            {% if user.is_authenticated %}
            <a class="ui fluid large red submit button"
href={% url 'logout' %}>Выход</a>
            {% else %}
            <a class="ui button" href="login">Вход</a>
            {% endif %}
        </div>
    </div>
</div>
{% block content %}
{% endblock %}
</body>
</html>

```

Базовый шаблон включает в себя глобальные стили, шрифты, изображения и скрипты, загруженные из директории «*static*», после чего каждая из страниц, отвечающая за свою конечную точку, отображает содержимое страницы, включая базовый шаблон и глобальные компоненты. Это позволяет не использовать повторно один и тот же код, например, навигационной панели. Базовый шаблон основного приложения расположен в директории «*main*», а шаблоны авторизации в папке «*registration*».

1.3 Тестирование и отладка

Тестирование и отладка производились средствами среды разработки и инструментами отладки *Django*. Для тестирования была создана учетная запись администратора командой:

```
python ./manage.py createsuperuser
```

В итоговом приложении администраторы обладают большим набором привилегий, в отличие от пользователей, они могут добавлять, редактировать и удалять данные, а также добавлять пользователей в систему. На рисунке 4 представлена диаграмма прецедентов итогового приложения.

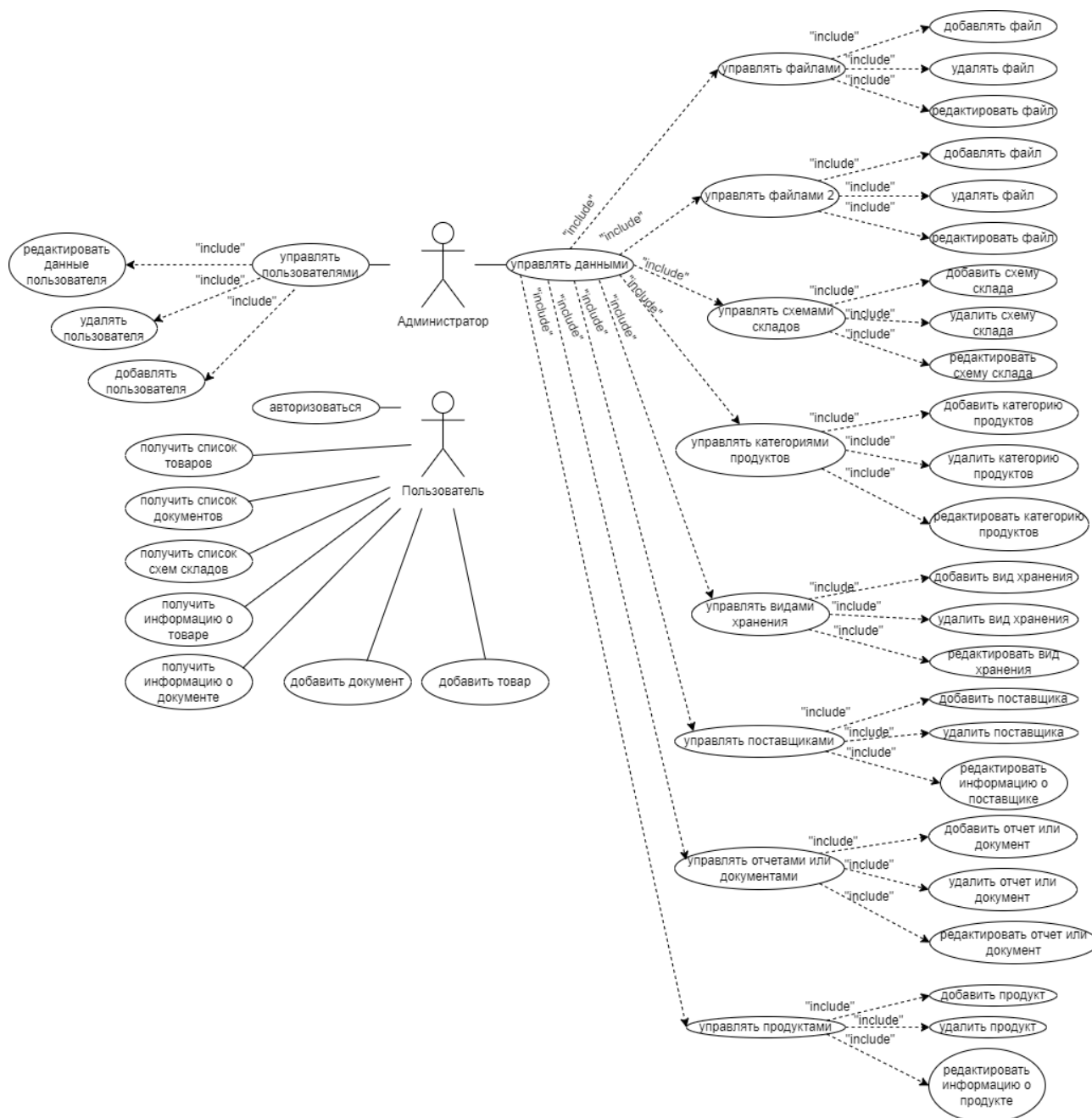


Рисунок 4 – UML-диаграмма прецедентов

Варианты использования приложения пользователями представлены на диаграмме последовательностей, представленной на рисунке 5.

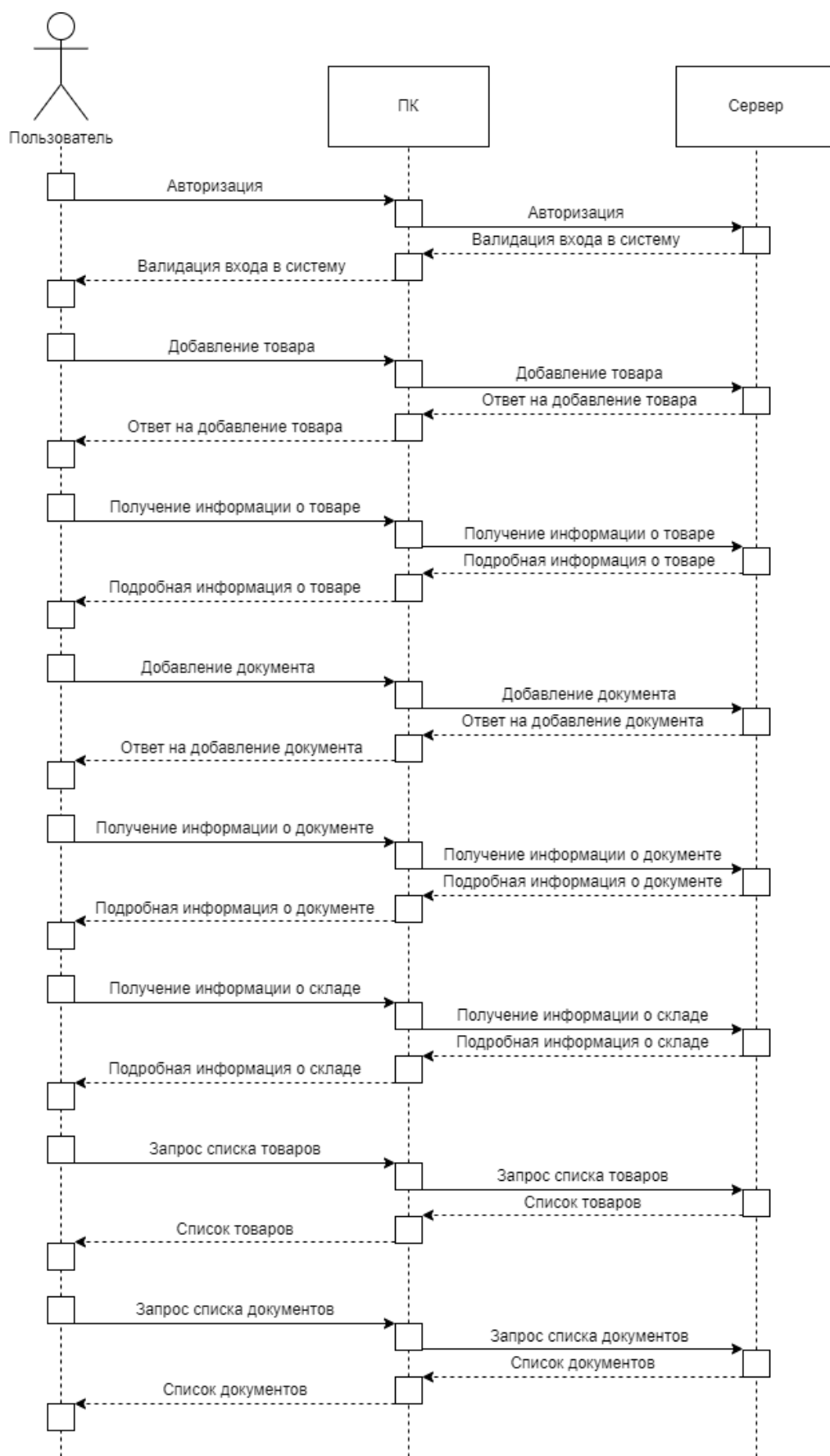


Рисунок 5 – UML-диаграмма последовательностей

1.3.1 Функционал веб-приложения

Перед началом работы пользователю необходимо авторизоваться в веб-приложении введя логин и пароль, предоставленные администратором системы. На рисунке 6 представлена страница авторизации системы.

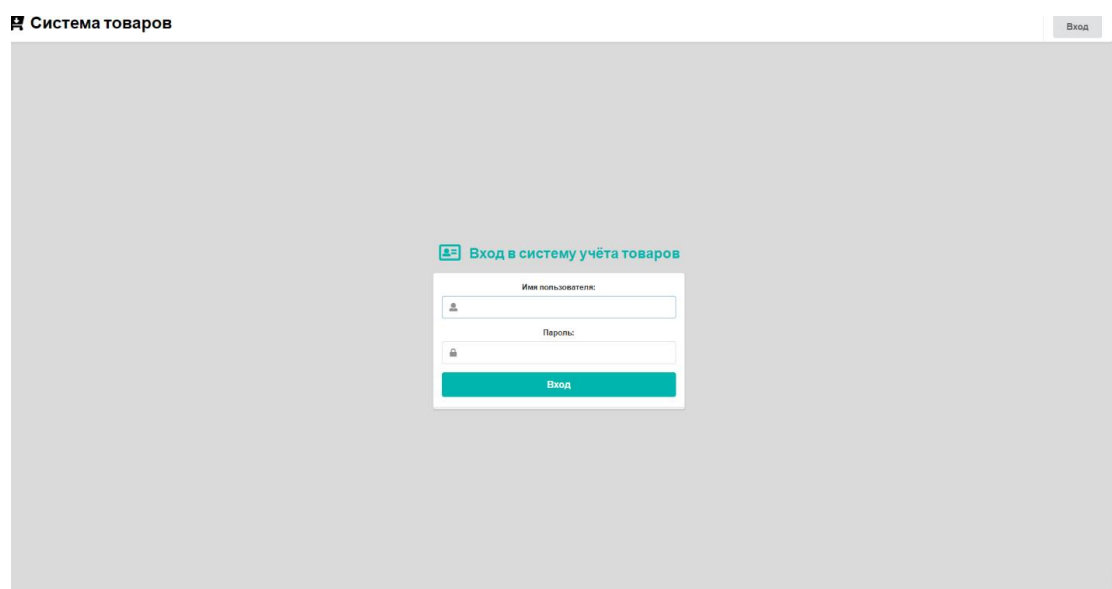


Рисунок 6 – Страница авторизации системы

Если пользователь введет неверный логин или пароль, система выдаст соответствующее сообщение с ошибкой, представленное на рисунке 7.

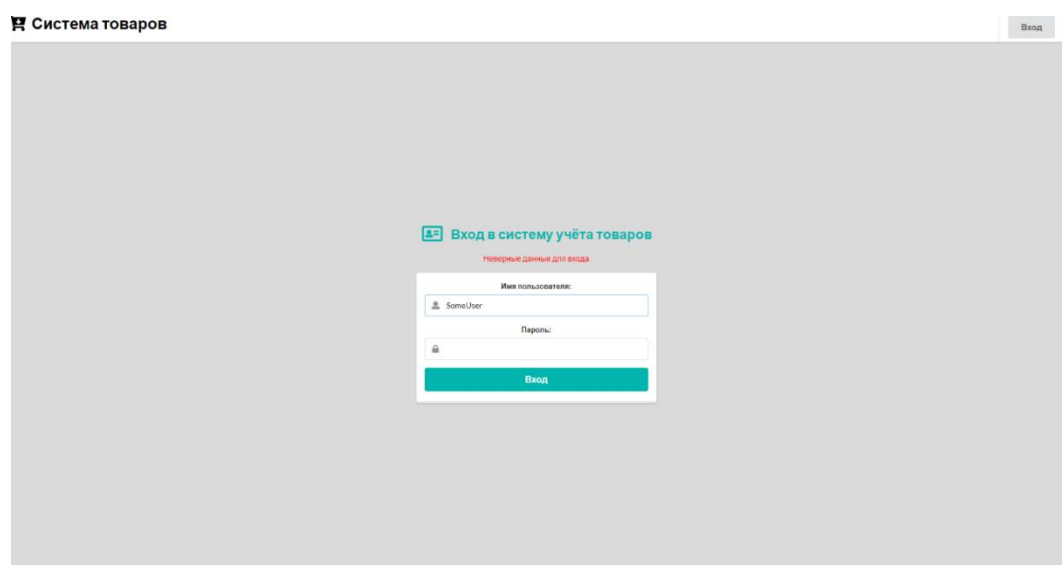


Рисунок 7 – Ошибка авторизации пользователя в системе

Как только пользователь будет авторизован, он попадет на главную страницу приложения, на которой он сможет произвести выход из системы, чтобы сменить учетную запись или начать работу с данными.

На рисунке 8 представлена страница выхода из системы.

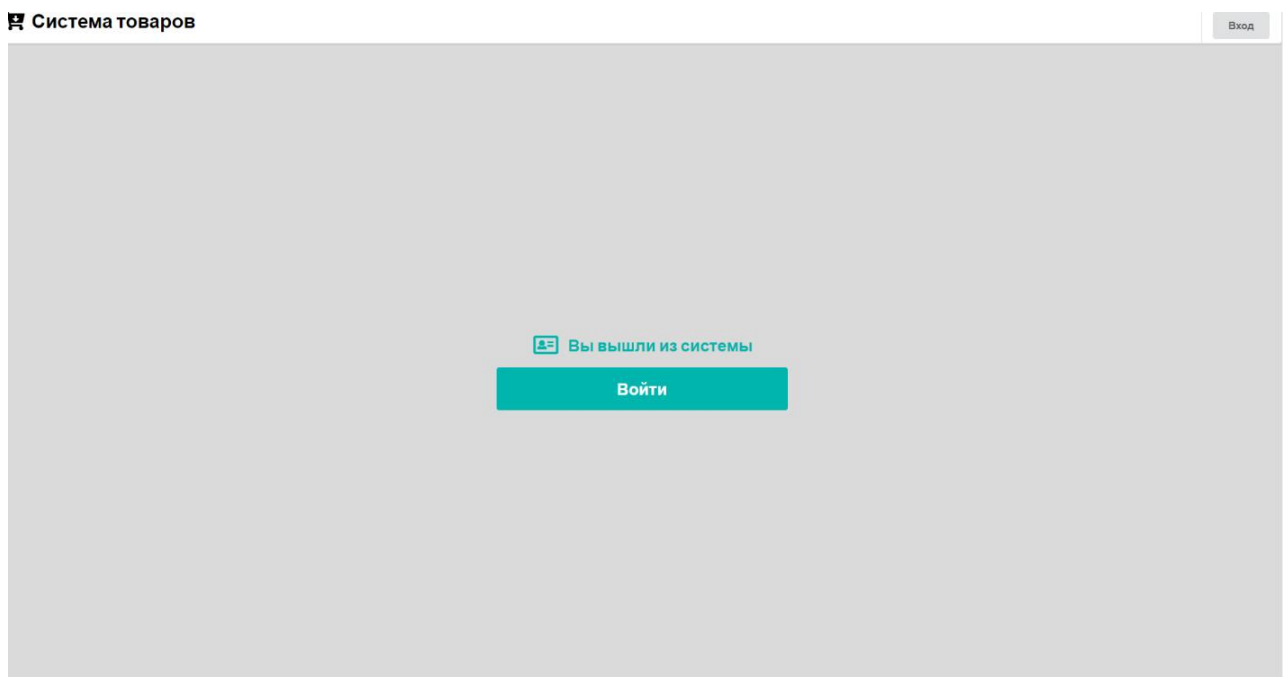


Рисунок 8 – Страница выхода из системы

На главной странице пользователь может переключаться между тремя активными вкладками «Товары», «Схемы складов хранения» и «Документация». Переход на каждую из вкладок загружает новую страницу со списком элементов в соответствующей категории.

На рисунке 9 представлена главная страница системы.

В разделе управления товарами пользователь может добавить товар.

Страница добавления товара представлена на рисунке 10.

Если пользователь введет данные не подходящего формата, система выведет уведомление с указанием на поле, в котором допущена ошибка.

Товары	Товар	Фото товара	Дата поставки товара	Количество
Схемы складов хранения	Добавить товар			
Документация				

Рисунок 9 – Главная страница системы

Новый товар

Название:	<input type="text"/>
Описание:	<input type="text"/>
URL:	<input type="text"/>
Категория товара:	<input type="text"/>
Вид хранения:	<input type="text"/>
Поставщик:	<input type="text"/>
Склад хранения:	<input type="text"/>
Количество ед. товара:	<input type="text"/>
Высота в см. 1шт:	<input type="text"/>
Ширина в см. 1шт:	<input type="text"/>

Рисунок 10 – Страница добавления товара

На рисунке 11 представлен вывод ошибки при валидации добавления данных в систему.

Новый товар

Название:

Описание: Заполните это поле.

URL:

Категория товара:

Вид хранения:

Поставщик:

Склад хранения:

Количество ед. товара:

Высота в см. Шт:

Ширина в см. Шт:

Рисунок 11 – Ошибка валидации данных при добавлении товара в систему

Для добавления данных в систему пользователю необходимо заполнить все поля формы корректно. На рисунке 12 представлена корректно заполненная форма товара.

Новый товар

Название:

Описание:

URL:

Категория товара:

Вид хранения:

Поставщик:

Склад хранения:

Рисунок 12 – Корректно заполненная форма товара

Изм.	Лист	№ докум.	Подпись	Дата

После добавления товара в систему форма очистится. Далее появляется возможность, позволяющая пользователям добавлять несколько товаров подряд.

Для проверки добавленных данных пользователю необходимо вернуться на главную страницу приложения, а затем просмотреть список товаров.

На рисунке 13 представлен список товаров после добавления нескольких позиций.

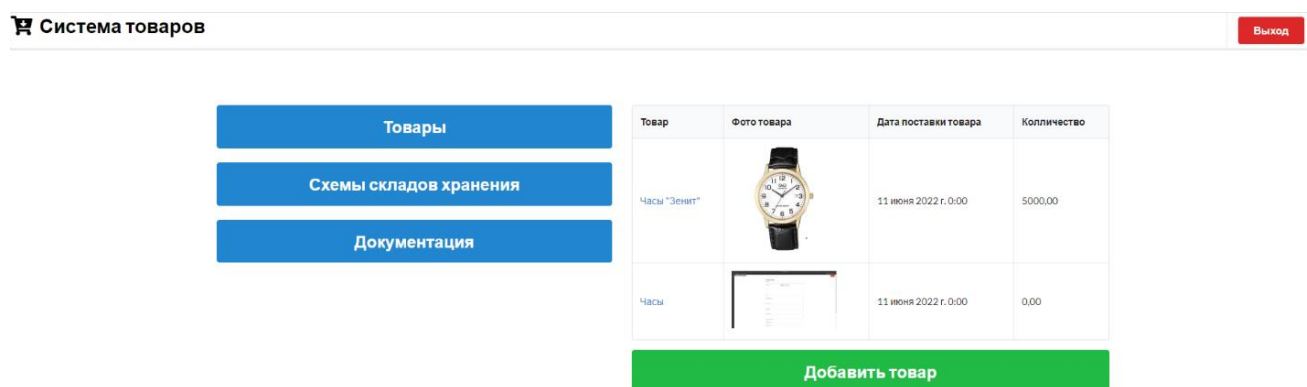


Рисунок 13 – Список товаров после добавления нескольких позиций

Щелчком левой кнопки мыши по названию товара пользователь может открыть страницу с подробным описанием товара.

На рисунке 14, расположенном ниже, представлена страница с подробным описанием товара.

- Товары
- Схемы складов хранения
- Документация

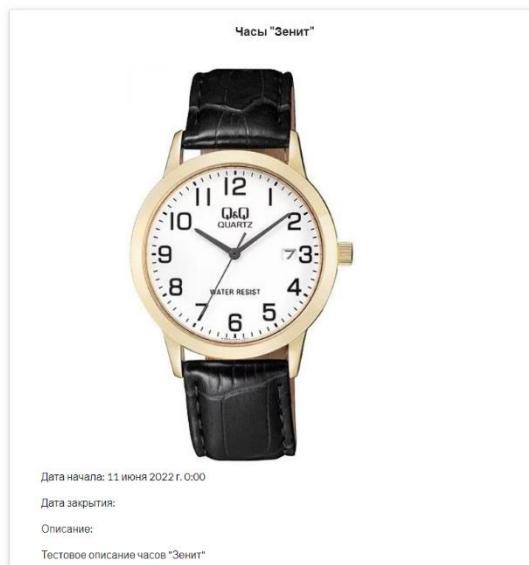


Рисунок 14 – Страница с подробным описанием товара

На рисунках 15 и 16 представлены страницы списков схем складов и списки документов соответственно.

- Товары
- Схемы складов хранения
- Документация

Название	Площадь	Схема	Описание
Пенза - Центр	150,00		Расположен недалеко от здания ТЦ "Пассаж"

Рисунок 15 – Страница списка схем складов

Товары	Название	Номер	Дата создания
Схемы складов хранения	Проверка добавления документа	1	11 июня 2022 г. 10:37
Документация	Добавить документ		

Рисунок 16 – Страница списка документов

Также, как и в случае с товарами щелчком левой кнопки мыши по названию документа открывается его подробное описание. На рисунке 17 представлена страница с подробным описанием документа.

Товары	Проверка добавления документа
Схемы складов хранения	Документ для проверки системы
Документация	Документ создан: 11 июня 2022 г. 10:37

Рисунок 17 – Страница с описанием документа

1.3.2 Функционал панели администратора

Панель администратора встроена в каждое приложение *Django*. Программисту необходимо только создать классы представления данных в панели, чтобы администратор мог ими манипулировать.

На рисунке 18 представлена стартовая страница панели администратора *Django*.

Администрирование Django

Администрирование сайта

MAIN		
Виды хранения	+ Добавить	✎ Изменить
Документы и отчёты	+ Добавить	✎ Изменить
Категории товаров	+ Добавить	✎ Изменить
Общие файлы	+ Добавить	✎ Изменить
Поставщики	+ Добавить	✎ Изменить
Продукты	+ Добавить	✎ Изменить
Схема складов	+ Добавить	✎ Изменить
Файлы	+ Добавить	✎ Изменить
ПОЛЬЗОВАТЕЛИ И ГРУППЫ		
Группы	+ Добавить	✎ Изменить
Пользователи	+ Добавить	✎ Изменить

Рисунок 18 – Стартовая страница панели администратора *Django*

На рисунке 19 представлена страница добавления видов хранения.

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN, ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Main · Виды хранения · Добавить Виды хранения

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить Виды хранения

Название: Особый

Описание:

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 19 – Страница добавления видов хранения

На рисунке 20 представлена страница добавления документов и отчетов.

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN, ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Main · Документы и отчёты · Добавить Документы и отчёты

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить Документы и отчёты

Название:

Номер акта:

Описание:

URL:

☐ Публикация

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 20 – Страница добавления документов и отчетов

На рисунке 21, расположенном ниже, представлена страница добавления категории товара.

На рисунке 22 представлена страница добавления общих файлов.

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN · ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Main · Категории товаров · Добавить Категория товара

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

«

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить Категория товара

Название: Часы

Описание: Наручные, настенные, напольные устройства для определения времени

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 21 – Страница добавления категории товара

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN · ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Main · Общие файлы · Добавить Общие файлы

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

«

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить Общие файлы

Название:

Выберите файл Файл не выбран

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 22 – Страница добавления общих файлов

На рисунке 23, расположенном ниже, представлена страница добавления поставщиков.

На рисунке 24 представлена страница добавления схемы склада.

Также администраторы обладают возможностью добавления изображений и файлов в том числе.

На рисунке 25 представлена страница добавления пользователя в систему.

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Main · Поставщики · Добавить Поставщик

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить Поставщик

Название:

Часовой завод, Пенза

Описание:

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 23 – Страница добавления поставщиков

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Main · Схема складов · Добавить Схема складов

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить Схема складов

Название:

Пенза - Центр

Площадь площадки для выставки в М2:

150

Схема площадки склада:

Выберите файл

схема Пе... Центр.png

Описание:

Расположен недалеко от здания ТЦ "Пассаж"

☒ Публикация

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 24 – Страница добавления схемы склада

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало · Пользователи и группы · Пользователи · Добавить пользователь

Start typing to filter...

MAIN

Виды хранения + Добавить

Документы и отчёты + Добавить

Категории товаров + Добавить

Общие файлы + Добавить

Поставщики + Добавить

Продукты + Добавить

Схема складов + Добавить

Файлы + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

Добавить пользователь

Сначала введите имя пользователя и пароль. Затем вы сможете ввести больше информации о пользователе.

Имя пользователя:

user1

Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./+/-/_

Пароль:

.....

Пароль не должен быть слишком похож на другую вашу личную информацию.
Ваш пароль должен содержать как минимум 8 символов.
Пароль не должен быть слишком простым и распространенным.
Пароль не может состоять только из цифр.

Подтверждение пароля:

.....

Для подтверждения введите, пожалуйста, пароль ещё раз.

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

СОХРАНИТЬ

Рисунок 25 – Страница добавления пользователя в систему

2 Безопасность жизнедеятельности

Безопасность человека должна решаться на всех этапах жизненного цикла, будь то разработка программы, ее реализация или эксплуатация.

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности, которые участвуют в разработке программного обеспечения.

2.1 Анализ опасных и вредных факторов, воздействующих на программиста при разработке приложения

Охрана труда является одним из важнейших аспектов комплекса мер по улучшению условий труда. Этим вопросам с каждым годом уделяется все больше внимания, поскольку здоровье человека имеет не только государственное значение, но и стало конкурентным фактором для работодателей при привлечении рабочей силы [6].

Для успешного выполнения всех мероприятий по охране труда и технике безопасности необходимо:

- знание физиологии труда, чтобы соответствующим образом организовать свой рабочий процесс;
- рассмотрение необходимости широкого применения передовых форм организации труда;
- сведение к минимуму ручного и низкоквалифицированного труда;
- создание условий, исключающих профессиональные заболевания и травмы [6].

Безопасность людей должна приниматься во внимание на всех этапах жизненного цикла, будь то при разработке, реализации или эксплуатации программы.

Обеспечение безопасной жизни человека во многом зависит от правильной оценки опасных и вредных производственных факторов.

Изменения в организме человека одинаковой степени тяжести могут быть вызваны разными причинами. Это могут быть факторы, связанные с рабочей средой, чрезмерные физические и умственные нагрузки, нервно-эмоциональный стресс или их комбинация.

Рабочее место должно обеспечивать защиту от возможного воздействия опасных и вредных факторов производственного процесса. Уровень этих факторов не должен превышать пределов, установленных законодательными, техническими, санитарными и инженерными нормами. Эти нормы требуют, чтобы условия труда были такими, чтобы воздействие опасных и вредных факторов на работников было полностью исключено или не превышало допустимых пределов [6].

Опасные и вредные вещества в зависимости от их природы делятся на следующие группы:

- физические;
- химические;
- психофизиологические;
- биологические.

Следующие физические факторы могут негативно повлиять на внутренний программатор:

- высокие и низкие температуры воздуха;
- чрезмерное загрязнение воздуха пылью и газами;
- чрезмерная запыленность и загазованность;
- высокая или низкая влажность;
- неадекватное освещение на рабочем месте;
- чрезмерно высокий уровень шума;
- избыточное ионизирующее излучение;
- избыточное электромагнитное поле;
- избыточное статическое электричество;
- опасность поражения электрическим током;

- на экране появляются пятна.

Химические опасности, с которыми постоянно сталкивается программист:

- излучение активных частиц при ионизации воздуха во время работы компьютера;
- в помещении нет биологической опасности.

Психологические вредные факторы, воздействующие на оператора во время смены:

- нервозность;
- психологический стресс;
- эмоциональная перегрузка.

Далее более подробно рассматриваются опасные и вредные факторы, с которыми программист должен иметь дело при разработке этой системы.

2.2 Микроклимат рабочей зоны программиста

Микроклимат производственного предприятия — это климат внутренней среды предприятия, который определяется сочетанием температуры, влажности и скорости ветра, воздействующих на организм человека [6].

В производственных помещениях должны соблюдаться следующие требования.

- оптимальная температура: 22 °С (допустимый диапазон: 20-24 °С);
- оптимальная относительная влажность: 40-60% (допустимый диапазон: максимум 75%);
- скорость ветра: макс. 0,1 м/с.

Водяное отопление автоматически создает и поддерживает оптимальные значения температуры, влажности, чистоты и скорости движения воздуха в лаборатории, независимо от внешних условий, в то время как кондиционирование воздуха используется в жаркое время года [6].

2.3 Окраска и коэффициенты отражения

Цветовая гамма и обстановка помещения должны способствовать созданию хорошего визуального впечатления и атмосферы.

Источники света, такие как лампы и окна, отражаясь от поверхности экрана, значительно снижают точность сигнала и вызывают физиологические нарушения, которые могут привести к значительному стрессу, особенно в течение длительного рабочего дня. Отражения, включая вторичные источники света, должны быть сведены к минимуму. Шторы и экраны могут использоваться для защиты от чрезмерного ослепления из окон [6].

В зависимости от ориентации окон рекомендуется использовать следующие цвета стен и пола:

окна, выходящие на юг:

- сине-зеленые или светло-голубые стены;
- зеленые полы.

Окна, выходящие на север:

- светло-оранжевые или желто-оранжевые стены;
- красно-оранжевый пол.

Окна, выходящие на восток:

- желто-зеленые стены;
- зеленый или красно-зеленый пол.

Окна, выходящие на восток:

- желто-зеленые стены;
- зеленый или красновато-коричневый пол.

Окна, выходящие на запад:

- желто-зеленые или сине-зеленые стены;
- зеленый или красновато-коричневый пол.

Коэффициент отражения света в компьютерном зале должен быть следующим:

- потолок: 60-70 %;
- стены: 40-50 %;
- пол: около 30 %.

Для письменных столов и другой мебели: 30...40 %.

2.4 Освещение рабочего места

Работа с компьютерными технологиями имеет следующие недостатки:

- полная слепота;
- измененный контраст между изображением и фоном;
- отражения на экране.

Правильно спроектированное и внедренное промышленное освещение улучшает визуальные условия труда, снижает утомляемость, повышает производительность, положительно влияет на рабочую среду, оказывает положительное психологическое воздействие на сотрудников, повышает безопасность труда и снижает количество несчастных случаев.

Недостаток света вызывает напряжение глаз, рассеянность и преждевременную усталость. Слишком яркий свет вызывает блики, раздражение и дискомфорт в глазах. Неправильное направление света на рабочем месте может вызвать сильные тени, блики и дезориентацию. Все это может привести к несчастным случаям на производстве или заболеваниям, поэтому так важно правильно рассчитать освещение.

Существует три типа света: естественный, искусственный и комбинированный (естественный и искусственный свет вместе).

Естественное освещение - освещение помещения дневным светом, проникающим через световые проемы во внешней конструкции здания. Естественное освещение характеризуется тем, что оно очень изменчиво в зависимости от времени суток, сезона, характера местности и некоторых других факторов.

Искусственное освещение используется для работы ночью и днем, когда невозможно достичь нормальных значений коэффициента естественного освещения (пасмурная погода, короткий световой день). Освещение, при котором недостаточное естественное освещение дополняется искусственным, называется комбинированным.

Искусственное освещение подразделяется на рабочее, аварийное, эвакуационное и охранное. Рабочее освещение может быть общим или комбинированным.

Общее освещение - освещение, при котором светильники расположены равномерно в верхней части пространства или в соответствии с расположением оборудования.

Комбинированное освещение - освещение, при котором местное освещение добавляется к общему.

Для работ в категории высокой точности зрения (минимальный размер различаемого объекта 0,3...0,5 мм) значение коэффициента естественной освещенности должно быть не менее 1,5 %, а для работ в категории средней точности зрения (минимальный размер различаемого объекта 0,5...1,0 мм) должен быть не менее 1,0 %. В качестве искусственных источников света обычно используются люминесцентные лампы, сгруппированные попарно в светильниках, которые должны быть равномерно расположены на рабочей поверхности.

Кроме того, все поле зрения должно быть равномерно освещено — это основное гигиеническое требование. Другими словами, уровень и яркость освещения экрана компьютера должны быть примерно одинаковыми, так как яркий свет в поле зрения значительно увеличивает напряжение глаз и, как следствие, вызывает усталость глаз.

2.5 Воздействие шума на программиста. Защита от шума

Одним из недостатков закрытой среды является высокий уровень шума от печатного оборудования, оборудования для кондиционирования воздуха и вентиляторов на системах охлаждения компьютеров.

Чтобы понять необходимость и важность снижения уровня шума, необходимо знать уровень шума на рабочем месте.

Результаты расчета должны быть сопоставлены с допустимым уровнем шума на соответствующем рабочем месте. Если результаты расчета превышают допустимый уровень шума, необходимо принять специальные меры по снижению шума. К ним относятся покрытие стен и потолков зала шумопоглощающими материалами, снижение шума в месте его возникновения, правильная планировка оборудования и рациональное расположение рабочего места оператора.

Уровни звукового давления источников шума, воздействующих на оператора на рабочем месте, приведены в таблице ниже.

Таблица 2 – Уровни звукового давления различных источников

Источник шума	Уровень шума, дБ
Жесткий диск	40
Вентилятор	45
Монитор	17
Клавиатура	10
Принтер	45
Сканер	42

Типичное рабочее место пользователя оснащено жестким диском в системном блоке, вентилятором охлаждения компьютера, монитором, клавиатурой, принтером и сканером.

Ввод уровня звукового давления каждой единицы оборудования в формулу дает следующий результат:

$$L_{\Sigma}=10 \cdot \lg(104+104,5+101,7+101+104,5+104,2)=49,5 \text{ дБ.}$$

В результате предельный уровень шума на рабочем месте пользователя в 65 дБ не превышает. Это значение еще ниже, если принять во внимание маловероятное одновременное использование периферийных устройств, таких как сканеры и принтеры. Кроме того, благодаря автоматической подаче бумаги пользователям не нужно обращать внимание на процесс печати.

					ПГУ 1.090301.16.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В процессе выполнения данной работы были изучены актуальные среды разработки, методы хранения файлов и изображений с помощью СУБД *sqlite3*, основы разработки приложений.

Также были получены навыки разработки *WEB*-приложений с использованием фреймворка Django на языке Python. Были получены навыки работы с *sqlite3*, навыки разработки шаблонов клиентской части приложения.

Изучены архитектуры приложений, их типовые примеры и инструментарий. Получены навыки по разработке клиент-серверной архитектуры.

Построены основные *UML*-диаграммы: диаграмма последовательностей, диаграмма развертывания, диаграмма классов и диаграмма прецедентов.

В результате данной работы спроектировано, разработано и отлажено программное приложение валидации объектов. Приложение создано на основе высокоуровневого языка программирования *Python* с использованием фреймворка *Django*.

Список использованных источников

1. Аладышева, Ж.И. Практические аспекты работ по валидации аналитических методик / Ж.И. Аладышева, В.В. Беляев, В.В. Береговых // Фармация. - 2008. - №7. - С. 21-24.
2. Информация о языке программирования *Python* [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Python> свободный.
3. Доусон Майкл. Програмируем на *Python*. - 3-е изд. Прогресс книга, 2019. - 416 с.
4. Документация к фреймворку *Django*. [Электронный ресурс] / Режим доступа: <https://docs.djangoproject.com/en/3.2/> свободный.
5. Плюсы и минусы *Django* [Электронный ресурс] – Режим доступа: <https://python-scripts.com/django-obzor> свободный.
6. Фролов, А. В. Безопасности жизнедеятельности. Охрана труда: Учебное пособие для вузов / А. В. Фролов. – Ростов н/Дю: Феникс, 2008. — 750 с.

Приложение А

(Обязательное)

Программное приложение валидации объектов

Листинг программы

					ПГУ 1.090301.16.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

Файл 0001_initial.py

```
from django.db import migrations, models
import django.db.models.deletion

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='File_2',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(blank=True, max_length=250,
null=True, verbose_name='Название')),
                ('file', models.FileField(upload_to='files/%Y/%m/%d',
verbose_name='')),
                ('created', models.DateTimeField(auto_now_add=True,
verbose_name='Создано')),
                ('uploaded', models.DateTimeField(auto_now=True,
verbose_name='Обновлено')),
            ],
            options={
                'verbose_name': 'Общие файлы',
                'verbose_name_plural': 'Общие файлы',
            },
        ),
        migrations.CreateModel(
            name='Product_category',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=250,
verbose_name='Название')),
                ('body', models.TextField(blank=True, null=True,
verbose_name='Описание')),
            ],
            options={
                'verbose_name': 'Категория товара',
                'verbose_name_plural': 'Категории товаров',
            },
        ),
    ]
```

```

    ),
    migrations.CreateModel(
        name='Provider',
        fields=[
            ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
            ('title', models.CharField(max_length=250,
verbose_name='Название')),
            ('body', models.TextField(blank=True, null=True,
verbose_name='Описание')),
        ],
        options={
            'verbose_name': 'Поставщик',
            'verbose_name_plural': 'Поставщики',
        },
    ),
    migrations.CreateModel(
        name='Reports',
        fields=[
            ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
            ('title', models.CharField(max_length=250,
verbose_name='Название')),
            ('number_act', models.DecimalField(blank=True,
decimal_places=0, max_digits=20, null=True, verbose_name='Номер акта')),
            ('body', models.TextField(verbose_name='Описание')),
            ('slug', models.SlugField(max_length=150, unique=True,
verbose_name='URL')),
            ('publication', models.BooleanField(default=False,
verbose_name='Публикация')),
            ('created', models.DateTimeField(auto_now_add=True,
verbose_name='Создано')),
            ('uploaded', models.DateTimeField(auto_now=True,
verbose_name='Обновлено')),
        ],
        options={
            'verbose_name': 'Документы и отчёты',
            'verbose_name_plural': 'Документы и отчёты',
        },
    ),
    migrations.CreateModel(

```

```

        name='Room_layout',
        fields=[
            ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
            ('title', models.CharField(max_length=250,
verbose_name='Название')),
            ('area', models.DecimalField(blank=True, decimal_places=2,
max_digits=5, null=True, verbose_name='Площадь площадки для выставки в М2')),
            ('photo_scheme', models.ImageField(blank=True, null=True,
upload_to='img/Room_icon/%Y/%m/%d', verbose_name='Схема площади склада')),
            ('body', models.TextField(blank=True, null=True,
verbose_name='Описание')),
            ('publication', models.BooleanField(default=False,
verbose_name='Публикация')),
        ],
        options={
            'verbose_name': 'Схема складов',
            'verbose_name_plural': 'Схема складов',
        },
    ),
    migrations.CreateModel(
        name='Storage_type',
        fields=[
            ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
            ('title', models.CharField(max_length=250,
verbose_name='Название')),
            ('body', models.TextField(blank=True, null=True,
verbose_name='Описание')),
        ],
        options={
            'verbose_name': 'Виды хранения',
            'verbose_name_plural': 'Виды хранения',
        },
    ),
    migrations.CreateModel(
        name='Product',
        fields=[
            ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),

```

```

        ('title', models.CharField(max_length=250,
verbose_name='Название')),
        ('body', models.TextField(blank=True,
verbose_name='Описание')),
        ('slug', models.SlugField(max_length=150, unique=True,
verbose_name='URL')),
        ('quantity', models.DecimalField(decimal_places=2, default=0,
max_digits=20, verbose_name='количество ед. товара')),
        ('height', models.DecimalField(decimal_places=2, default=0,
max_digits=20, verbose_name='Высота в см. 1шт')),
        ('width', models.DecimalField(decimal_places=2, default=0,
max_digits=20, verbose_name='Ширина в см. 1шт')),
        ('depth', models.DecimalField(decimal_places=2, default=0,
max_digits=20, verbose_name='Глубина в см. 1шт')),
        ('weight', models.DecimalField(decimal_places=2, default=0,
max_digits=20, verbose_name='Вес в кг. 1шт')),
        ('publication', models.BooleanField(default=False,
verbose_name='Публикация')),
        ('start_date', models.DateTimeField(verbose_name='Дата
поставок')),
        ('created', models.DateTimeField(auto_now_add=True,
verbose_name='Создано')),
        ('uploaded', models.DateTimeField(auto_now=True,
verbose_name='Обновлено')),
        ('product_icon',
models.ImageField(upload_to='img/Product_icon/%Y/%m/%d', verbose_name='Фото
товара')),
        ('product_category',
models.ManyToManyField(to='main.product_category', verbose_name='Категория
товара')),
        ('provider', models.ManyToManyField(to='main.provider',
verbose_name='Поставщик')),
        ('room_layout', models.ManyToManyField(to='main.room_layout',
verbose_name='Склад хранения')),
        ('storage_type',
models.ManyToManyField(to='main.storage_type', verbose_name='Вид хранения')),
    ],
    options={
        'verbose_name': 'Продукты',
        'verbose_name_plural': 'Продукты',
    },

```

```

    ),
    migrations.CreateModel(
        name='File',
        fields=[
            ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
            ('name', models.CharField(blank=True, max_length=250,
null=True, verbose_name='Название')),
            ('file', models.FileField(upload_to='files/%Y/%m/%d',
verbose_name='')),
            ('created', models.DateTimeField(auto_now_add=True,
verbose_name='Создано')),
            ('uploaded', models.DateTimeField(auto_now=True,
verbose_name='Обновлено')),
            ('product', models.ForeignKey(blank=True, null=True,
on_delete=django.db.models.deletion.CASCADE, related_name='product_file',
to='main.product', verbose_name='Товар')),
        ],
        options={
            'verbose_name': 'Файл',
            'verbose_name_plural': 'Файлы',
        },
    ),
]

```

Файл admin.py

```

from django.contrib import admin
from django.utils.safestring import mark_safe

from . models import *

class FileAdmin(admin.TabularInline):
    model = File
    search_fields = ('name',)

@admin.register(File_2)
class File_2Admin(admin.ModelAdmin):
    search_fields = ('name',)

@admin.register(Room_layout)
class Room_layoutAdmin(admin.ModelAdmin):
    search_fields = ('title',)

```

```

@admin.register(File)
class FilesAdmin(admin.ModelAdmin):
    model = File
    search_fields = ('name',)

@admin.register(Product_category)
class Product_categoryAdmin(admin.ModelAdmin):
    list_display = ('title',)

@admin.register(Storage_type)
class Storage_typeAdmin(admin.ModelAdmin):
    search_fields = ('title',)

@admin.register(Provider)
class ProviderAdmin(admin.ModelAdmin):
    model = Provider
    search_fields = ('title',)

@admin.register(Reports)
class ReportsAdmin(admin.ModelAdmin):
    model = Reports
    search_fields = ('title', "numder_act")

@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    prepopulated_fields = {"slug": ("title",)} # Поля автозаполнения -
переводит название в красивый url
    search_fields = ('title', 'slug') # Поля по которым осуществляется поиск
    list_filter = ('publication', 'created', 'uploaded') # Поля по которым
работают фильтры
    list_display = ('title', 'publication', 'image_show', 'slug', 'created',
'uploaded') # Поля отображаемые в панели администратора
    list_editable = ('publication',) # Поля которые можно редактировать из
панети администратора
    inlines = [FileAdmin,]
    readonly_fields = ["preview"]

    def image_show(self, obj):
        if obj.product_icon:

```

```

        return mark_safe("<img src='{ }' width='60'
/>".format(obj.product_icon.url))
    return "None"
image_show.__name__ = "Картинка"

def preview(self, obj):
    return mark_safe(f'')

```

Файл apps.py

```

from django.apps import AppConfig

class MainConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'main'

```

Файл forms.py

```

from django import forms
from .models import *
#Класс формы товара
class ProductForm(forms.ModelForm):
    files =
forms.FileField(widget=forms.ClearableFileInput(attrs={'multiple': True}),
required=False)
    class Meta:
        model = Product
        fields = '__all__'
#Класс формы отчёта
class ReportsForm(forms.ModelForm):
    class Meta:
        model = Reports
        fields = '__all__'

```

Файл models.py

```

from xml.dom.minidom import Document
from django.db import models
# Класс модели категорий товара - пищевые, канцелярские, строительные и т.д.

class Product_category(models.Model):
    title = models.CharField(max_length=250, verbose_name = 'Название')
    body = models.TextField(null=True, blank=True, verbose_name = 'Описание')

    def __str__(self):
        return self.title

```

```

class Meta:
    verbose_name = "Категория товара"
    verbose_name_plural = "Категории товаров"
# Класс модели видов хранения - заморозка, охлаждение, комнатная температура
и т.д.

class Storage_type(models.Model):
    title = models.CharField(max_length=250, verbose_name = 'Название')
    body = models.TextField(null=True, blank=True, verbose_name = 'Описание')

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = "Виды хранения"
        verbose_name_plural = "Виды хранения"

# Класс модели поставщика

class Provider(models.Model):
    title = models.CharField(max_length=250, verbose_name = 'Название')
    body = models.TextField(null=True, blank=True, verbose_name = 'Описание')

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = "Поставщик"
        verbose_name_plural = "Поставщики"

# Класс модели складов хранения

class Room_layout(models.Model):
    title = models.CharField(max_length=250, verbose_name = 'Название')
    area = models.DecimalField(decimal_places=2, max_digits=5, null=True,
blank=True, verbose_name = 'Площадь площадки для выставки в М2')
    photo_scheme = models.ImageField(null=True, blank=True,
upload_to='img/Room_icon/%Y/%m/%d', verbose_name="Схема площади склада")
    body = models.TextField(null=True, blank=True, verbose_name = 'Описание')
    publication = models.BooleanField(default=False, verbose_name =
'Публикация')

```

```

def __str__(self):
    return self.title

class Meta:
    verbose_name = "Схема складов"
    verbose_name_plural = "Схема складов"

class File_2(models.Model):
    name = models.CharField(null=True, blank=True, max_length=250,
verbose_name = 'Название')
    file = models.FileField(upload_to="files/%Y/%m/%d", verbose_name = '')
    created = models.DateTimeField(auto_now_add = True,
verbose_name='Создано')
    uploaded = models.DateTimeField(auto_now=True, verbose_name='Обновлено')

def __str__(self):
    return self.name

class Meta:
    verbose_name = "Общие файлы"
    verbose_name_plural = "Общие файлы"

# класс отчётов и документов

class Reports(models.Model):
    title = models.CharField(max_length=250, verbose_name = 'Название')#
Название. Проверяется на длину. Максимум 250 символов. Не может быть пустым.
    number_act = models.DecimalField(decimal_places=0, max_digits=20,
null=True, blank=True, verbose_name = 'Номер акта')# Количество знаков после
запятой не более двух. Колличество символов не более 20.
    body = models.TextField(verbose_name = 'Описание')# Не может быть пустым
    slug = models.SlugField(max_length=150, unique=True, verbose_name="URL")#
Ссылка на документ. Не может быть больше 150 символов. Только латинские
символы. Только уникальные(не повторяющиеся) значения.
    publication = models.BooleanField(default=False, verbose_name =
'Публикация')
    created = models.DateTimeField(auto_now_add = True,
verbose_name='Создано')# Создается автоматически

```

```

        uploaded = models.DateTimeField(auto_now=True, verbose_name='Обновлено')#
Создается автоматически

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = "Документы и отчёты"
        verbose_name_plural = "Документы и отчёты"

# Класс товаров

class Product(models.Model):
    title = models.CharField(max_length=250, verbose_name = 'Название')#
Название товара. Проверяется на длинну. Максимум 250 символов. Не может быть
пустым.
    body = models.TextField(blank=True, verbose_name = 'Описание')# Описание
товара. Может быть пустым
    slug = models.SlugField(max_length=150, unique=True, verbose_name="URL")#
Ссылка на товар. Не может быть больше 150 символов. Только латинские символы.
Только уникальные(не повторяющиеся) значения.
    product_category = models.ManyToManyField(Product_category, verbose_name
= 'Категория товара') # Не может быть пустой
    storage_type = models.ManyToManyField(Storage_type, verbose_name = 'Вид
хранения')# Не может быть пустой
    provider = models.ManyToManyField(Provider, verbose_name = 'Поставщик')#
Не может быть пустой
    room_layout = models.ManyToManyField(Room_layout, verbose_name = 'Склад
хранения')# Не может быть пустой
    quantity = models.DecimalField(decimal_places=2, max_digits=20,
default=0, verbose_name = 'количество ед. товара')# Количество знаков после
запятой не более двух. Колличество символов не более 20.
    height = models.DecimalField(decimal_places=2, max_digits=20, default=0,
verbose_name = 'Высота в см. 1шт')# Количество знаков после запятой не более
двух. Колличество символов не более 20.
    width = models.DecimalField(decimal_places=2, max_digits=20, default=0,
verbose_name = 'Ширина в см. 1шт')# Количество знаков после запятой не более
двух. Колличество символов не более 20.
    depth = models.DecimalField(decimal_places=2, max_digits=20, default=0,
verbose_name = 'Глубина в см. 1шт')# Количество знаков после запятой не более
двух. Колличество символов не более 20.

```

```

weight = models.DecimalField(decimal_places=2, max_digits=20, default=0,
verbose_name = 'Вес в кг. 1шт')# Количество знаков после запятой не более
двух. Колличество символов не более 20.

publication = models.BooleanField(default=False, verbose_name =
'Публикация')

start_date = models.DateTimeField(verbose_name='Дата поставок')# Не может
быть пустой. Проверяется корректность введенных данных - формат пример 2022-
03-07 12:00

created = models.DateTimeField(auto_now_add = True,
verbose_name='Создано')# Создается автоматически

uploaded = models.DateTimeField(auto_now=True, verbose_name='Обновлено')#
Создается автоматически

product_icon = models.ImageField(upload_to='img/Product_icon/%Y/%m/%d',
verbose_name="Фото товара")# Не может быть пустым.

def __str__(self):
    return self.title

class Meta:
    verbose_name = "Продукты"
    verbose_name_plural = "Продукты"
# Класс для хранения файлов

class File(models.Model):
    name = models.CharField(null=True, blank=True, max_length=250,
verbose_name = 'Название')
    file = models.FileField(upload_to="files/%Y/%m/%d", verbose_name = '')
    product = models.ForeignKey(Product, on_delete=models.CASCADE, null=True,
blank=True, related_name='product_file', verbose_name = 'Товар')
    created = models.DateTimeField(auto_now_add = True,
verbose_name='Создано')
    uploaded = models.DateTimeField(auto_now=True, verbose_name='Обновлено')

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

class Meta:
    verbose_name = "Файл"
    verbose_name_plural = "Файлы"

```

Файл urls.py

```
from importlib.resources import path
from django.urls import path
from .views import *
from django.contrib.auth import views

urlpatterns = [
    path('', product_show, name='product_show'),
    path('product', product_show, name='product_show'), # url отображения
всех товаров
    path('product_details/<slug:slug>', product_details_show,
name='product_details_show'), # url для отображения отдельного товара
    path('product/create', product_create, name='product_create'), # url
создания товара
    path('room_layout', room_layout_show, name='room_layout_show'), # url
отображения всех складов
    path('documentation', documentation_show, name='documentation_show'), #
url отображения документации
    path('documentation/create', documentation_create,
name='documentation_create'), # url создания документации
    path('documentation/<slug:slug>', documentation_details_show,
name='documentation_details_show'), # url для отображения отдельно взятого
документа
    path('login', views.LoginView.as_view(), name='login'), # url выхода
    path('logout/', views.LogoutView.as_view(), name='logout'), # url выхода
]
```

Файл views.py

```
from django.shortcuts import render, redirect
from .models import *
from .forms import *

# Функция отвечающая за обработку аутентификации
def login(request):
    return render(request, 'main/login.html')

#Функция отвечающая за обработку вывода всех товаров
def product_show(request):
    product_list = Product.objects.all().filter(publication = True)
    if request.user.is_authenticated:
        return render(request, 'main/index.html', {'product': product_list})
```

```

else:
    return redirect("login")

#Функция отвечающая за создание товаров в интерфейсе пользователя
def product_create(request):
    if request.user.is_authenticated:
        if request.method == "POST":
            form = ProductForm(request.POST, request.FILES)
            files = request.FILES.getlist('files')
            if form.is_valid():
                post = form.save().pk
                product = Product.objects.get(pk=post)
                if files:
                    for f in files:
                        fl = File(product=product, file = f)
                        fl.save()
                return redirect('product_show')
        else:
            form = ProductForm
            return render(request, 'main/product_create.html', {'form': form})
    else:
        return redirect("login")

#Функция вывода информации о конкретном товаре
def product_details_show(request, slug):
    product_details_list = Product.objects.get(slug = slug)
    if request.user.is_authenticated:
        return render(request, 'main/product_detail_show.html',
{'product_details': product_details_list})
    else:
        return redirect("login")

#Функция отвечающая за создание документа/отчета
def documentation_create(request):
    if request.user.is_authenticated:
        if request.method == "POST":
            form = ReportsForm(request.POST)
            if form.is_valid():
                form.save()
                return redirect('documentation_show')
    else:

```

```

        form = ReportsForm
        return render(request, 'main/documentation_create.html', {'form':
form})
    else:
        return redirect("login")

#Функция отвечающая за отображение документов
def documentation_show(request):
    reports_list = Reports.objects.all().filter(publication = True)
    if request.user.is_authenticated:
        return render(request, 'main/documentation_show.html', {'reports':
reports_list})
    else:
        return redirect("login")

#Функция отвечающая за отображение конкретного документа
def documentation_details_show(request, slug):
    documentation_details_list = Reports.objects.get(slug = slug)
    if request.user.is_authenticated:
        return render(request, 'main/documentation_detail_show.html',
{'documentation_details': documentation_details_list})
    else:
        return redirect("login")

#Функция отвечающая за отображение складов
def room_layout_show(request):
    room_layout_list = Room_layout.objects.all().filter(publication = True)
    if request.user.is_authenticated:
        return render(request, 'main/room_layout_show.html', {'room_layout':
room_layout_list})
    else:
        return redirect("login")

```

Файл settings.py

```

from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/

```

					ПГУ 1.090301.16.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-b!r)tnt!az=6vl74&kd2m7-a@rzhs8&j@1m+c+y-uv-
f&cz*%b'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'main',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'colorfield',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'museum.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
    },
]

```

```

        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],

WSGI_APPLICATION = 'museum.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
        'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
        'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
        'django.contrib.auth.password_validation.CommonPasswordValidator',
    },

```

```

        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE_CODE = 'ru'

TIME_ZONE = 'Europe/Moscow'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/howto/static-files/

STATIC_URL = 'static/'

STATIC_ROOT = os.path.join(BASE_DIR, 'static')

#STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

# Media files

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# Default primary key field type
# https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

```

Файл wsgi.py

```

import os
from django.core.wsgi import get_wsgi_application
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'museum.settings')
application = get_wsgi_application()

```

Файл manage.py

```
#!/usr/bin/env python
```

```

"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'museum.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()

```

					ПГУ 1.090301.16.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б

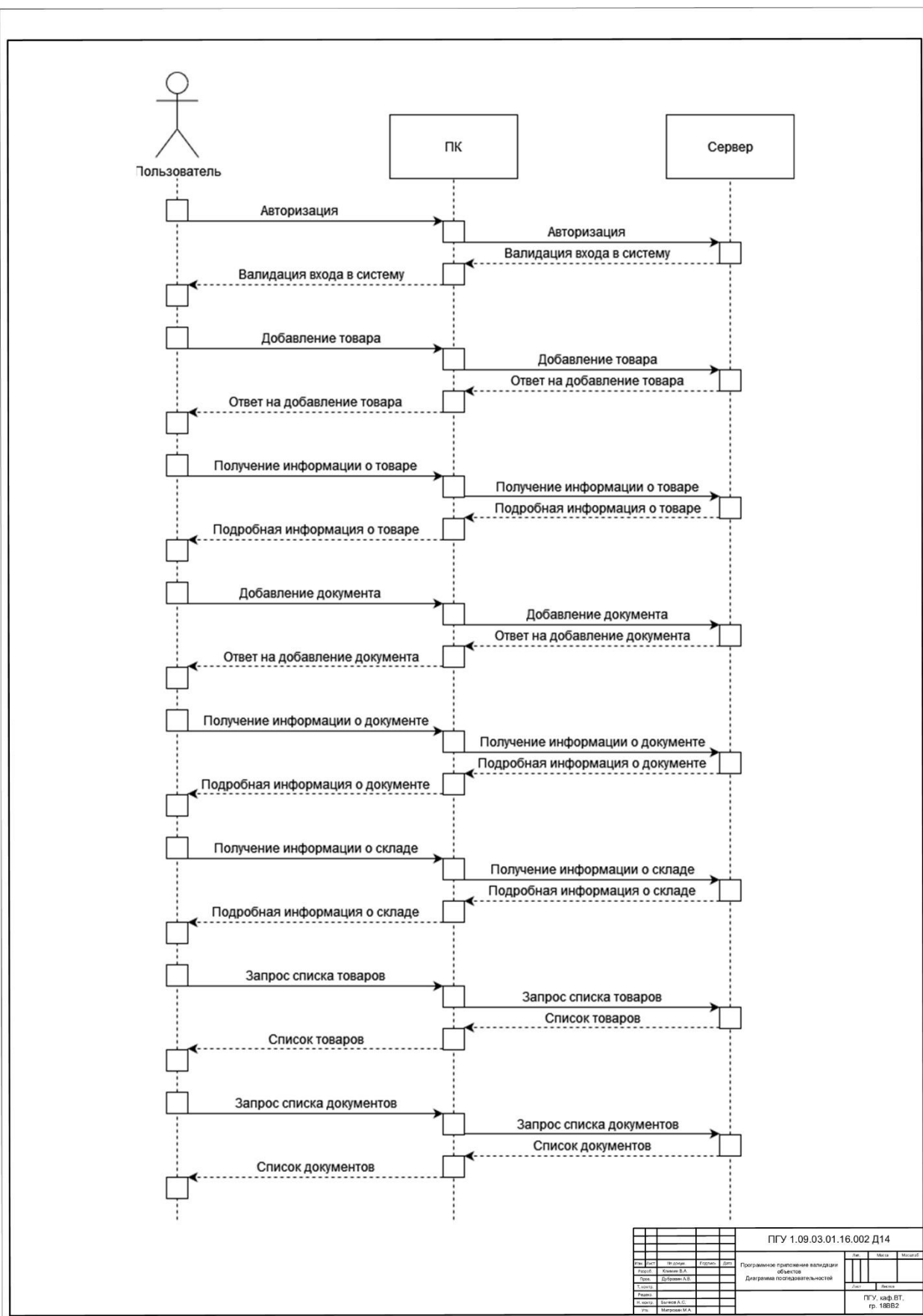
(Обязательное)

Программное приложение валидации объектов

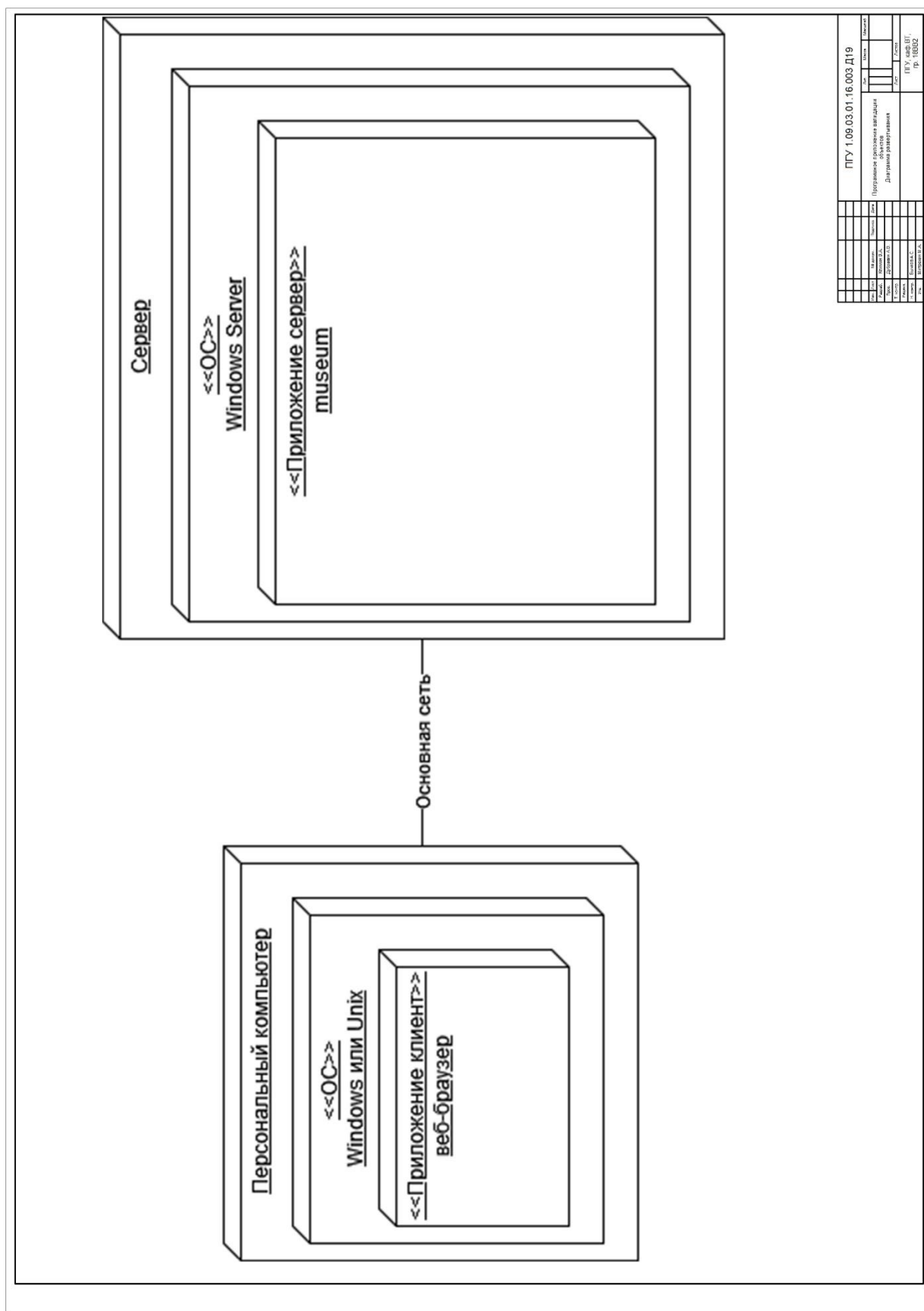
UML-диаграммы

					ПГУ 1.090301.16.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

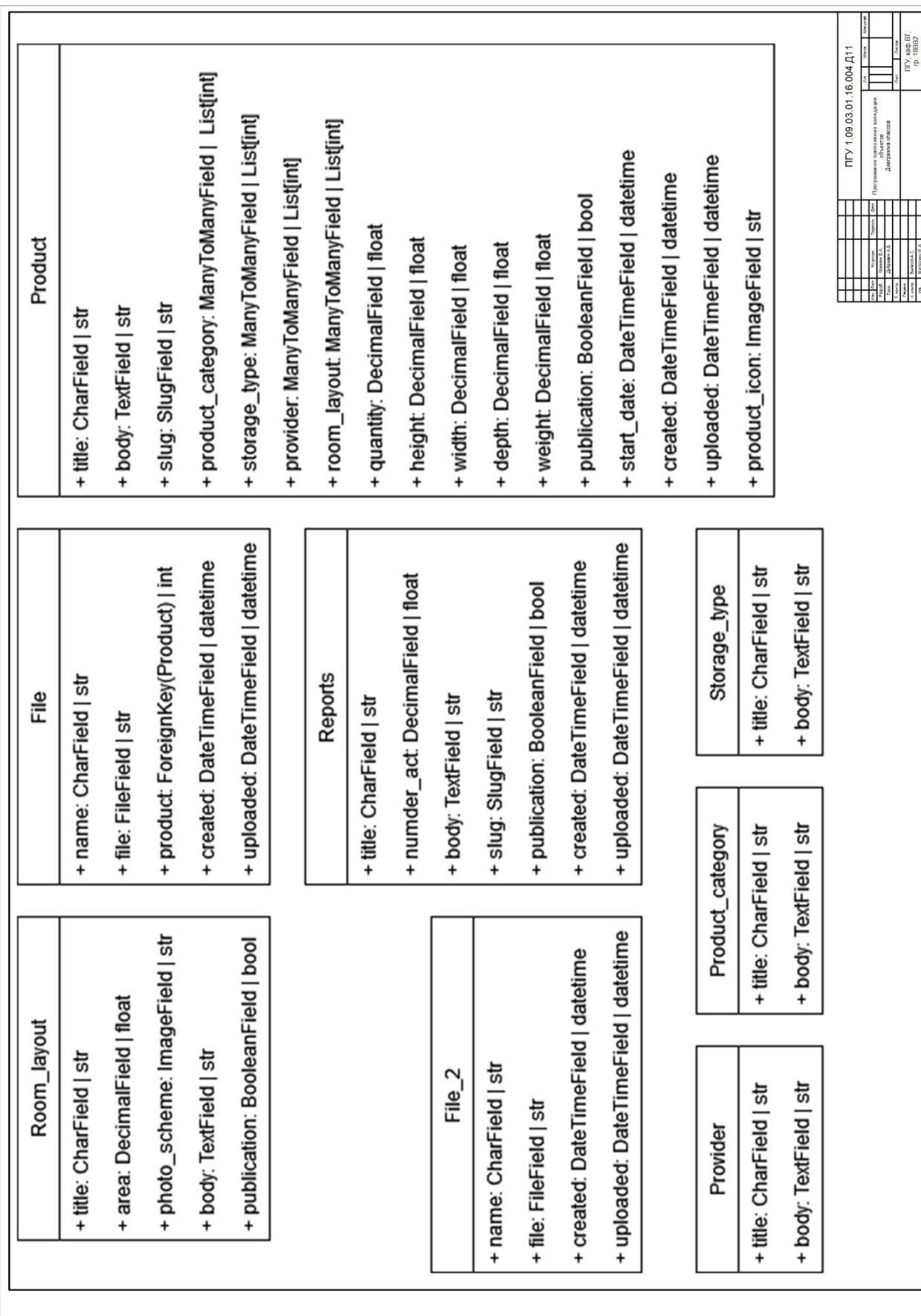
Приложение Б.1 - Диаграмма последовательностей



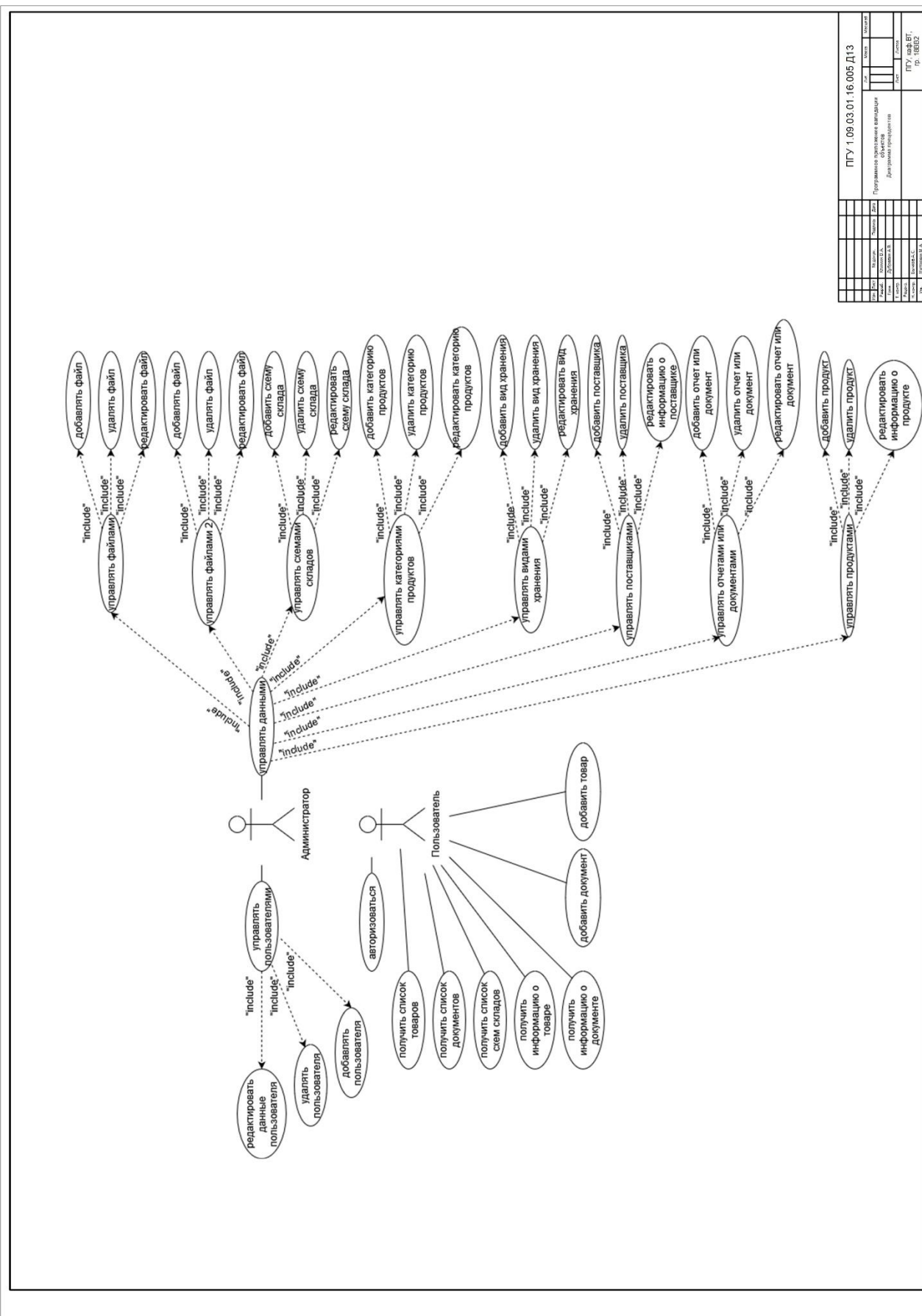
Приложение Б.2 - Диаграмма развертывания



Приложение Б.3 – Диаграмма классов



Приложение Б.4 – Диаграмма прецедентов



Приложение В

(Обязательное)

Программное приложение валидации объектов

Презентация

					ПГУ 1.090301.16.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

Программное приложение валидации объектов

Автор работы:

студент гр. 18ВВ2 Климин Владислав Алексеевич

Научный руководитель:

к.т.н., доцент каф. ВТ ПГУ Дубравин Алексей Викторович

Актуальность

Валидация — это проверка того, что процесс, продукт полностью удовлетворяет заданным требованиям заказчика. К примеру, человек захотел купить планшет, он планирует играть в игры, сидеть в интернете, делать фотографии и т.д. Все это есть в инструкции, но если планшет не открывает приложения, а фотографии не появляются в галереи, то валидация не пройдена.

Актуальность выбранной темы обусловлена потребностью в возможности проверки объектов на установленные требования в целях повышения качества объекта и достоверности в том, что продуктом можно пользоваться по назначению.

					ПГУ 1.090301.16.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

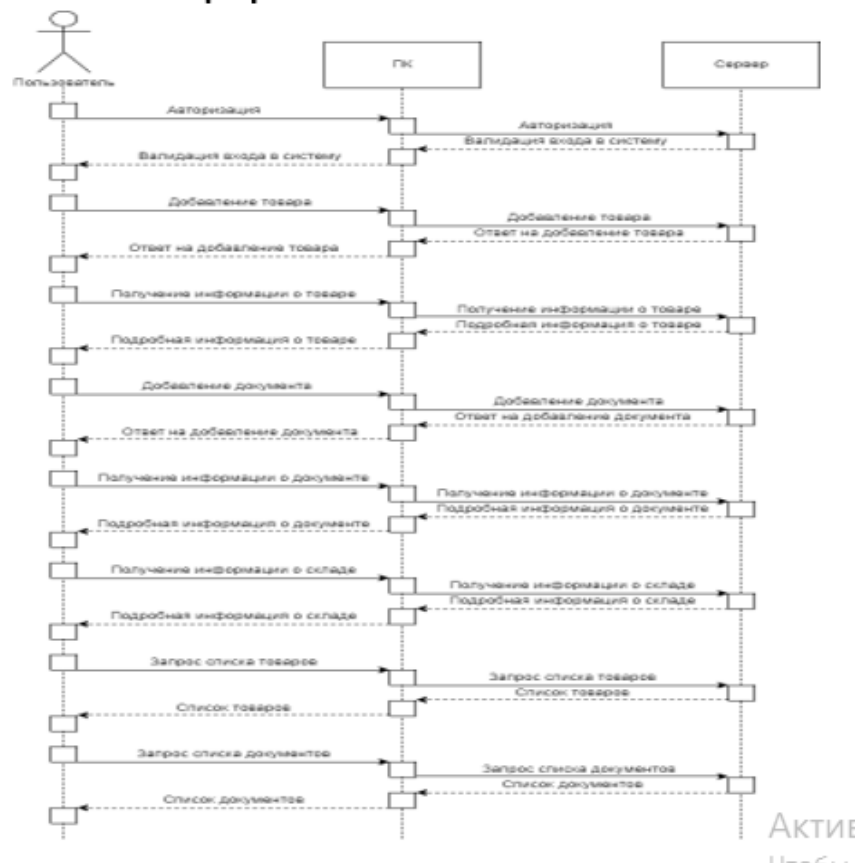
Цель работы и функции приложения

Цель работы – разработать программное приложение, осуществляющее проверку требований к товарам народного потребления перед публикацией информации о них в веб-приложении.

Приложение должно выполнять следующие функции:

- Наличие системы авторизации пользователя;
- Осуществление валидации при добавлении данных в систему;
- Возможность добавления пользователем несколько товаров подряд.

Диаграмма последовательностей



```

    usecaseDiagram
        actor Admin as Администратор
        actor User as Пользователь

        Admin --> UC1(управлять пользователями)
        UC1 -.-> UC1_1(редактировать данные пользователя)
        UC1 -.-> UC1_2(удалить пользователя)
        UC1 -.-> UC1_3(добавлять пользователя)

        Admin --> UC2(управлять данными)
        UC2 -.-> UC2_1(управлять файлами)
        UC2 -.-> UC2_2(управлять файлами 2)
        UC2 -.-> UC2_3(управлять схемами складов)
        UC2 -.-> UC2_4(управлять категориями продуктов)
        UC2 -.-> UC2_5(управлять видами хранения)
        UC2 -.-> UC2_6(управлять поставщиками)
        UC2 -.-> UC2_7(управлять отчетами или документами)
        UC2 -.-> UC2_8(управлять продуктами)

        UC2_1 -.-> UC2_1_1(добавлять файл)
        UC2_1 -.-> UC2_1_2(удалять файл)
        UC2_1 -.-> UC2_1_3(редактировать файл)

        UC2_2 -.-> UC2_2_1(добавлять файл)
        UC2_2 -.-> UC2_2_2(удалять файл)
        UC2_2 -.-> UC2_2_3(редактировать файл)

        UC2_3 -.-> UC2_3_1(добавить схему склада)
        UC2_3 -.-> UC2_3_2(удалить схему склада)
        UC2_3 -.-> UC2_3_3(редактировать схему склада)

        UC2_4 -.-> UC2_4_1(добавить категорию продуктов)
        UC2_4 -.-> UC2_4_2(удалить категорию продуктов)
        UC2_4 -.-> UC2_4_3(редактировать категорию продуктов)

        UC2_5 -.-> UC2_5_1(добавить вид хранения)
        UC2_5 -.-> UC2_5_2(удалить вид хранения)
        UC2_5 -.-> UC2_5_3(редактировать вид хранения)

        UC2_6 -.-> UC2_6_1(добавить поставщика)
        UC2_6 -.-> UC2_6_2(удалить поставщика)
        UC2_6 -.-> UC2_6_3(редактировать информацию о поставщике)

        UC2_7 -.-> UC2_7_1(добавить отчет или документ)
        UC2_7 -.-> UC2_7_2(удалить отчет или документ)
        UC2_7 -.-> UC2_7_3(редактировать отчет или документ)

        UC2_8 -.-> UC2_8_1(добавить продукт)
        UC2_8 -.-> UC2_8_2(удалить продукт)
        UC2_8 -.-> UC2_8_3(редактировать информацию о продукте)

        User --> UC3(авторизоваться)
        User --> UC4(получить список товаров)
        User --> UC5(получить список документов)
        User --> UC6(получить список схем складов)
        User --> UC7(получить информацию о товаре)
        User --> UC8(получить информацию о документе)
        User --> UC9(добавить документ)
        User --> UC10(добавить товар)
        User --> UC11(управлять продуктами)
    
```

Система товаров

Вход

Вход в систему учёта товаров

Имя пользователя:

Пароль:

Вход

Страница ошибки авторизации при неверных учетных данных

Система товаров Вход

Вход в систему учёта товаров
Неверные данные для входа

Имя пользователя:

Пароль:

Вход

Главная страница приложения

Система товаров Выход

Товары	Товар	Фото товара	Дата поставки товара	Количество
Схемы складов хранения	Добавить товар			
Документация				

Страница добавления товара

Система товаров Выход

Новый товар

Название:

Описание:

URL:

Категория товара:

Вид хранения:

Поставщик:

Склад хранения:

Количество кд. товара:

Высота в см. шт:

Ширина в см. шт:

Активация Windows
Чтобы активировать Windows, перейдите
"Параметры".

Ошибка валидации данных при добавлении товара

Система товаров Выход

Новый товар

Название:

Описание:

URL:

Категория товара:

Вид хранения:

Поставщик:

Склад хранения:

Количество ад. товара:

Высота в см. Зшт:

Ширина в см. Зшт:

Активация Windows
Чтобы активировать Windows, перейдите
"Параметры".

Корректная форма заполнения при добавлении товара

Система товаров Выход

Новый товар

Название:

Описание:

URL:

Категория товара:

Вид хранения:

Поставщик:

Склад хранения:

Активация Windows
Чтобы активировать Windows, перейдите
"Параметры".

Страница с подробным описанием товара

Система товаров

Выход

Товары

Схемы складов хранения

Документация

Часы "Зенит"



Дата начала: 11 июня 2022 г. 0:00
Дата закрытия:
Описание:
Тестовое описание часов "Зенит"

Активация Windows
Чтобы активировать Windows, перейдите "Параметры".

Страница списка схем складов хранения

Система товаров

Выход

Товары

Схемы складов хранения

Документация

Название	Площадь	Схема	Описание
Писка - Центр	150,00		Расположен недалеко от здания ТЦ "Павлов"

Страница списка документов

Система товаров

Выход

Товары

Схемы складов хранения

Документация

Название	Номер	Дата создания
Проверка добавления документа	1	11 июня 2022 г. 10:37

Добавить документ

Страница описания документа

Система товаров

Выход

Товары

Схемы складов хранения

Документация

Проверка добавления документа

Документ для проверки системы

Документ создан: 11 июня 2022 г. 10:37

Стартовая страница панели администратора

Администрирование Django

Администрирование сайта

MAIN

Виды хранения	+ Добавить	✎ Изменить
Документы и отчёты	+ Добавить	✎ Изменить
Категории товаров	+ Добавить	✎ Изменить
Общие файлы	+ Добавить	✎ Изменить
Поставщики	+ Добавить	✎ Изменить
Продукты	+ Добавить	✎ Изменить
Схема складов	+ Добавить	✎ Изменить
Файлы	+ Добавить	✎ Изменить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы	+ Добавить	✎ Изменить
Пользователи	+ Добавить	✎ Изменить

Активация Win
Использовать
"Параметры".

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

82

Заключение

В результате данной работы спроектировано, разработано и отлажено программное приложение, предоставляющее инструменты проверки данных о товаре установленным требованиям.

Данное приложение:

- создано на основе высокоуровневого языка программирования Python с использованием фреймворка Django;
- имеет понятный веб-интерфейс;
- обладает возможностью расширения функции валидации товара.

Спасибо за внимание!

					ПГУ 1.090301.16.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		