БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

КУРСОВОЙ ПРОЕКТ

по дисциплине: «Модульное тестирование»

Тема: «Сайт по высшей математике»

|  |  |
| --- | --- |
| Исполнитель: | студент группы 10701119  Маканов Д.В. |
| Руководитель проекта: | Алейникова Е.В |

Минск 2021

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по курсовому проекту

по дисциплине: «Модульное тестирование»

Тема: «Сайт по высшей математике»

|  |  |
| --- | --- |
| Исполнитель: | студент группы 10701119  Маканов Д.В. |
| Руководитель проекта: | Алейникова Е.В |

Минск 2021

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 5](#_Toc90137977)

[1 ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ 7](#_Toc90137978)

[1.1 Общие сведения 7](#_Toc90137979)

[1.2 Аутентификация 7](#_Toc90137980)

[1.3 Пользователь 8](#_Toc90137981)

[1.4 Администратор 10](#_Toc90137982)

[1.5 Гость 16](#_Toc90137983)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ 17](#_Toc90137984)

[2.1 Архитектура программы 17](#_Toc90137985)

[2.2 Проектирование функционала 19](#_Toc90137986)

[3. РЕАЛИЗАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ 21](#_Toc90137987)

[3.1 Реализации моделей классов предметной области 21](#_Toc90137988)

[3.2 Реализация функций представлений моделей 21](#_Toc90137989)

[3.3 Реализация форм 23](#_Toc90137990)

[3.4 Реализация утилитарных классов StatisticsManager и Analizator 25](#_Toc90137991)

[3.5 Реализация функциональности администратора 26](#_Toc90137992)

[4 ПРОЕКТИРОВАНИЕ МОДУЛЬНЫХ ТЕСТОВ 27](#_Toc90137993)

[4.1 Построение ориентированных графом МакКейба 27](#_Toc90137994)

[4.2 Формирование проходов и классов эквивалентностей 29](#_Toc90137995)

[5 РЕАЛИЗАЦИЯ МОДУЛЬНОГО ТЕСТИРОВАНИЯ 33](#_Toc90137996)

[5.1 Реализация модульного тестирования 33](#_Toc90137997)

[5.2 Анализ результатов тестов 34](#_Toc90137998)

[5.3 Отладка тестов 36](#_Toc90137999)

[ЗАКЛЮЧЕНИЕ 37](#_Toc90138000)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc90138001)

[*ПРИЛОЖЕНИЕ А* 39](#_Toc90138002)

[*ПРИЛОЖЕНИЕ B* 46](#_Toc90138003)

# ВВЕДЕНИЕ

Тестирование программного обеспечения — процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.

На протяжении десятилетий развития разработки ПО к вопросам тестирования и обеспечения качества подходили очень и очень по-разному. Можно выделить несколько основных «эпох тестирования». В 50–60-х годах прошлого века процесс тестирования был предельно формализован, отделён от процесса непосредственной разработки ПО и «математизирован». Фактически тестирование представляло собой скорее отладку программ. Существовала концепция т.н. «исчерпывающего тестирования — проверки всех возможных путей выполнения кода со всеми возможными входными данными. Но очень скоро было выяснено, что исчерпывающее тестирование невозможно, т.к. количество возможных путей и входных данных очень велико, а также при таком подходе сложно найти проблемы в документации.

В нулевые годы нынешнего века развитие тестирования продолжалось в контексте поиска всё новых и новых путей, методологий, техник и подходов к обеспечению качества. Серьёзное влияние на понимание тестирования оказало появление гибких методологий разработки и таких подходов, как «разработка под управлением тестированием (test-driven development)». Автоматизация тестирования уже воспринималась как обычная неотъемлемая часть большинства проектов, а также стали популярны идеи о том, что во главу процесса тестирования следует ставить не соответствие программы требованиям, а её способность предоставить конечному пользователю возможность эффективно решать свои задачи.

В современном этапе развития тестирования совершенствуются гибкие методологии и гибкое тестирование, глубокая интеграция с процессом разработки, широкое использование автоматизации, колоссальный набор технологий и инструментальных средств, кроссфункциональность команды (когда тестировщик и программист во многом могут выполнять работу друг друга).[1]

Основные цели тестирования заключаются в повышении вероятности работы приложения корректно при любых обстоятельствах, повышении вероятности соответствия с описанными требованиями, предоставлении актуальной информации о состоянии продукта на данный момент.

В данной курсовой будут продемонстрированы основные концепции модульного тестирования веб-приложения “Сайт по высшей математике”. Пояснительная записка состоит из следующих разделов:

* требования к программному продукту. В данном разделе описаны требования к разрабатываемому приложению;
* проектирование программных модулей. В этом разделе описана модель приложения (его архитектура, интерфейс, функционал);
* реализация программных модулей. В данном разделе описаны методики и алгоритмы работы разработанного функционала;
* проектирование модульных тестов. В этом разделе описаны ориентированные графы МакКейба, проходы по разработанным методам;
* реализация модульного тестирования. В данном разделе описаны автоматизированные тесты, разработанные по графам и проходам.

Целью курсового проекта является разработка веб-приложения “Сайт по высшей математике” и реализация тестирования во всех этапах жизненного цикла программного проекта.

Для реализации программного продукта были использованы язык программирования Python 3.7, ide PyCharm 2021.3, база данных SQLite. Для разработки модульных тестов использовался фреймфорк PyTest.

# 1 ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ

# Общие сведения

В данном документе рассматривается проектируемый программный продукт «Сайт по высшей математике». Система предназначена для изучения математики по разделам в виде лекций и выполнения соответствующих заданий по ним. В данной системе рассматривается 3 основных типа пользователей:

1. Администратор – авторизованный пользователь, который может создавать, редактировать и просматривать разделы с темами, на которые он может выкладывать лекции и задания по математике, а также ограничивать доступ к ним.
2. Пользователь – авторизованный пользователь, который может просматривать разделы и темы, которые разрешены администратором, выполнять задания по темам и проверять таким образом себя.
3. Гость – пользователь, который может просматривать разделы и темы, разрешенные администратором, зарегистрироваться на сайте. Задания по темам данный пользователь выполнять не может.

# Аутентификация

**R2.1. Страница входа в систему**

При переходе по url = «/login» открывается страница для входа в систему, которая содержит такие элементы как:

* Текстовое поле для ввода логина.
* Текстовое поле для ввода пароля.
* Кнопка входа в систему.
* Кнопка “Регистрация” для перехода на страницу регистрации
* Кнопка “На Главную” для перехода на главную страницу сайта

**R2.1.1 Логин**

Логин должен содержать в своем составе любые буквы, цифры и знаки препинания. Длина имени пользователя должна быть от 3 до 10 символов. Логин чувствительно к регистру (например, имена *Dima* и *dima* не будут являться идентичными). Логин должен быть уникален.

Логин является обязательным полем для заполнения.

**R2.1.2 Пароль**

Пароль может содержать любые буквы, цифры и знаки препинания. Длина пароля должна быть от 8 до 20 символов. Прописные и строчные буквы считаются различными.

Пароль является обязательным полем для заполнения.

**R2.3.1 Реакция системы на неверно введенный логин**

Если пользователь ввел неверный логин и нажал на кнопку «Войти», то на экране выдается сообщение об ошибке: «Пожалуйста, введите правильные имя пользователя и пароль. Оба поля могут быть чувствительны к регистру.».

**R2.3.2 Реакция системы на неверно введенный пароль**

Если пользователь ввел неверный пароль и нажал на кнопку «Войти», то на экране выдается сообщение об ошибке: «Пожалуйста, введите правильные имя пользователя и пароль. Оба поля могут быть чувствительны к регистру.».

**R2.3.3 Действие кнопки “Регистрация”**

При нажатии на данную кнопку пользователь переходит на страницу регистрации. Требования к данной странице представлены в разделе 5.

**R2.3.4 Действие кнопки “На Главную”**

При нажатии на данную кнопку пользователь переходит на главную страницу сайта. Требования к данной странице представлены в разделе 3.

**R2.3.5 Вход в систему**

Если логин и пароль введены верно, то по нажатию на кнопку «Войти» пользователь входит в систему на главную страницу сайта.

# Пользователь

Данную часть системы может видеть авторизованный пользователь. В верхней части экрана располагается навигационная панель. Также любой пользователь может выйти из системы.

**R3.1 Отображение навигационной панели**

После успешной аутентификации пользователь переходит на главную страницу, на которой будут отображаться навигационная панель, в которой есть кнопки “Главная”, “О сайте”, имя пользователя, “Выйти”. Данная навигационная панель будет видна на всех страницах, которые посещает пользователь, кроме страниц регистрации и входа на сайт.

**R3.1.1 Реакция системы на нажатие кнопки “Главная” на навигационной панели**

При нажатии данной кнопки пользователь сможет перейти на главную страницу сайта, если он находится на какой-либо другой странице сайта.

**R3.2 Отображение главной страницы**

На главной странице есть список разделов с их названиями и рядом с каждым разделом есть кнопка для перехода по нему. При нажатии на одну из этих кнопок пользователь сможет перейти на страницу с нужным разделом.

**R3.3 Отображение разделов**

На странице с отображением раздела есть название раздела, название каждой темы и рядом с каждой темой есть кнопка для перехода по соответствующей теме. При нажатии на одну из этих кнопок пользователь сможет перейти на страницу с нужной темой.

**R3.4 Отображение тем**

Страница любой темы содержит название данной темы, область для просмотра лекции, которую пользователь может пролистывать в вертикальном направлении, и задания, на которые можно давать ответы прямо на сайте с возможностью их проверки.

**R3.4.1 Описание заданий в теме**

Каждое задание будет реализовано в виде виджета, в котором написан текст задания, есть поле для ввода целых чисел, а также кнопка для проверки ответа на задание.

**R3.4.2 Выполнение заданий в теме**

Задание выполняется посредством ввода целого числа в область ответов. После ввода ответа пользователь должен нажать на кнопку “Проверить”. Также к числу выполненных заданий по теме считается именно число верно выполненных заданий.

**R3.4.3 Требования к вводу ответов в задании**

Вводить можно целые числа до 5 знаков, можно также вводить и отрицательные числа. По умолчанию в поле ответов будет выставлена пустая строка.

**R3.4.4 Реакция системы на слишком длинное число**

Пользователь не может ввести число, большее 5 знаков.

**R3.4.5 Реакция системы на верно выполненное задание**

При нажатии на кнопку проверки ответа, пользователь получит сообщение “Задание выполнено” при верном выполнении задания. Выполнение задания считается верным, если ответ пользователя совпадает с ответом администратора, который создал это задание.

**R3.4.6 Реакция системы на неверно выполненное задание**

При нажатии на кнопку проверки ответа, пользователь получит сообщение “Задание не выполнено” при неверном выполнении задания.

**R3.4.7 Реакция системы на пустой ответ на задание**

При нажатии на кнопку проверки ответа, пользователь получит сообщение “Заполните это поле ответа”.

**R3.4.8 Реакция системы на нажатие кнопки “Назад” в конце страницы темы**

При нажатии на кнопку “Назад” пользователь перейдёт на страницу раздела, где расположена текущая тема.

**R3.5 Отображение пользователя**

Справа на навигационной панели должно располагаться имя пользователя. Имя должно быть в виде ссылки.

**R3.5.1 Страница профиля пользователя**

На странице профиля пользователь может посмотреть статистику прохождения тем на сайте.

**R3.6 Отображение статистики пользователя**

Пользователь может увидеть статистику по выполненным заданиям по каждой теме в виде процента выполненных заданий от общего количества заданий. Также внизу данной страницы будет кнопка для перехода назад на главную страницу, по нажатию которой пользователь сможет перейти на главную страницу.

**R3.7 Выход из системы**

Если пользователь нажмет на «Выйти», то пользователь выйдет из системы. Он перейдет на страницу аутентификации.

# Администратор

Данную часть системы может видеть авторизованный пользователь(администратор) с расширенными возможностями. Он может просматривать/создавать/редактировать/удалять разделы, темы и задания по ним, которые может видеть обычный пользователь.

**R4.1 Главная страница администратора**

Главная страница администратора (url=/admin) будет область просмотра и редактирования категорий сайта: разделов и вложенных в них тем, заданий. В этой области есть ссылки на редактирование соответствующего типа объекта, по нажатию которых администратор может приступить к работе с ним. Также рядом с каждой ссылкой есть кнопки “Добавить” и “Изменить”. Кроме того, на главной странице администрирования есть область просмотра последних действий администратора.

**R4.1.1 Реакция системы на нажатие кнопки «Добавить» рядом с каждой ссылкой на тип редактируемого объекта.**

Если администратор нажал на кнопку «Добавить», то он перейдёт на страницу добавления соответствующего объекта, которая включает в себя все необходимые поля для заполнения и добавления данного типа объекта.

**R4.1.2 Реакция системы на нажатие кнопки «Изменить» рядом с каждой ссылкой на тип редактируемого объекта.**

Если администратор нажал на кнопку «Изменить», то он перейдёт на страницу просмотра всех объектов заданного типа. На этой странице пользователь выбирает нужный объект для изменения, после чего он переходит на страницу редактирования объекта.

**R4.1.3 Общие требования к странице создания/редактирования выбранного объекта.**

При создании или редактировании объекта не зависимо от типа объекта на странице редактирования будут расположены следующие элементы: кнопка “Сохранить и добавить другой объект”, кнопка “Сохранить и продолжить редактирование”, кнопка “ Сохранить”, ссылки на категории сайта (темы, разделы, задания, пользователи) и поля для редактирования объекта. На странице редактировании объекта будет расположена кнопка “Удалить”.

**R4.1.3.1 Реакция системы на нажатие кнопки «Сохранить».**

При нажатии данной кнопки в случае верного заполнения всех данных система сохранит объект и переведёт пользователя на страницу просмотра всех объектов заданной категории. Требования верного заполнения всех полей представлены в пунктах R4.2-4.4.

**R4.1.3.2 Реакция системы на нажатие кнопки «Сохранить и продолжить редактирование».**

При нажатии данной кнопки в случае верного заполнения всех данных система сохранит объект и переведёт пользователя на текущую страницу редактирования данного объекта.

**R4.1.3.3 Реакция системы на нажатие кнопки «Сохранить и добавить другой объект».**

При нажатии данной кнопки в случае верного заполнения всех данных система сохранит объект и переведёт пользователя на страницу создания нового объекта той же категории.

**R4.1.3.4 Реакция системы на нажатие кнопки «Удалить» на странице редактирования объекта заданного типа.**

При нажатии данной кнопки система удалит объект и переведёт пользователя на страницу просмотра всех объектов заданной категории.

**R4.2 Создание и редактирование разделов**

В данной области администратор может создавать и редактировать разделы. На данной панели отображается список всех разделов, а также есть кнопка “Добавить”.

**R4.2.1 Добавление раздела**

Если администратор нажмет на кнопку «Добавить», то в начале соответствующей области появится поле для ввода названия раздела, галочка для публикации раздела, поля для ввода url раздела и url картинки для отображения раздела на главной странице сайта.

**R4.2.1.1 Требования к названию раздела**

Администратор может ввести название раздела от 1 до 100 знаков, причём строчные и прописные буквы в названии темы считаются различными.

**R4.2.1.2 Реакция системы на слишком длинное название раздела**

Администратор не сможет ввести название раздела, которое длиннее 100 символов, так как ввод на большее количество символов будет недоступен.

**R4.2.1.3 Реакция системы на слишком короткое название раздела**

Если название раздела будет меньше 1 символа, то администратор получит сообщение об обязательном вводе названия раздела.

**R4.2.1.4 Реакция системы на неуникальное название раздела**

Администратор имеет право на создание раздела с таким же названием.

**R4.2.1.5 Требования к изменению названия раздела**

Администратор может изменить название раздела, кликнув по названию раздела. Требования к вводу названия раздела описаны в пунктах R4.2.1.1-R4.2.1.4.

**R4.2.2.1 Требования к полю ввода url раздела**

Администратор может ввести url раздела для его определения в адресной строке браузера от 1 до 255 знаков латинского языка, цифр, знаков дефиса и нижнего подчеркивания, причём строчные и прописные буквы в названии темы считаются различными.

**R4.2.2.2 Реакция системы на слишком длинный url раздела**

Администратор не сможет ввести название раздела, которое длиннее 255 символов, так как ввод на большее количество символов будет недоступен.

**R4.2.2.3 Реакция системы на слишком короткое url раздела**

Если название раздела будет меньше 1 символа, то администратор получит сообщение об обязательном вводе url.

**R4.2.2.4 Требования к изменению url раздела**

Администратор может изменить url раздела, кликнув по url раздела. Требования к вводу url раздела описаны в пунктах R4.2.2.1-R4.2.2.3.

**R4.2.3.1 Требования к полю ввода url картинки раздела**

Администратор может ввести url картинки раздела для его визуального определения на главной странице сайта от 1 до 255 знаков латинского языка, цифр, знаков дефиса и нижнего подчеркивания, причём строчные и прописные буквы в названии темы считаются различными.

**R4.2.3.2 Реакция системы на слишком длинный url картинки раздела**

Администратор не сможет ввести название раздела, которое длиннее 255 символов, так как ввод на большее количество символов будет недоступен.

**R4.2.3.3 Реакция системы на слишком короткое url картинки раздела**

Если название раздела будет меньше 1 символа, то администратор получит сообщение об обязательном вводе url.

**R4.2.3.4 Требования к изменению url картинки раздела**

Администратор может изменить url картинки раздела, кликнув по url картинки раздела. Требования к вводу url картинки раздела описаны в пунктах R4.2.3.1-R4.2.3.3.

**R4.2.4.1 Требования к описанию раздела**

Администратор может ввести описание раздела от 1 до 100 знаков, причём строчные и прописные буквы в описании темы считаются различными.

**R4.2.4.2 Реакция системы на слишком длинное название раздела**

Администратор не сможет ввести описание раздела, которое длиннее 100 символов, так как ввод на большее количество символов будет недоступен.

**R4.2.4.3 Реакция системы на слишком короткое описание раздела**

Если описание раздела будет меньше 1 символа, то администратор получит сообщение об обязательном вводе описания раздела.

**R4.2.4.4 Требования к изменению названия раздела**

Администратор может изменить описание раздела, кликнув по описанию раздела. Требования к вводу описаны в пунктах R4.2.4.1-R4.2.4.3.

**R4.2.5 Публикация раздела**

Пользователь может нажать на галочку для публикации раздела на сайте, после чего он станет доступным пользователю для просмотра. По умолчанию галочка включена на публикацию.

**R4.2.5.1 Нажатие на неотмеченную галочку публикации**

Если администратор нажмёт на неотмеченную галочку, то после этого раздел будет доступен пользователям сайта.

**R4.2.5.2 Нажатие на отмеченную галочку публикации**

Если администратор нажмёт на отмеченную галочку, то после этого раздел не будет доступен пользователям сайта.

**R4.3 Создание и редактирование темы**

В данной области администратор может создавать и редактировать темы. На данной панели отображается список всех разделов, а также есть кнопка “Добавить”.

**R4.3.1** **Добавление темы**

Если администратор нажмет на кнопку «Добавить», то в начале соответствующей области появится поле для ввода названия темы, поле для набора лекции, поля для ввода url темы и выбор раздела.

**R4.3.2.1 Требования к названию темы**

Пользователь может дать название темы в специальном поле, которое расположено в начале страницы темы.

Название темы может содержать латинские или русские буквы, цифры и знаки препинания. Длина названия темы должна быть от 1 до 255 символов. Прописные и строчные буквы считаются различными.

Название темы является обязательным полем для заполнения.

**R4.3.1.2 Реакция системы на слишком длинное название темы**

Администратор не сможет ввести название темы, которое длиннее 255 символов, так как ввод на большее количество символов будет недоступен.

**R4.3.1.3 Реакция системы на слишком короткое название темы**

Если название раздела будет меньше 1 символа, то администратор получит сообщение об обязательном вводе названия темы.

**R4.3.1.4 Реакция системы на неуникальное название темы**

Администратор имеет право на создание темы с таким же названием.

**R4.3.1.5 Требования к изменению названия темы**

Администратор может изменить название темы, кликнув по названию темы. Требования к вводу названия темы описаны в пунктах R4.3.1.1-R4.3.1.4.

**R4.3.2.1 Требования к полю ввода url темы**

Администратор может ввести url темы для его определения в адресной строке браузера от 1 до 255 знаков латинского языка, цифр, знаков дефиса и нижнего подчеркивания, причём строчные и прописные буквы в названии темы считаются различными.

**R4.3.2.2 Реакция системы на слишком длинный url темы**

Администратор не сможет ввести название раздела, которое длиннее 255 символов, так как ввод на большее количество символов будет недоступен.

**R4.3.2.3 Реакция системы на слишком короткое url темы**

Если название темы будет меньше 1 символа, то администратор получит сообщение об обязательном вводе url.

**R4.3.2.4 Требования к изменению url темы**

Администратор может изменить url темы, кликнув по url раздела. Требования к вводу описаны в пунктах R4.3.2.1-R4.3.2.3.

**R4.3.3 Требования к выкладыванию лекционных материалов на сайт**

Администратор может написать лекцию в виде текста в специальное поле, длина которого может быть произвольной, но ненулевой**.** Символы могут быть различными.

**R4.3.4 Определение принадлежности темы к разделу**

Администратор определяет принадлежность темы к разделу через выбор разделов из списка, в котором отображены все существующие разделы. Причем пользователь имеет право не определять тему к разделу. В этом случае пользователь не сможет увидеть данную тему и заданий по ней.

**R4.3.5 Публикация темы**

Пользователь может нажать на галочку для публикации темы на сайте, после чего она станет доступной пользователю для просмотра. По умолчанию галочка включена на публикацию.

**R4.3.5.1 Нажатие на неотмеченную галочку публикации**

Если администратор нажмёт на неотмеченную галочку, то после этого тема будет доступна пользователям сайта.

**R4.3.5.2 Нажатие на отмеченную галочку публикации**

Если администратор нажмёт на отмеченную галочку, то после этого тема не будет доступна пользователям сайта.

**R4.4 Требования к заданиям по теме**

В данной области администратор может создавать и редактировать задания. На данной панели отображается список всех заданий, а также есть кнопка “Добавить”.

**R4.4.1.1 Требования к описанию задания**

Описание задания может содержать латинские или русские буквы, цифры и знаки препинания. Длина описания задания должна быть от 1 до 255 символов. Прописные и строчные буквы считаются различными.

Описание задания является обязательным полем для заполнения.

**R4.4.1.2 Реакция системы на слишком длинное описание задания**

Администратор не сможет ввести описание задания, которое длиннее 255 символов, так как ввод на большее количество символов будет недоступен.

**R4.4.1.3 Реакция системы на слишком короткое описание задания**

Если описание задания будет меньше 1 символа, то администратор получит сообщение об обязательном вводе названия темы.

**R4.4.1.4 Реакция системы на неуникальное описание задания**

Администратор имеет право на создание описания задания с таким же названием.

**R4.4.1.5 Требования к изменению описание задания**

Администратор может изменить описание задания, кликнув на соответствующее поле. Требования к вводу описаны в пунктах R4.4.1.1-R4.4.1.4.

**R4.4.2.1 Требования к ответу на задание**

Ответ может содержать латинские или русские буквы, цифры и знаки препинания. Длина ответа должна быть от 1 до 5 символов. Прописные и строчные буквы считаются различными.

Ответ является обязательным полем для заполнения.

**R4.4.2.2 Реакция системы на слишком длинный ответ**

Администратор не сможет ввести правильный ответ, который длиннее 5 символов, так как ввод на большее количество символов будет недоступен.

**R4.4.2.3 Реакция системы на слишком короткий ответ**

Если ответ будет меньше 1 символа, то администратор получит сообщение об обязательном вводе поля.

**R4.4.2.4 Требования к изменению ответа**

Администратор может изменить ответ задания, кликнув на соответствующее поле. Требования к вводу описаны в пунктах R4.4.2.1-R4.4.2.3.

**R4.4.3 Определение принадлежности задания к теме**

Администратор определяет принадлежность задания к теме через выбор тем из списка, в котором отображены все существующие темы. Причем пользователь имеет право не определять задание к теме. В этом случае пользователь не сможет выполнить это задание.

**R4.5 Доступ администратора ко всем компонентам для обычного пользователя**

Перейдя в режим просмотра, администратор может выполнять все действия обычного пользователя, требования к которым описаны в пунктах **R3.1-R3.7**.

* 1. **Гость**

Гость имеет возможность просматривать разделы и темы, разрешенные администратором, зарегистрироваться на сайте. Задания по темам данный пользователь выполнять не может.

**R5.1 Регистрация**

Переход на страницу регистрации осуществляется на странице аутентификации или с главной страницы сайта. На этой странице пользователь может создать свою учетную запись и заполняет поля нового логина и пароля, повтора пароля. Кроме того, у гостя есть возможность перейти на главную страницу со страницы регистрации.

**R5.1.1 Логин**

Требования к логину представлены в пунктах R2.1.1.

**R5.1.2 Пароль**

Требования к паролю представлены в пунктах R2.1.1.

**R5.1.3 Реакция системы на нарушение требований пароля**

В случае ввода пароля, который не соответствует требованиям будет выведено соответствующее сообщение, что пароль не соответствует требованиям и будет отображен список требований к паролю.

**R5.1.4 Реакция системы на ввод существующего логина**

Если пользователь ввел существующий логин и нажал на кнопку «Зарегистрироваться», то на экране выдается сообщение об ошибке: «Пользователь с таким именем уже существует».

**R5.1.5 Реакция системы на нажатие кнопки “Регистрация”**

Если пользователь ввел корректные данные, то он сможет нажать на данную кнопку, будет выведено сообщение об успешной регистрации. После это будет осуществлен переход на страницу аутентификации сайта.

**R5.1.6 Реакция системы на нажатие кнопки “На главную”**

При нажатии кнопки “На Главную” гость перейдет на главную страницу сайта.

**R5.2 Доступ гостя ко всем компонентам для обычного пользователя кроме заданий**

Гость может выполнять все действия обычного пользователя кроме выполнения заданий и просмотра статистики по заданиям, требования к которым описаны в пунктах **R3.1-R3.4**.

# 2 ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

# 2.1 Архитектура программы

Для реализации программного продукта было решено использовать язык Python 3.7 и фреймворк Django.

Для разработки модульных тестов используются тестовый фреймворк PyTest.

Операционная система для разработки приложения — Windows 11.

Среда для разработки — PyCharm 2021.3.

База данных, используемая для разработки —SQLite.

В Django встроен механизм взаимодействия с таблицами БД через объекты классов языка Python посредством технологии ORM (Object-Relational Mapping). Это означает, что UML- диаграмма классов приложения полностью отражает структуру таблиц в базе данных. Кроме того, любое Django-приложение не привязано к конкретному типу базы данных. На рисунке 2.1 изображена UML - диаграмма приложения.

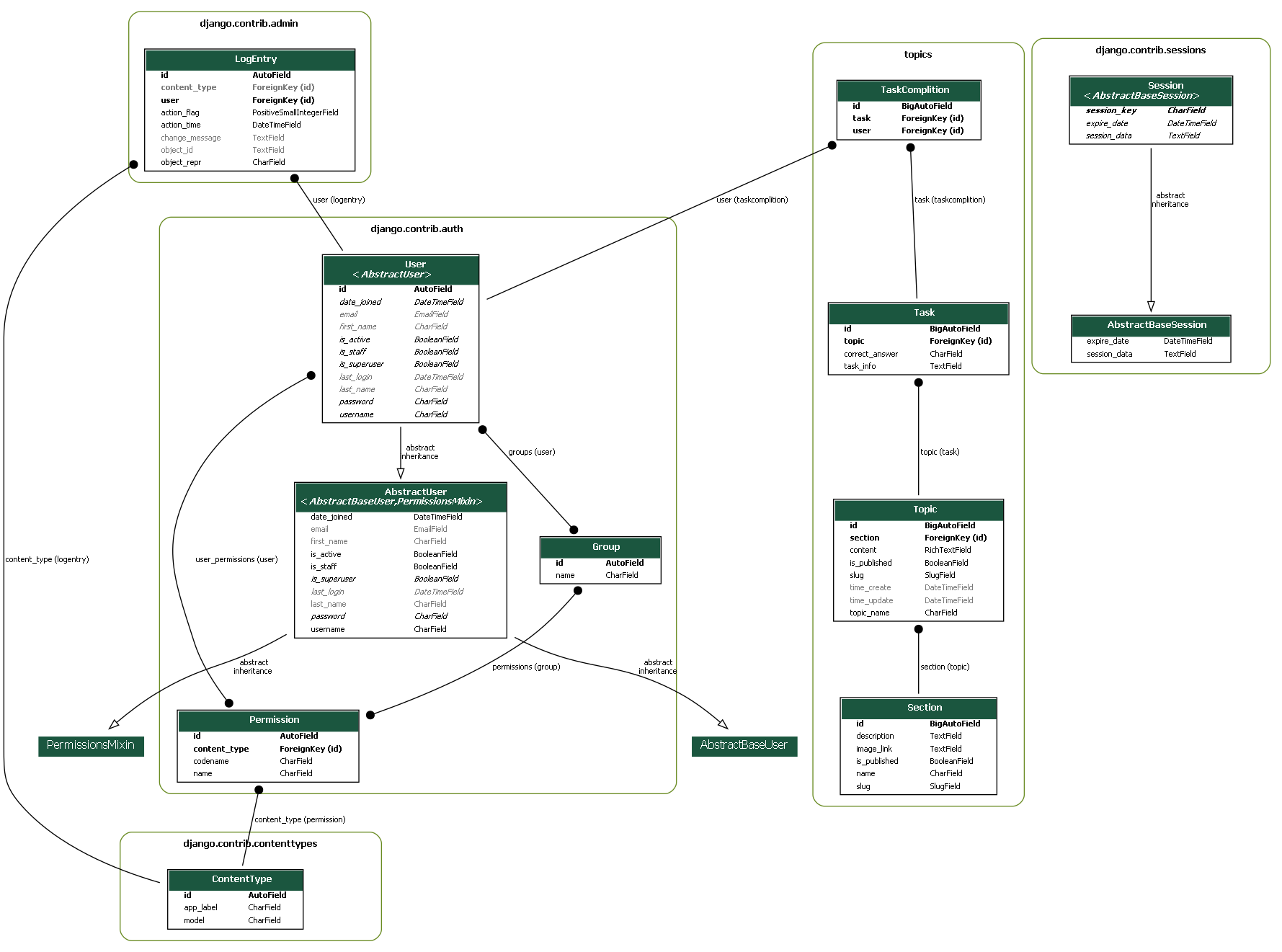


Рисунок 2.1 – UML - диаграмма приложения

На рисунках 2.2-2.4 изображены user-case диаграммы приложения.

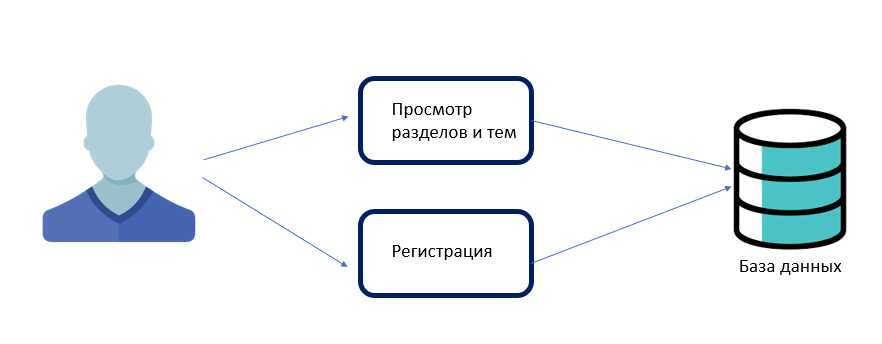


Рисунок 2.2 – User-case диаграмма незарегистрированного пользователя

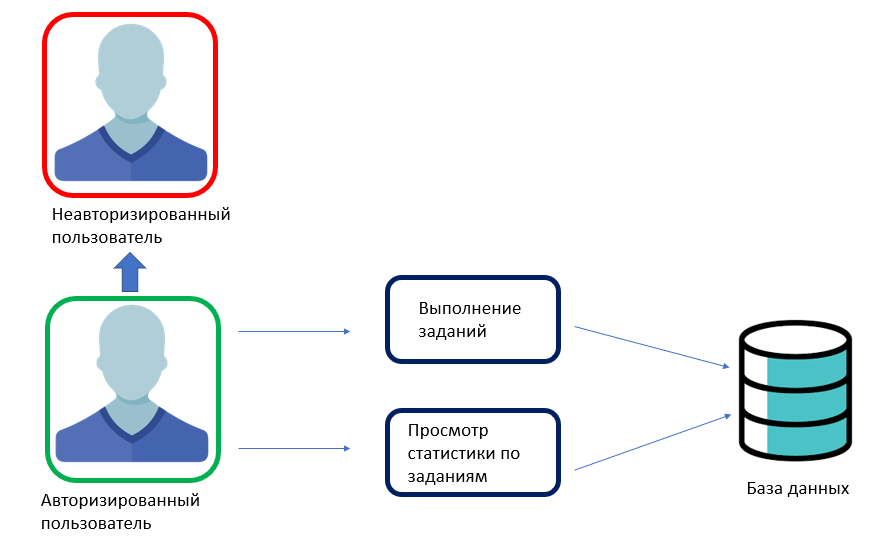


Рисунок 2.3 – User-case диаграмма зарегистрированного пользователя

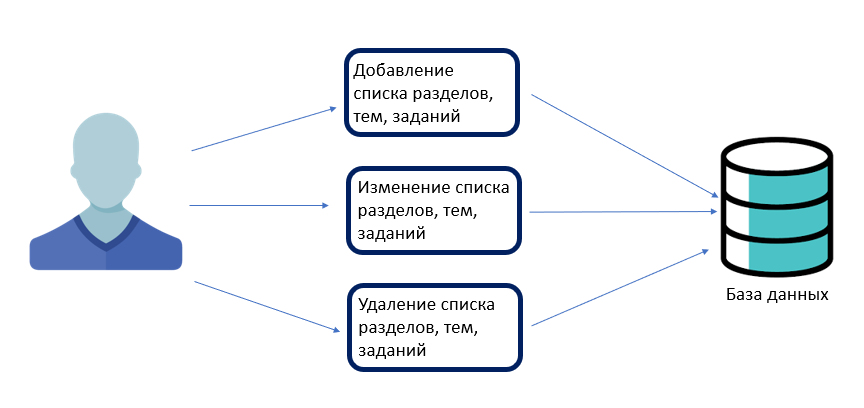


Рисунок 2.4 – User-case диаграмма администратора

Основной механизм приложения имеет следующие шаги. Вначале, когда запрос приходит на фреймворк, он пропускается через блок маршрутизации. Здесь фиксируется тип URL-адреса и в списке шаблонов предопределенных адресов ищется первое совпадение. Вообще, может быть прописано множество типов URL-адресов и каждый адрес связан со своим обработчиком – представлением. Если текущий запрос от пользователя не совпал ни с одним предопределенным URL, то возвращается код ошибки 404 – страница не найдена.

Предположим, что маршрутизатор нашел совпадение. Далее, активизируется представление, связанное с найденным типом URL-адреса. Представление (иногда его еще называют контроллером) – это или функция или класс, который отвечает за формирование ответа на соответствующий запрос. Как правило, ответом является HTML-страница. Эта страница, затем, возвращается клиенту, и он ее видит в браузере. Чтобы контроллер мог сформировать страницу, в общем случае, требуются данные,  шаблоны, в которые эти данные упаковываются. Далее активизируется соответствующее представление, которое берет шаблон информационной страницы и наполняет ее данными, хранящиеся в БД. На выходе получаем сформированную HTML-страницу, которая и возвращается пользователю. Такое разделение на данные (model), шаблоны (templates) и представления (views) представляет собой общеизвестный паттерн MTV, то есть, разделение данных и HTML-шаблонов. Кратко данный механизм проиллюстрирован на рисунке 2.5.[2]

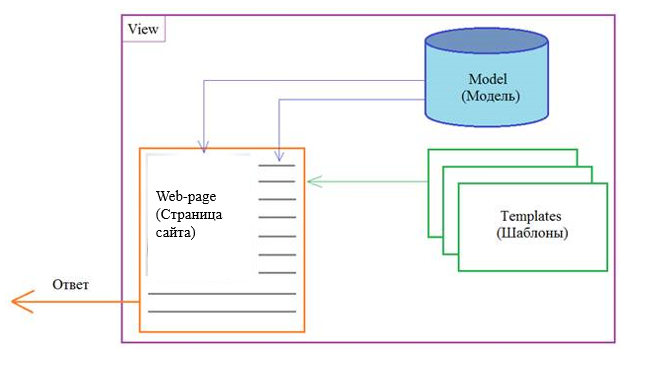
­

Рисунок 2.5 – Схема работы приложения на Django

# 2.2 Проектирование функционала

Для реализации основной модели и функций сайта по высшей математике необходимо реализовать следующие программные сущности и дополнительный функционал приложения:

* класс Topic для работы с темами сайта, включающими в себя лекционный материал и название темы;
* класс Section для группировки тем;
* класс Task для определения заданий в системе и сохранением правильного ответа на задание; в этом классе также будет реализована возможность проверки получаемого ответа и правильного ответа на задание;
* класс Analizator, включающий в себя функциональные методы сортировки данных, корректности вводимых строк;
* класс StatisticsManager для реализации статических методов получения статистики по теме или разделу.

Классы пользователя, администратора, административной панели, взаимодействия серверной части приложения и базы данных, регистрация и авторизация пользователей уже реализованы в Django. Единственным, что необходимо выполнить программисту, является подключение всего этого функционала в общий проект.

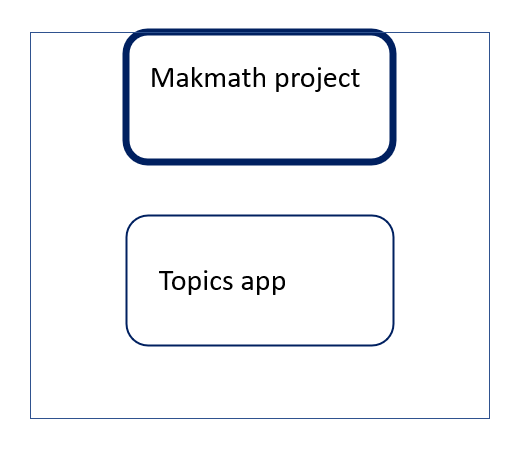
Кроме того, важно отметить про концепцию приложений в Django. Сам проект Django может иметь несколько приложений. Основной функционал сайта будет расположен в приложении topics. Общая структура проекта изображена на рисунке 2.6.

Рисунок 2.6 – Структура проекта на Django

# 3. РЕАЛИЗАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ

Для реализации поставленной задачи необходимо дополнить соответствующие модули Django-приложения пользовательским кодом:

* реализация ORM-моделей классов предметной области Topic, Section, Task в модуле modеls.py;
* реализация функций представлений основных моделей программы в модуле views.py;
* реализация форм регистрации пользователя и ввода ответа на задание в модуле forms.py.
* реализация утилитарных классов StatisticsManager и Analizator.

Подробный код данных реализаций находится в приложении А.

# 3.1 Реализации моделей классов предметной области

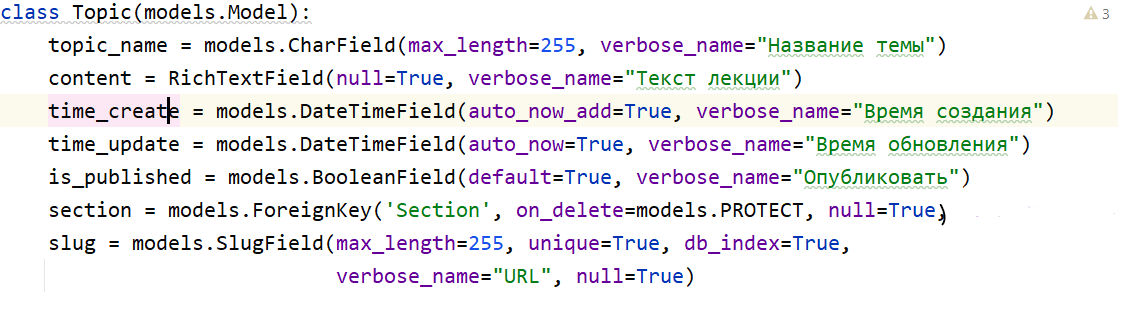
 Сама реализация моделей происходит в модуле models.py приложения. Например, для реализации модели Topic необходимо описать класс с таким названием, а также указать поля этого класса и проинициализировать их специальными классами, отвечающими за ввод соответствующих данных в админ-панели. После создания всех моделей необходимо выполнить все миграции и сохранить изменения в моделях Django-приложения. Пример реализации класса Topic представлен на рисунке 3.1.

Рисунок 3.1 – реализация класса Topic

# 3.2 Реализация функций представлений моделей

Благодаря функциям представлениям Django может отобразить html-шаблон приложения. Данные функции реализуются в модуле views.py. Ключевой особенностью данных функций является аргумент request функции и возврат результата функции render, которой на вход подается относительный путь шаблона. В такие функции можно также отправлять будущее содержимое страницы. Если в коде шаблона есть код, который понятен встроенному шаблонизатору Django, то он вставит соответствующее содержимое в шаблон. Пример функции представления показан на рисунке 3.2, пример кода шаблонизатора в шаблоне страницы показан на рисунке 3.3. Результаты исполнения страниц сайта “О сайте”, темы сайта, раздела сайта показаны на рисунках 3.4, 3.5, 3.6 соответственно.

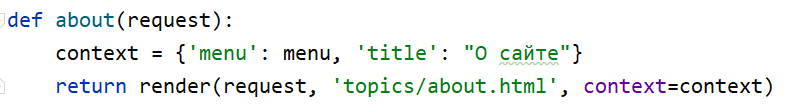


Рисунок 3.2 – реализация функции представления страницы сайта “О сайте”

Рисунок 3.3 – пример код для шаблонизатора в шаблоне страницы сайта

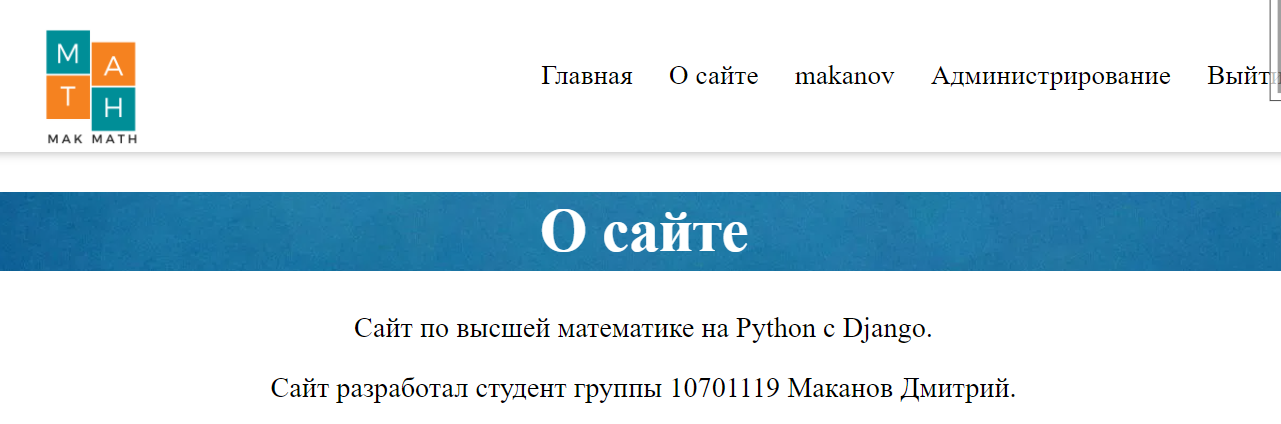


Рисунок 3.4 – отображение страницы сайта “О сайте”

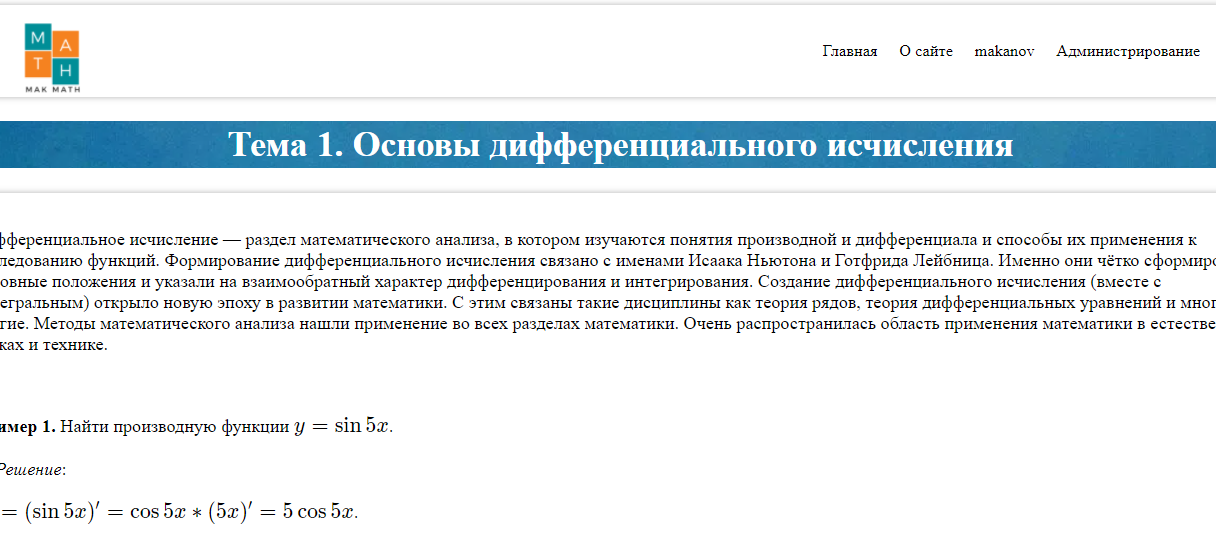


Рисунок 3.5 – отображение темы сайта

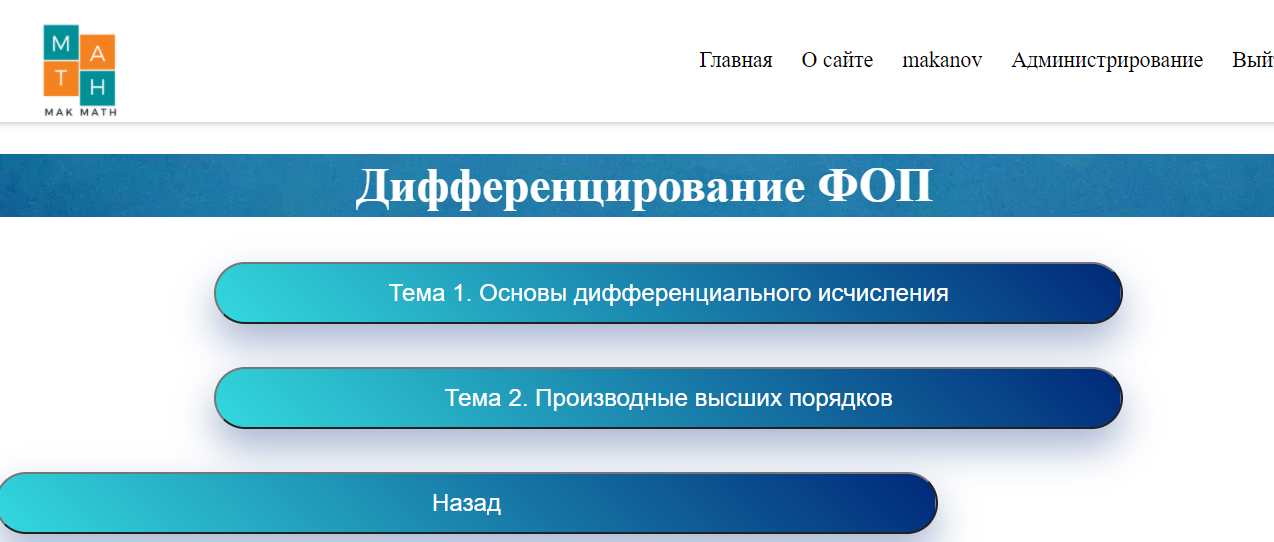
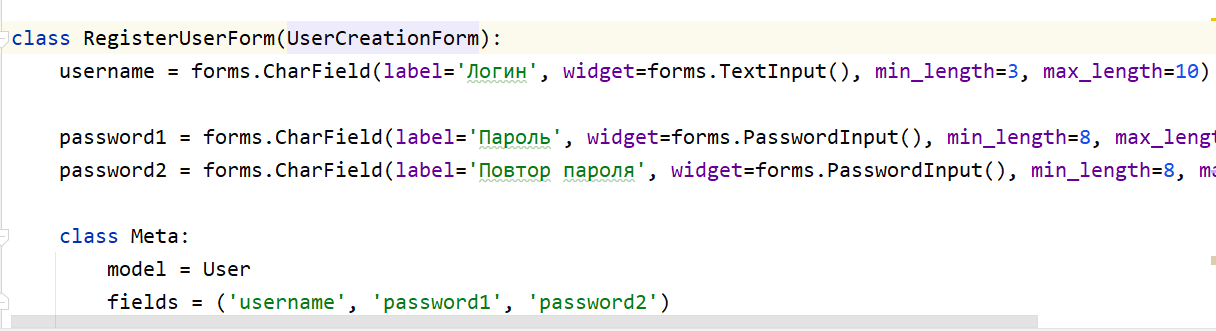


Рисунок 3.6 – отображение раздела сайта

# 3.3 Реализация форм

Django поддерживает создание форм через классы форм. Объекты этих классов передаются в шаблон через функцию представления. При этом само место вставки формы должно быть указано в шаблоне. Далее форма преобразуется в html-код и вставляется в страницу сайта. Код реализации формы регистрации показан на рисунке 3.7. В данном коде идет прямое наследование функционала класса UserCreationForm, который поставляется вместе с Django. Отображение данной формы на сайте представлено на рисунке 3.8.



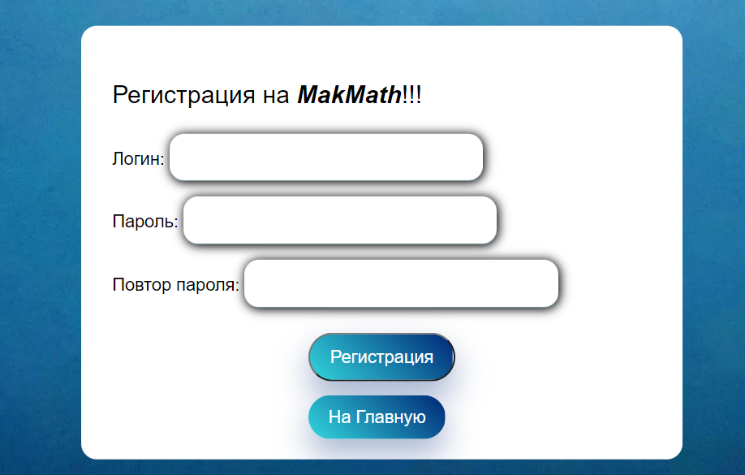
Рисунок 3.7 – код класса формы регистрации

Рисунок 3.8 – Форма регистрации

Аналогичным образом была реализована форма по выполнению заданий, её отображение представлено на рисунке 3.9.

Рисунок 3.9 – Форма выполнения заданий

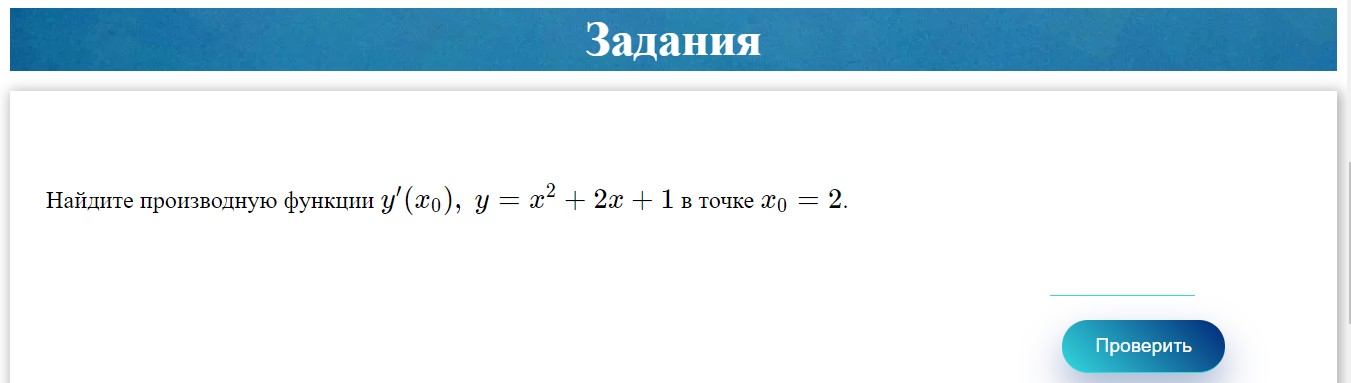
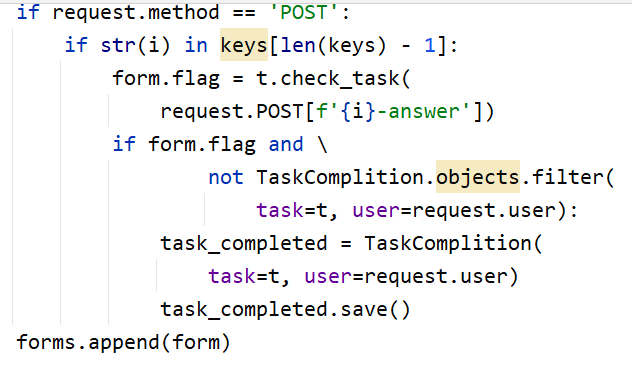
Важно также отметить, что данные формы обрабатываются при помощи метода POST. Как пример, реализация обработки формы по выполнению заданий представлена на рисунке 3.10.

Рисунок 3.10 – Код обработки формы выполнения заданий

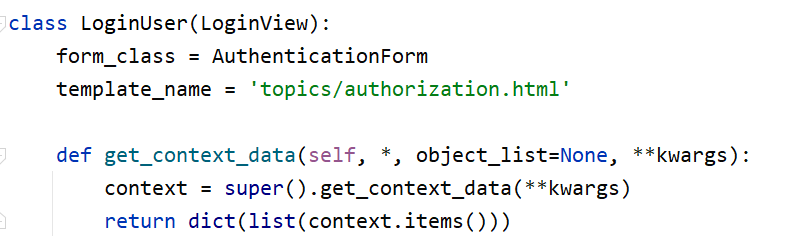
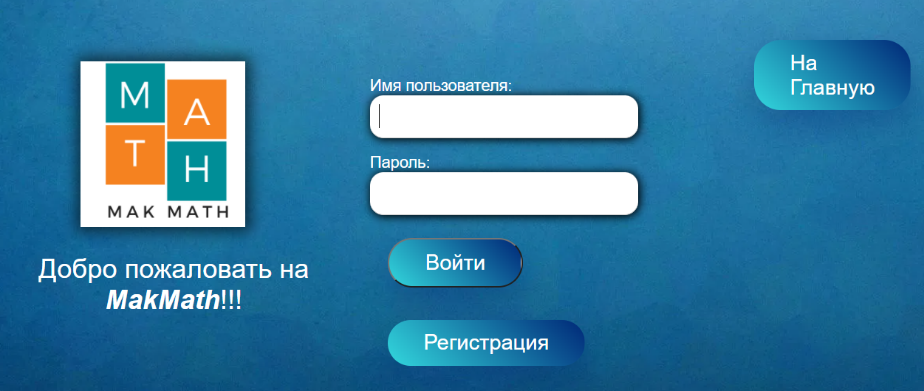
Реализация формы авторизации и обработка аутентификации, авторизации пользователей поставляется вместе с Django. Программисту остается лишь связать форму и шаблон для исполнения такой функциональности в соответствующем классе представления, код которого представлен на рисунке 3.11. Сама форма входа на сайт представлена на рисунке 3.12.

Рисунок 3.11 – Код обработки формы выполнения заданий

Рисунок 3.12 – Форма входа на сайт

# 3.4 Реализация утилитарных классов StatisticsManager и Analizator

Необходимые вспомогательные методы были объединены в классы StatisticsManager и Analizator. Функционал Analizator:

* метод is\_correct\_string(text, min\_length, max\_length) для проверки строки на заданный алфавит и пределы размера;
* метод find\_index\_of\_value\_in\_array(value, array) для нахождения элемента в списке по значению;
* метод def insertion\_sort(data) для сортировки численных данных из списка.

Функционал StatisticsManager:

* метод get\_statistics\_for\_topic(topic, user) для получения статистики пользователя по заданной теме;
* метод get\_statistics\_for\_section(section, user) для получения статистики пользователя по заданному разделу;
* метод get\_statistics\_in\_html(user) для получения статистики пользователя по по всем темам и разделам для вставки в шаблон.

Реализации этих функциональностей находятся в приложении А под заголовком utils.py. Изображение статистики пользователя представлено на рисунке 3.13.

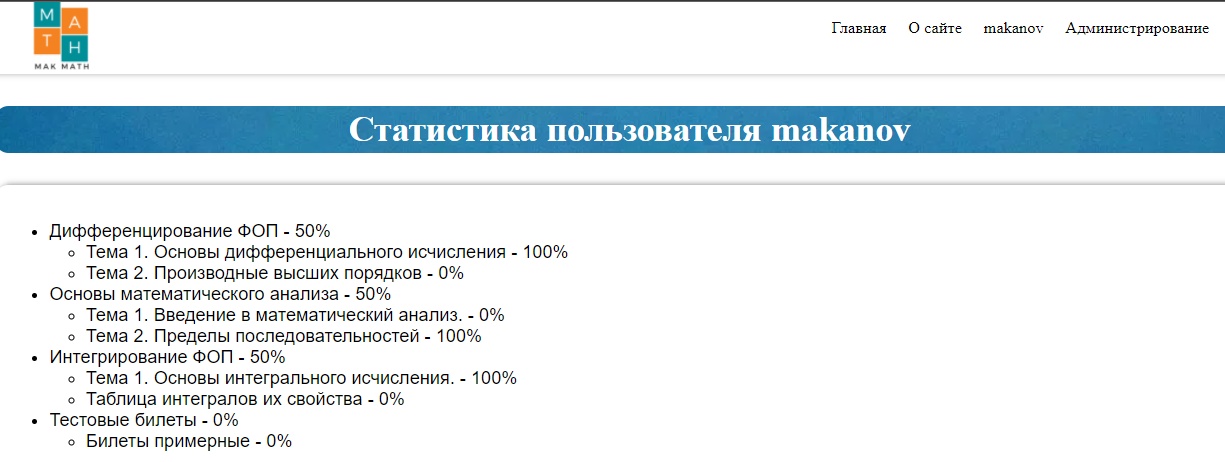


Рисунок 3.13 – Статистика пользователя

# 3.5 Реализация функциональности администратора

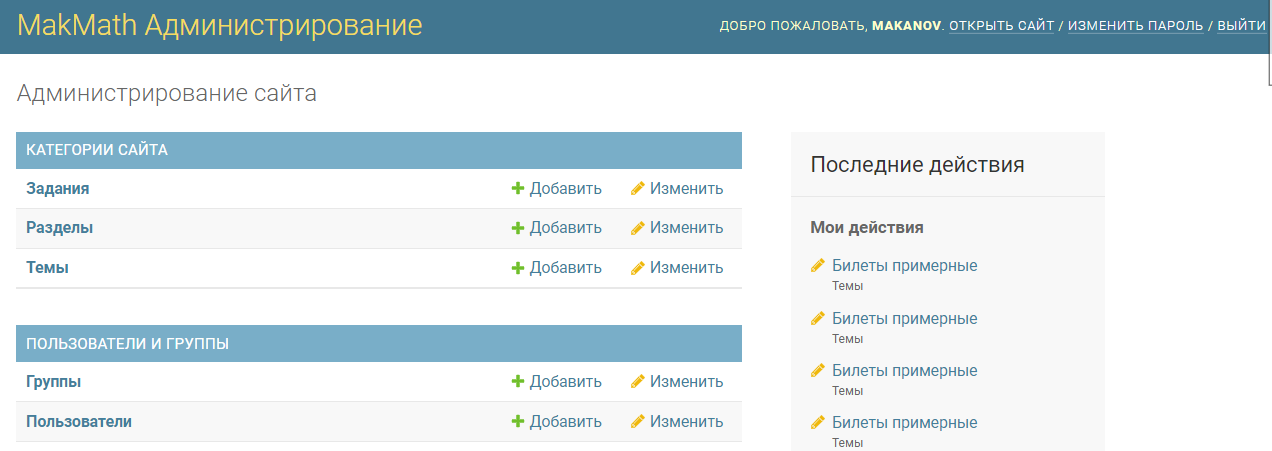
Реализация функциональности администратора полностью выполнена и поставляется в Django. Все разработанные модели и функционал работы с ними подключается автоматически. Программисту нужно лишь создать администратора средствами Django и подключить шаблон admin-панели на сайт для входа администратора. Скриншоты админ-панели, создание одного из объектов темы представлены на рисунках 3.14 и 3.15 соответственно.

Рисунок 3.14 – Админ-панель сайта

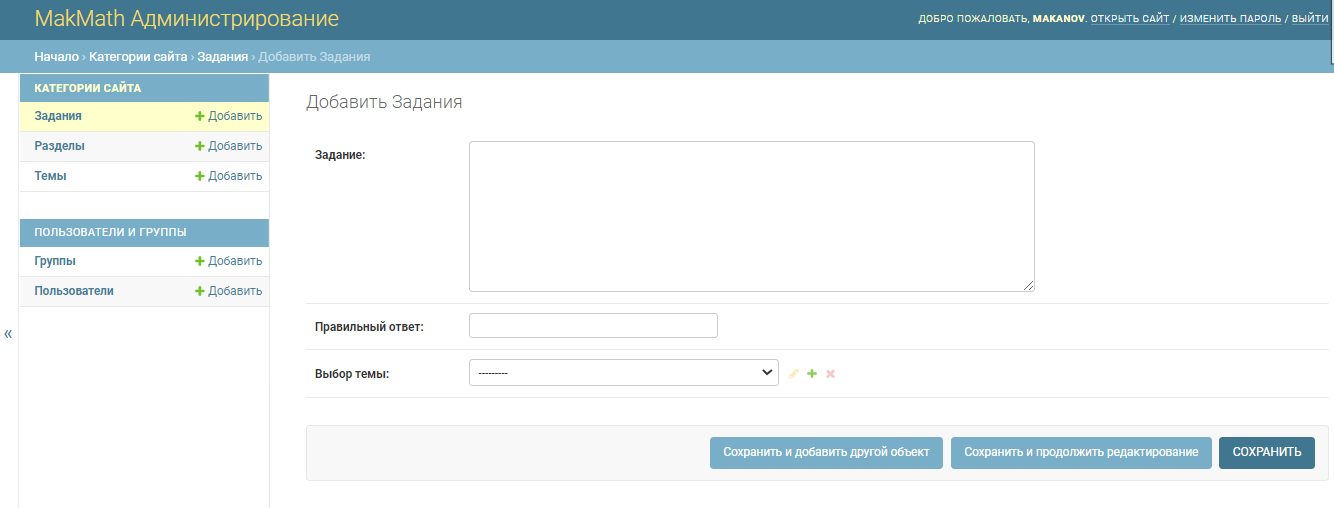


Рисунок 3.15 – Создание объекта на сайте

# 4 ПРОЕКТИРОВАНИЕ МОДУЛЬНЫХ ТЕСТОВ

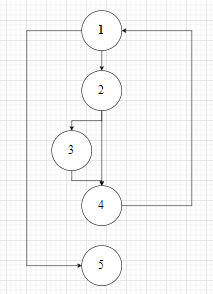
Из всей реализованной функциональности в данном разделе будут описываться аспекты проектирования модульных тестов класса Analizer, методы вычисления статистики по теме или разделу класса StatisticsManager, функционал проверки заданий по ответу класса Task, тестирование ограничений и параметров моделей Section, Topic и Task, тестирование получения доступа к страницам по их url.

# 4.1 Построение ориентированных графом МакКейба

Основной метрикой сложности является цикломатическая сложность графа программы (или цикломатическое число МакКейба), характеризующее трудоемкость тестирования программы.

Для вычисления цикломатического числа МакКейба применяется формула:

*G = R-V+2*

Где *R* – количество ребер графа; *V* – число вершин графа.

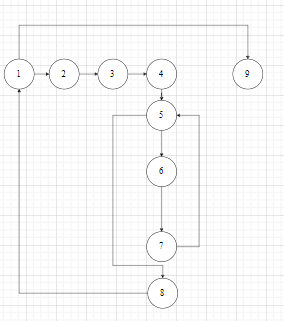
Рисунок 4.1 – Граф для метода find\_index\_of\_value\_in\_array

Рисунок 4.2 – Граф для метода insertion\_sort

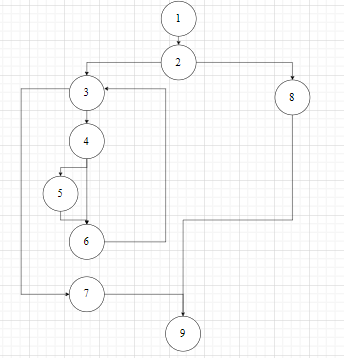


Рисунок 4.3 – Граф для метода is\_correct\_string

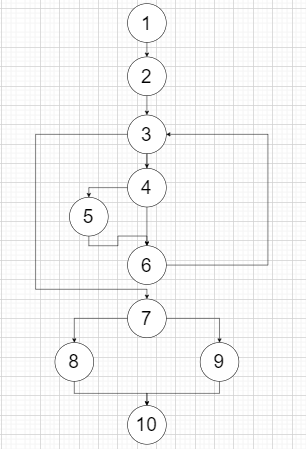


Рисунок 4.4 – Граф для метода get\_statistics\_for\_topic

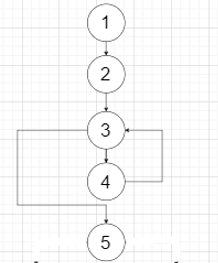
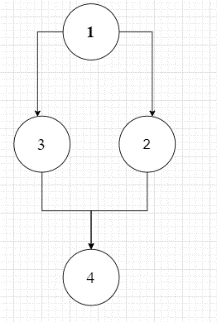
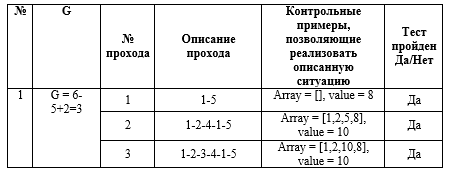


Рисунок 4.5 – Граф для метода get\_statistics\_for\_section

Рисунок 4.6 – Граф для метода check\_task

Тестирование ограничений и параметров моделей Section, Topic и Task, тестирование получения доступа к страницам по их url не нуждаются в построении тестовых проходов и определения цикломатического числа, так как оно заключается в вызове специальных функций Django для определения соответствующих параметров и проверки соответствия их правильным тестовым значениям.

## Формирование проходов и классов эквивалентностей

Таблица 4.1.1 – Проходы для модуля find\_index\_of\_value\_in\_array

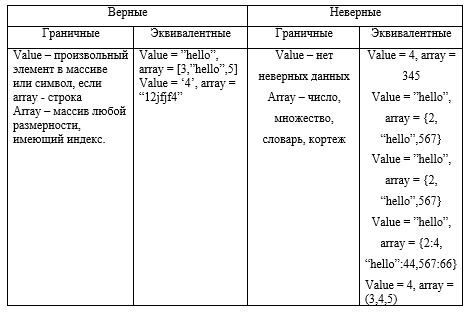
Таблица 4.1.2 – Классы эквивалентности для модуля find\_index\_of\_value\_in\_array()

Таблица 4.2.1 – Проходы для модуля insertion\_sort()

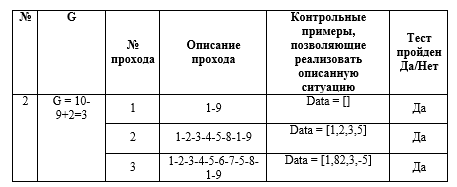


Таблица 4.2.2 – Классы эквивалентности для модуля insertion\_sort()

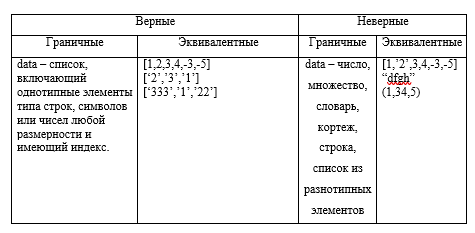


Таблица 4.3.1 – Проходы для модуля is\_correct\_string

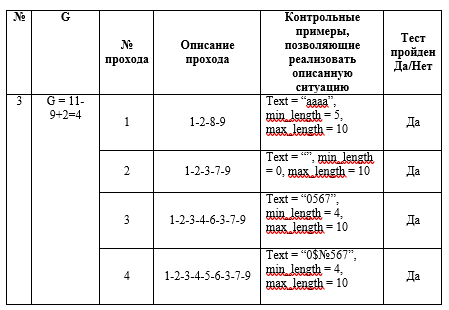


Таблица 4.3.2 – Классы эквиволентности для модуля is\_correct\_string

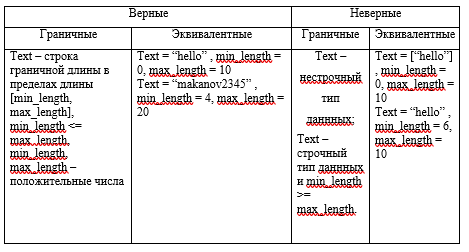


Таблица 4.4.1 – Проходы для модуля get\_statistics\_for\_topic

\

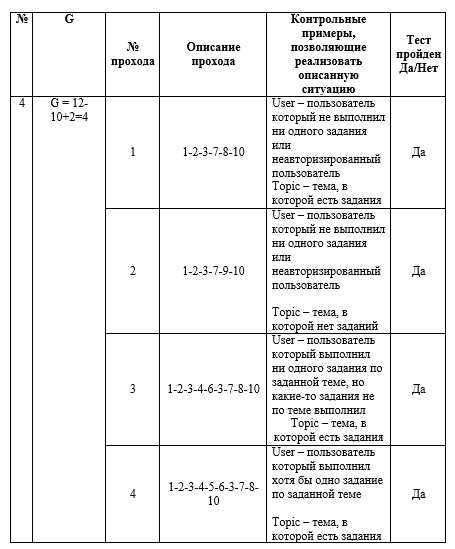


Таблица 4.4.2 – Классы эквивалентности для модуля get\_statistics\_for\_topic

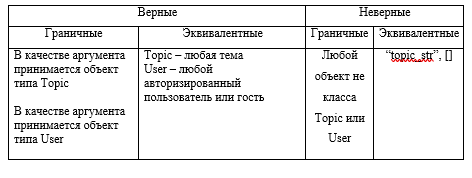


Таблица 4.5.1 – Проходы для модуля get\_statistics\_for\_section()

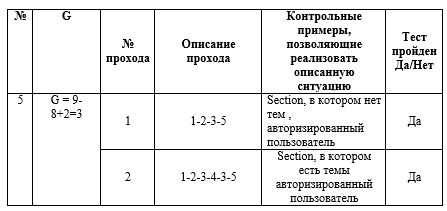


Таблица 4.5.2 – Классы эквивалентностей для модуля get\_statistics\_for\_section()

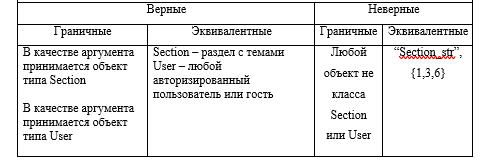


Таблица 4.6.1 – Проходы для модуля check\_task()

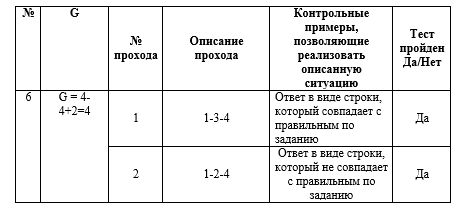
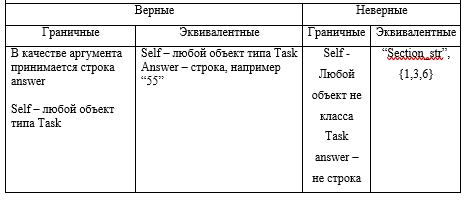


Таблица 4.6.2 – Классы эквивалентностей для модуля check\_task()



# 5 РЕАЛИЗАЦИЯ МОДУЛЬНОГО ТЕСТИРОВАНИЯ

Основная идея юнит-тестирования – это изолированная проверка каждого отдельного элемента путем запуска тестов в искусственной среде.

В данном главе будут реализованы модульные и интеграцианные тесты методов класса Analizer, методов вычисления статистики по теме или разделу класса StatisticsManager, функционал проверки заданий по ответу класса Task, тестирование ограничений и параметров моделей Section, Topic и Task, тестирование получения доступа к страницам по их url.

. Для тестирования использован фреймворк PyTest.[3]

## Реализация модульного тестирования

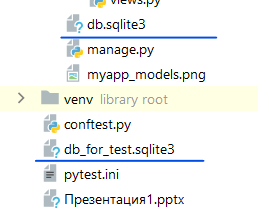
 Реализация тестирования вычисляемых функциональностей приложения таких, как методов класса Analizer, методов вычисления статистики по теме или разделу класса StatisticsManager, функционал проверки заданий по ответу класса Task, проводилась по тестовым проходам из главы 4.2. Для удобства тестирования применялись параметаризированные тесты. Кроме того, для тестирования методов StatisticsManager брались данные из тестовой базы данных. Рисунки 5.1 и 5.2 показывают расположение баз данных приложения и настройки PyTest.

Рисунок 5.1 – Расположение тестовой и используемой баз данных

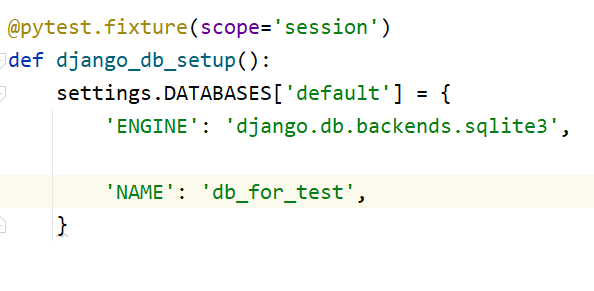


Рисунок 5.2 – Настройки подключения тестовой базы данных в PyTest

Тестирование ограничений и параметров моделей Section, Topic и Task, а также тестирование получения доступа к страницам по их url заключалось в вызове стандартных средств Django для определения необходимых данных и проверке их на соответствие нужным значениям. Пример такого теста изображен на рисунке 5.3.

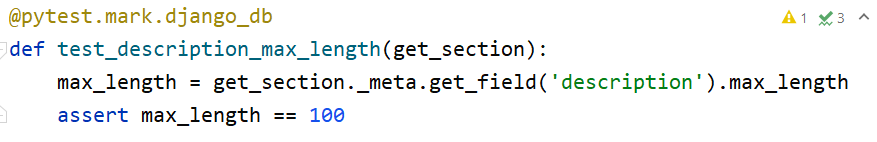


Рисунок 5.3 – Тест на проверку максимальной длины описания раздела

Полная реализация всех тестов находится в приложении B.

## Анализ результатов тестов

В ходе курсовой работы было реализовано 52 модульных и интеграционных тестов для работы с логикой программы, проверки вводимых значений. Разрабатывались как позитивные, так и негативные тесты. Все тесты реализованы с помощью PyTest.

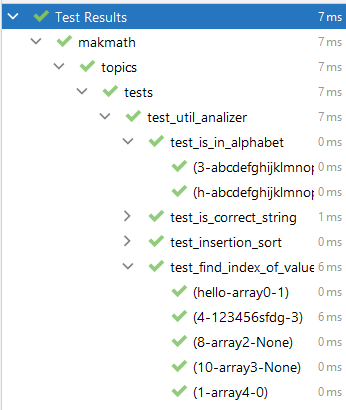
На рисунках 5.4 – 5.9 показаны результаты прохождения тестов. Все тесты завершились успешно.

Рисунок 5.4 – Результат прохождения модульных тестов testAnalizer.py

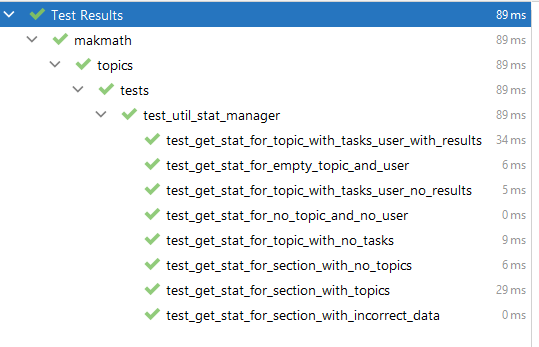


Рисунок 5.5 – Результат прохождения тестов testStatManager

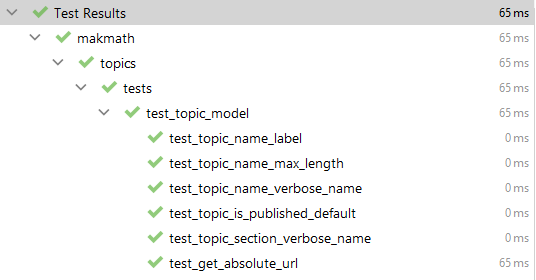


Рисунок 5.6 – Результат прохождения тестов testTopicModel

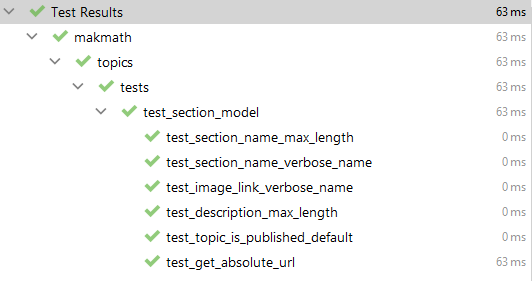


Рисунок 5.7 – Результат прохождения тестов testSectionModel

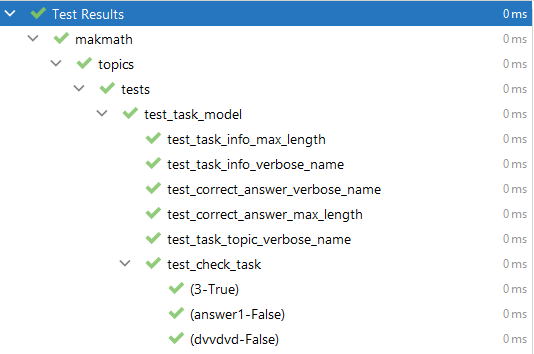


Рисунок 5.8 – Результат прохождения тестов testTaskModel

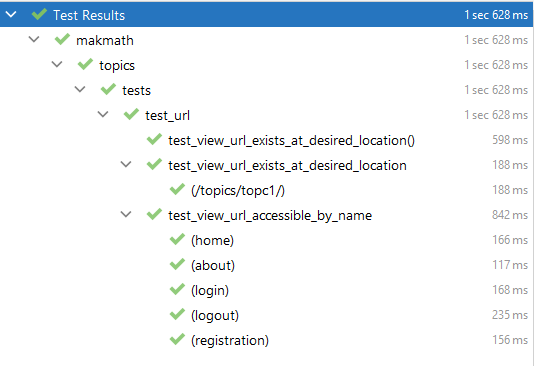


Рисунок 5.9 – Результат прохождения тестов testUrl

## Отладка тестов

Результаты тестирования показывают, что данный программный продукт работает корректно, так как все тесты прошли успешно.

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы было разработано приложение «Сайт по высшей математике». Данное приложение обладает простой навигацией и понятен для пользователей, прост в дальнейшем расширении.

Были изучены основы тестирования программного обеспечения. Были составлены требования для программного продукта, освоены основные этапы тестирования приложений, разработаны позитивные и негативные тесты. Также были разработаны модульные и интеграционные тесты, а ещё тесты для проверки преобразования данных.

Разработаны требования для каждой роли, реализованной в приложении.

Были разработаны и реализованы модули для программного продукта.

В рамках работы были разработаны модульные тесты для разрабатываемого приложения, построены графы МакКейба для методов, содержащих в себе циклы и операторы ветвления. Так же были вычислены цикломатические числа для построенных проходов.

В результате работы были выполнены все требования. Приложение имеет простую навигацию. В приложении минимизирован риск появления ошибки путем уменьшения ввода данных от пользователя.

В процессе тестирования не было найдено ошибок как статическими, так и динамическими методами. Тестирование помогло разработать проект с достаточным уровнем надёжности.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Майерс, Г. Искусство тестирования программ/ Г. Майерс; Пер. с англ. под ред. Позина. – М.: Финансы и статистика, 1982.
2. Python Tutorials [Электронный ресурс] – Доступ: <https://proproprogs.ru/python> - Дата доступа: 11.12.2021.
3. Testing of Django applications [Электронный ресурс] – Доступ: <https://realpython.com/django-pytest-fixtures/> - Дата доступа: 11.12.2021.

# *ПРИЛОЖЕНИЕ А*

Forms.py

from django.contrib.auth.forms import UserCreationForm, UserChangeForm  
from django.contrib.auth.models import User  
from django.core.exceptions import ValidationError  
  
from django import forms  
from django.utils.translation import ugettext\_lazy as \_  
from .utils import Analizator  
  
  
class TaskForm(forms.Form):  
 answer = forms.CharField(max\_length=5)  
 flag = False  
  
 def clean\_answer(self):  
 data = self.cleaned\_data['answer']  
  
 if not data.isdigit():  
 raise ValidationError(\_('Invalid data not int digit given!!!'))  
  
 return data  
  
  
class RegisterUserForm(UserCreationForm):  
 username = forms.CharField(label='Логин', widget=forms.TextInput(), min\_length=3, max\_length=10)  
  
 password1 = forms.CharField(label='Пароль', widget=forms.PasswordInput(), min\_length=8, max\_length=20)  
 password2 = forms.CharField(label='Повтор пароля', widget=forms.PasswordInput(), min\_length=8, max\_length=20)  
  
 class Meta:  
 model = User  
 fields = ('username', 'password1', 'password2')

class UserCreationForm(forms.ModelForm):  
 *"""  
 A form that creates a user, with no privileges, from the given username and  
 password.  
 """* error\_messages = {  
 'password\_mismatch': \_('The two password fields didn’t match.'),  
 }  
 password1 = forms.CharField(  
 label=\_("Password"),  
 strip=False,  
 widget=forms.PasswordInput(attrs={'autocomplete': 'new-password'}),  
 help\_text=password\_validation.password\_validators\_help\_text\_html(),  
 )  
 password2 = forms.CharField(  
 label=\_("Password confirmation"),  
 widget=forms.PasswordInput(attrs={'autocomplete': 'new-password'}),  
 strip=False,  
 help\_text=\_("Enter the same password as before, for verification."),  
 )  
  
 class Meta:  
 model = User  
 fields = ("username",)  
 field\_classes = {'username': UsernameField}  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 if self.\_meta.model.USERNAME\_FIELD in self.fields:  
 self.fields[self.\_meta.model.USERNAME\_FIELD].widget.attrs['autofocus'] = True  
  
 def clean\_password2(self):  
 password1 = self.cleaned\_data.get("password1")  
 password2 = self.cleaned\_data.get("password2")  
 if password1 and password2 and password1 != password2:  
 raise ValidationError(  
 self.error\_messages['password\_mismatch'],  
 code='password\_mismatch',  
 )  
 return password2  
  
 def \_post\_clean(self):  
 super().\_post\_clean()  
 *# Validate the password after self.instance is updated with form data  
 # by super().* password = self.cleaned\_data.get('password2')  
 if password:  
 try:  
 password\_validation.validate\_password(password, self.instance)  
 except ValidationError as error:  
 self.add\_error('password2', error)  
  
 def save(self, commit=True):  
 user = super().save(commit=False)  
 user.set\_password(self.cleaned\_data["password1"])  
 if commit:  
 user.save()  
 return user

utils.py

from .models import \*  
  
  
class StatisticsManager:  
 @staticmethod  
 def get\_statistics\_for\_topic(topic, user):  
 completed\_tasks = TaskComplition.objects.filter(user=user)  
  
 topic\_completed\_tasks = []  
  
 for item in completed\_tasks:  
  
 if item.task.topic == topic:  
 topic\_completed\_tasks.append(item)  
  
 amount\_of\_topic\_tasks = len(Task.objects.filter(topic=topic))  
 return (len(topic\_completed\_tasks) \* 100 // amount\_of\_topic\_tasks  
 if amount\_of\_topic\_tasks  
 else 0)  
  
 @staticmethod  
 def get\_statistics\_for\_section(section, user):  
  
 topics = Topic.objects.filter(section=section)  
  
 sum = 0  
  
 for item in topics:  
 sum += StatisticsManager.get\_statistics\_for\_topic(item, user)  
  
 return sum // len(topics) if topics else 0  
  
 @staticmethod  
 def get\_statistics\_in\_html(user):  
 stat\_info = "<ul>"  
 for item in Section.objects.filter(is\_published=True):  
 stat\_info += (f"<li>{item.name} - "  
 f"{StatisticsManager.get\_statistics\_for\_section(item, user)}%<ul>")  
  
 for topic in Topic.objects.filter(section=item):  
 stat\_info += (f"<li>{topic.topic\_name} - "  
 f"{StatisticsManager.get\_statistics\_for\_topic(topic, user)}%</li>")  
  
 stat\_info += "</ul></li>"  
  
 stat\_info += "</ul>"  
  
 return stat\_info  
  
  
ALPHABET = "abcdefghijklmnopwrstuwxyz\_0123456789"  
  
  
class Analizator():  
  
 @staticmethod  
 def is\_in\_alphabet(character, alphabet):  
 return character.lower() in alphabet  
  
 @staticmethod  
 def is\_correct\_string(text, min\_length, max\_length):  
 correction = True  
  
 if min\_length <= len(text) <= max\_length:  
 for character in text:  
 if not Analizator.is\_in\_alphabet(character, ALPHABET):  
 correction = False  
 break  
 else:  
 correction = False  
  
 return correction

models.py

from django.db import models  
from django.urls import reverse  
from ckeditor.fields import RichTextField  
  
from django.contrib.auth.models import User

class Topic(models.Model):  
 topic\_name = models.CharField(max\_length=255, verbose\_name="Название темы")  
 content = RichTextField(null=True, verbose\_name="Текст лекции")  
 time\_create = models.DateTimeField(auto\_now\_add=True, verbose\_name="Время создания")  
 time\_update = models.DateTimeField(auto\_now=True, verbose\_name="Время обновления")  
 is\_published = models.BooleanField(default=True, verbose\_name="Опубликовать")  
 section = models.ForeignKey('Section', on\_delete=models.PROTECT, null=True, verbose\_name="Выбор раздела")  
 slug = models.SlugField(max\_length=255, unique=True, db\_index=True,  
 verbose\_name="URL", null=True)  
  
 def get\_absolute\_url(self):  
 return reverse('topic', kwargs={'topic\_slug': self.slug})  
  
 def \_\_str\_\_(self):  
 return self.topic\_name  
  
 class Meta:  
 verbose\_name = 'Темы'  
 verbose\_name\_plural = 'Темы'  
 ordering = ['section', 'time\_create']  
  
  
class Section(models.Model):  
 name = models.CharField(max\_length=100, db\_index=True, verbose\_name="Название раздела")  
 image\_link = models.TextField(null=True, verbose\_name="URL картинки")  
 description = models.TextField(max\_length=100, null=True, verbose\_name="Описание")  
 slug = models.SlugField(max\_length=255, unique=True, db\_index=True, verbose\_name="URL", null=True)  
 is\_published = models.BooleanField(default=True, verbose\_name="Опубликовать")  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 def get\_absolute\_url(self):  
 return reverse('section', kwargs={'section\_slug': self.slug})  
  
 class Meta:  
 verbose\_name = 'Разделы'  
 verbose\_name\_plural = 'Разделы'  
  
  
class Task(models.Model):  
 task\_info = models.TextField(max\_length=255,null=True, verbose\_name="Задание")  
 correct\_answer = models.CharField(max\_length=5, null=True, verbose\_name="Правильный ответ")  
 topic = models.ForeignKey('Topic', on\_delete=models.PROTECT, null=True, verbose\_name="Выбор темы")  
  
 def \_\_str\_\_(self):  
 return self.task\_info  
  
 class Meta:  
 verbose\_name = 'Задания'  
 verbose\_name\_plural = 'Задания'  
 ordering = ['topic']  
  
 def check\_task(self, answer):  
 return answer == self.correct\_answer  
  
  
class TaskComplition(models.Model):  
 task = models.ForeignKey('Task', on\_delete=models.SET\_NULL, null=True)  
 user = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True)

views.py

from django.contrib.auth import authenticate, login  
from django.contrib.auth.decorators import login\_required  
from django.contrib import auth  
from django.contrib.auth.forms import AuthenticationForm  
from django.contrib.auth.views import LoginView  
from django.core.checks import messages  
from django.http import HttpResponseNotFound, HttpResponseRedirect, JsonResponse  
from django.shortcuts import render, get\_object\_or\_404  
from django.urls import reverse\_lazy  
  
from django.views.generic import ListView, DetailView, CreateView, TemplateView  
  
from .forms import \*  
from .models import \*  
from .utils import StatisticsManager  
  
menu = [  
 {'title': "О сайте", 'url\_name': 'about'},  
]  
  
  
def show\_topic(request, topic\_slug):  
 topic = get\_object\_or\_404(Topic, slug=topic\_slug)  
 tasks = Task.objects.filter(topic=topic)  
 context = {  
 'topic': topic,  
 'menu': menu,  
 'title': topic.topic\_name,  
 'lection': topic.content,  
 'tasks': tasks,  
 'forms': "",  
 'section\_link': topic.section.get\_absolute\_url(),  
 }  
 forms = []  
 if request.user.is\_authenticated:  
  
 i = 0  
 if request.method == 'POST':  
 keys = list(request.POST.keys())  
 for t in tasks:  
 form = TaskForm(request.POST)  
 form.fields['answer'].label = t.task\_info  
 form.prefix = i + 1  
 form.flag = True if TaskComplition.objects.filter(  
 task=t, user=request.user) else False  
 i += 1  
 if request.method == 'POST':  
 if str(i) in keys[len(keys) - 1]:  
 form.flag = t.check\_task(  
 request.POST[f'{i}-answer'])  
 if form.flag and \  
 not TaskComplition.objects.filter(  
 task=t, user=request.user):  
 task\_completed = TaskComplition(  
 task=t, user=request.user)  
 task\_completed.save()  
 forms.append(form)  
  
 context['forms'] = forms  
 return render(request, 'topics/topic.html', context=context)  
  
  
class MakMathHome(ListView):  
 model = Section  
 template\_name = "topics/index.html"  
 context\_object\_name = "sections"  
  
 def get\_context\_data(self, \*, object\_list=None, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['menu'] = menu  
 context['title'] = 'MakMath главная страница'  
 return context  
  
 def get\_queryset(self):  
 return Section.objects.filter(is\_published=True)  
  
  
class SectionShower(ListView):  
 model = Topic  
 template\_name = "topics/section.html"  
 context\_object\_name = "topics"  
  
 def get\_context\_data(self, \*, object\_list=None, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['menu'] = menu  
 if context['topics']:  
 context['title'] = str(context['topics'][0].section)  
  
 return context  
  
 def get\_queryset(self):  
 return Topic.objects.filter(section\_\_slug=self.kwargs['section\_slug'])  
  
  
def about(request):  
 context = {'menu': menu, 'title': "O сайте"}  
 return render(request, 'topics/about.html', context=context)  
  
  
def page\_not\_found(request, exception):  
 return HttpResponseNotFound('<h1>Страница не найдена</h1>')  
  
  
class LoginUser(LoginView):  
 form\_class = AuthenticationForm  
 template\_name = 'topics/authorization.html'  
  
 def get\_context\_data(self, \*, object\_list=None, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 return dict(list(context.items()))  
  
 def get\_success\_url(self):  
 return reverse\_lazy('home')  
  
  
class RegisterUser(CreateView):  
 form\_class = RegisterUserForm  
 template\_name = 'topics/registration.html'  
 success\_url = reverse\_lazy('login')  
  
 def get\_context\_data(self, \*, object\_list=None, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
  
 return dict(list(context.items()))  
  
  
def show\_user\_info(request):  
  
 context = {'menu': menu,  
  
 'title': f"Профиль {request.user.username}",  
 'stat\_info':  
 StatisticsManager.get\_statistics\_in\_html(request.user)}  
 return render(request, 'topics/user\_cabinet.html', context=context)  
  
  
def logout(request):  
 auth.logout(request)  
 *# Перенаправление на страницу.* return HttpResponseRedirect(reverse\_lazy('login'))

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', 'makmath.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()

# 

# *ПРИЛОЖЕНИЕ B*

testUtilAnalizer.py

import pytest  
from topics.utils import \*  
  
  
*# Analizator testing*@pytest.mark.parametrize("character, alphabet, expected",  
 [('3', ALPHABET, True),  
 ('h', ALPHABET, True), ])  
def test\_is\_in\_alphabet(character, alphabet, expected):  
 assert Analizator.is\_in\_alphabet(character, alphabet) == expected  
  
  
@pytest.mark.parametrize("text, min\_length, max\_length, expected",  
 [  
 ("hello", 6, 10, False),  
 ('aaaa', 5, 10, False),  
 ('', 0, 10, True),  
 ("0567", 4, 10, True),  
 ("0$№5670$№567", 4, 10, False)])  
def test\_is\_correct\_string(text, min\_length, max\_length, expected):  
 assert Analizator.is\_correct\_string(text, min\_length,  
 max\_length) == expected  
  
  
@pytest.mark.parametrize("data, expected",  
 [  
 (['2', '3', '1'], ['1', '2', '3']),  
 (['333', '22', '1'], ['1', '22', '333']),  
 ([], []),  
 ([1, 2, 3, 5], [1, 2, 3, 5]),  
 ([1, 82, 3, -5], [-5, 1, 3, 82]),  
 ])  
def test\_insertion\_sort(data, expected):  
 assert Analizator.insertion\_sort(data) == expected  
  
  
@pytest.mark.parametrize("value, array, expected",  
 [  
 ('hello', [2, 'hello', 23], 1),  
 ('4', '123456sfdg', 3),  
 (8, [], None),  
 (10, [1, 2, 3, 5], None),  
 (1, [1, 82, 3, -5], 0),  
 ])  
def test\_find\_index\_of\_value\_in\_array(value, array, expected):  
 assert Analizator.find\_index\_of\_value\_in\_array(value, array) == expected

testStatManager.py

import pytest  
from topics.utils import \*  
  
  
*# StatisticsManager testing  
  
# get\_stat\_for\_topic testing*@pytest.mark.django\_db  
def test\_get\_stat\_for\_topic\_with\_tasks\_user\_with\_results():  
 topic, user = Topic.objects.filter(id=3)[0], User.objects.filter(id=2)[0]  
 assert list(Task.objects.filter(topic=topic)) != []  
 expected = 100  
 assert StatisticsManager.get\_statistics\_for\_topic(topic, user) == expected  
  
  
@pytest.mark.django\_db  
def test\_get\_stat\_for\_empty\_topic\_and\_user():  
 topic, user = Topic(), User()  
 assert list(Task.objects.filter(topic=topic)) == []  
 expected = 0  
 assert StatisticsManager.get\_statistics\_for\_topic(topic, user) == expected  
  
  
@pytest.mark.django\_db  
def test\_get\_stat\_for\_topic\_with\_tasks\_user\_no\_results():  
 topic, user = Topic.objects.filter(id=3)[0], User.objects.filter(id=12)[0]  
 assert list(Task.objects.filter(topic=topic)) != []  
 expected = 0  
 assert StatisticsManager.get\_statistics\_for\_topic(topic, user) == expected  
  
  
@pytest.mark.django\_db  
def test\_get\_stat\_for\_no\_topic\_and\_no\_user():  
 topic, user = {}, "nkjn"  
 expected = 0  
 assert StatisticsManager.get\_statistics\_for\_topic(topic, user) == expected  
  
  
@pytest.mark.django\_db  
def test\_get\_stat\_for\_topic\_with\_no\_tasks():  
 topic, user = Topic.objects.filter(id=2)[0], User.objects.filter(id=12)[0]  
 assert list(Task.objects.filter(topic=topic)) == []  
 expected = 0  
 assert StatisticsManager.get\_statistics\_for\_topic(topic, user) == expected  
  
  
*# get\_stat\_for\_section testing*@pytest.mark.django\_db  
def test\_get\_stat\_for\_section\_with\_no\_topics():  
 section, user = Section(), User.objects.filter(id=12)[0]  
 assert list(Topic.objects.filter(section=section)) == []  
 expected = 0  
 assert StatisticsManager.get\_statistics\_for\_section(section, user) \  
 == expected  
  
  
@pytest.mark.django\_db  
def test\_get\_stat\_for\_section\_with\_topics():  
 section, user = Section.objects.filter(id=1)[0], User.objects.filter(id=2)[0]  
 assert list(Topic.objects.filter(section=section)) != []  
 expected = 50  
 assert StatisticsManager.get\_statistics\_for\_section(section, user) \  
 == expected  
  
  
@pytest.mark.django\_db  
def test\_get\_stat\_for\_section\_with\_incorrect\_data():  
 section, user = [0], "some\_text"  
 expected = 0  
 assert StatisticsManager.get\_statistics\_for\_section(section, user) \  
 == expected

testTopicModel.py

@pytest.fixture()  
def get\_topic():  
 return Topic.objects.get(id=2)  
  
  
@pytest.mark.django\_db  
def test\_topic\_name\_label(get\_topic):  
 assert get\_topic.topic\_name == 'Тема 1. Введение в математический анализ.'  
  
  
@pytest.mark.django\_db  
def test\_topic\_name\_max\_length(get\_topic):  
 max\_length = get\_topic.\_meta.get\_field('topic\_name').max\_length  
 assert max\_length == 255  
  
  
@pytest.mark.django\_db  
def test\_topic\_name\_verbose\_name(get\_topic):  
 verbose\_name = get\_topic.\_meta.get\_field('topic\_name').verbose\_name  
 assert verbose\_name == "Название темы"  
  
  
@pytest.mark.django\_db  
def test\_topic\_name\_verbose\_name(get\_topic):  
 verbose\_name = get\_topic.\_meta.get\_field('content').verbose\_name  
 assert verbose\_name == "Текст лекции"  
  
  
@pytest.mark.django\_db  
def test\_topic\_is\_published\_default(get\_topic):  
 default = get\_topic.\_meta.get\_field('is\_published').default  
 assert default == True  
  
  
@pytest.mark.django\_db  
def test\_topic\_section\_verbose\_name(get\_topic):  
 verbose\_name = get\_topic.\_meta.get\_field('section').verbose\_name  
 assert verbose\_name == "Выбор раздела"  
  
  
@pytest.mark.django\_db  
def test\_get\_absolute\_url(get\_topic):  
 assert get\_topic.get\_absolute\_url() == '/topics/topc1/'

testTaskModel.py

@pytest.fixture()  
def get\_task():  
 return Task.objects.filter(id=2)[0]  
  
  
@pytest.mark.django\_db  
def test\_task\_info\_max\_length(get\_task):  
 max\_length = get\_task.\_meta.get\_field('task\_info').max\_length  
 assert max\_length == 255  
  
  
@pytest.mark.django\_db  
def test\_task\_info\_verbose\_name(get\_task):  
 verbose\_name = get\_task.\_meta.get\_field('task\_info').verbose\_name  
 assert verbose\_name == "Задание"  
  
  
@pytest.mark.django\_db  
def test\_correct\_answer\_verbose\_name(get\_task):  
 verbose\_name = get\_task.\_meta.get\_field('correct\_answer').verbose\_name  
 assert verbose\_name == "Правильный ответ"  
  
  
@pytest.mark.django\_db  
def test\_correct\_answer\_max\_length(get\_task):  
 max\_length = get\_task.\_meta.get\_field('correct\_answer').max\_length  
 assert max\_length == 5  
  
  
@pytest.mark.django\_db  
def test\_task\_topic\_verbose\_name(get\_task):  
 verbose\_name = get\_task.\_meta.get\_field('topic').verbose\_name  
 assert verbose\_name == "Выбор темы"  
  
  
*# check\_task testing*@pytest.fixture()  
def get\_simple\_task():  
 task = Task()  
 task.correct\_answer = "3"  
 return task  
  
  
@pytest.mark.parametrize("answer, expected",  
 [('3', True),  
 ([0], False),  
 ("dvvdvd", False)])  
def test\_check\_task(answer, expected, get\_simple\_task):  
 assert get\_simple\_task.check\_task(answer) == expected

testSectionModel.py

@pytest.fixture()  
def get\_section():  
 return Section.objects.filter(id=1)[0]  
  
  
@pytest.mark.django\_db  
def test\_section\_name\_max\_length(get\_section):  
 max\_length = get\_section.\_meta.get\_field('name').max\_length  
 assert max\_length == 100  
  
  
@pytest.mark.django\_db  
def test\_section\_name\_verbose\_name(get\_section):  
 verbose\_name = get\_section.\_meta.get\_field('name').verbose\_name  
 assert verbose\_name == "Название раздела"  
  
  
@pytest.mark.django\_db  
def test\_image\_link\_verbose\_name(get\_section):  
 verbose\_name = get\_section.\_meta.get\_field('image\_link').verbose\_name  
 assert verbose\_name == "URL картинки"  
  
  
@pytest.mark.django\_db  
def test\_description\_max\_length(get\_section):  
 max\_length = get\_section.\_meta.get\_field('description').max\_length  
 assert max\_length == 100  
  
  
@pytest.mark.django\_db  
def test\_topic\_is\_published\_default(get\_section):  
 default = get\_section.\_meta.get\_field('is\_published').default  
 assert default is True  
  
  
@pytest.mark.django\_db  
def test\_get\_absolute\_url(get\_section):  
 assert get\_section.get\_absolute\_url() == '/section/diff\_ovf/'

testUrl.py

BASE\_LOCATION = "http://127.0.0.1:8000"  
  
  
@pytest.mark.parametrize("location",  
 ['',  
 '/topics/topc1/',  
 ])  
def test\_view\_url\_exists\_at\_desired\_location(location):  
 resp = requests.get(BASE\_LOCATION + location)  
 assert resp.status\_code == 200  
  
  
@pytest.mark.parametrize("name",  
 ['home',  
 'about',  
 'login',  
 'logout',  
 'registration'  
 ])  
def test\_view\_url\_accessible\_by\_name(name):  
 resp = requests.get(BASE\_LOCATION + reverse(name))  
 assert resp.status\_code == 200