

Колледж космического машиностроения и технологий

**КУРСОВОЙ ПРОЕКТ**

**По МДК.01.02 «Прикладное программирование»**

**Тема: «Приложение "Управление заказами"»**

Выполнил студент

Батраков Д.С.

Группа П1-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Принял преподаватель

Гусятинер Л.Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Подпись)

**Королев 2021 г.**

Оглавление

[Введение 3](#_Toc69837022)

[1 Теоретическая часть 4](#_Toc69837023)

[1.1 Описание предметной области 4](#_Toc69837024)

[1.2 Описание существующих разработок 8](#_Toc69837025)

[2 Проектная часть 13](#_Toc69837026)

[2.1 Диаграмма прецедентов 13](#_Toc69837027)

[2.2 Выбор инструментов 13](#_Toc69837028)

[2.2.1 Визуальная часть сайта 13](#_Toc69837029)

[2.2.2 Среда разработки программного обеспечения 15](#_Toc69837030)

[2.3 Проектирование сценария 16](#_Toc69837031)

[2.4 Диаграммы классов 16](#_Toc69837032)

[2.5 Описание главного модуля. 17](#_Toc69837033)

[2.6 Описание спецификаций к модулям 18](#_Toc69837034)

[2.7 Описание модулей 18](#_Toc69837035)

[2.8 Описание тестовых наборов модулей 21](#_Toc69837036)

[2.9 Описание применения средств отладки 22](#_Toc69837037)

[3 Эксплуатационная часть 24](#_Toc69837038)

[3.1 Руководство оператора 24](#_Toc69837039)

[3.1.1 Назначение программы 24](#_Toc69837040)

[3.1.2 Условия выполнения программы 24](#_Toc69837041)

[3.1.3 Выполнение программы 24](#_Toc69837042)

[3.1.4 Сообщение оператору 27](#_Toc69837043)

[3.2 To-Do лист 28](#_Toc69837044)

[Заключение 29](#_Toc69837045)

[Список литературы и интернет ресурсов 30](#_Toc69837046)

[Приложения 31](#_Toc69837047)

# Введение

Данный курсовой проект заключается в разработке приложения, для быстрого и удобного взаимодействия с базами данных. Разработка приложения будет происходить на Python.

В первой части будут рассмотрены предметная область и существующие продукты по данной теме.

Во второй части будут описаны инструменты разработки программы и модули программы.

В третьей части описаны взаимодействие пользователя с программой и сообщения оператору.

В заключительной части будут приведены общие выводы по проекту.

# Теоретическая часть

## Описание предметной области

**Управление заказами** – объемное понятие, которое подразумевает контроль на каждом этапе цикла: от момента оформления заявки до доставки товара покупателю.

**Быстрая и корректная обработка заказов** – одно из важнейших конкурентных преимуществ в электронной коммерции. Оформляя покупку в интернет-магазине, потребитель ожидает получить товар в кратчайшие сроки. И он не придет на сайт повторно, если не будет мгновенной обратной связи, возникнут какие-либо накладки или заказ попросту потеряется среди десятков других. Чтобы исключить подобные ситуации, **нужно обеспечить грамотное и эффективное управление заказами покупателей**.

**Главная задача управления – качественная и четкая отработка всех этапов.** Этот процесс является одним из элементов общей концепции обслуживания покупателей. При условии правильной организации он помогает увеличить лояльность аудитории и поддерживать положительную репутацию компании.

**Эффективность управления зависит от следующих факторов:**

* **систематизация**. Информацию о товарах и текущих статусах сделок желательно структурировать и хранить в единой системе;
* **прозрачность**. Важно своевременно фиксировать и передавать актуальные данные, иметь наглядное представление цепочки поставок, оперативно выявлять и устранять проблемы;
* **аналитика**. Необходимо прогнозировать и анализировать важные для компании метрики, настраивать процессы, исходя из намеченных показателей. Прогнозирование объемов продаж позволяет создать достаточное количество запасов на складе, что способствует быстрому удовлетворению запросов клиента;
* **гибкость**. Следует разделять заказы на группы и сегменты, ставить отдельные задачи и передавать их в соответствующие отделы и системы.

## Описание существующих разработок

На данный момент существует множество уже разработанных приложений по этой тематике. Практически все из них предоставляют похожий функционал:

* размещение заказа в базе;
* проверка данных;
* сверка с актуальными запасами на складе;
* подтверждение покупки;
* создание счетов;
* прием оплаты;
* отгрузка и упаковка;
* передача груза перевозчику;
* информирование покупателя о текущем статусе заявки;
* доставка;
* обновление информации о доступных остатках в товарных карточках.

Примеры сайтов:

* RetailCRM[1]

помогает продавать товары в интернете, работать с клиентской базой, автоматизировать обработку заказов, анализировать статистику, вести email-маркетинг.

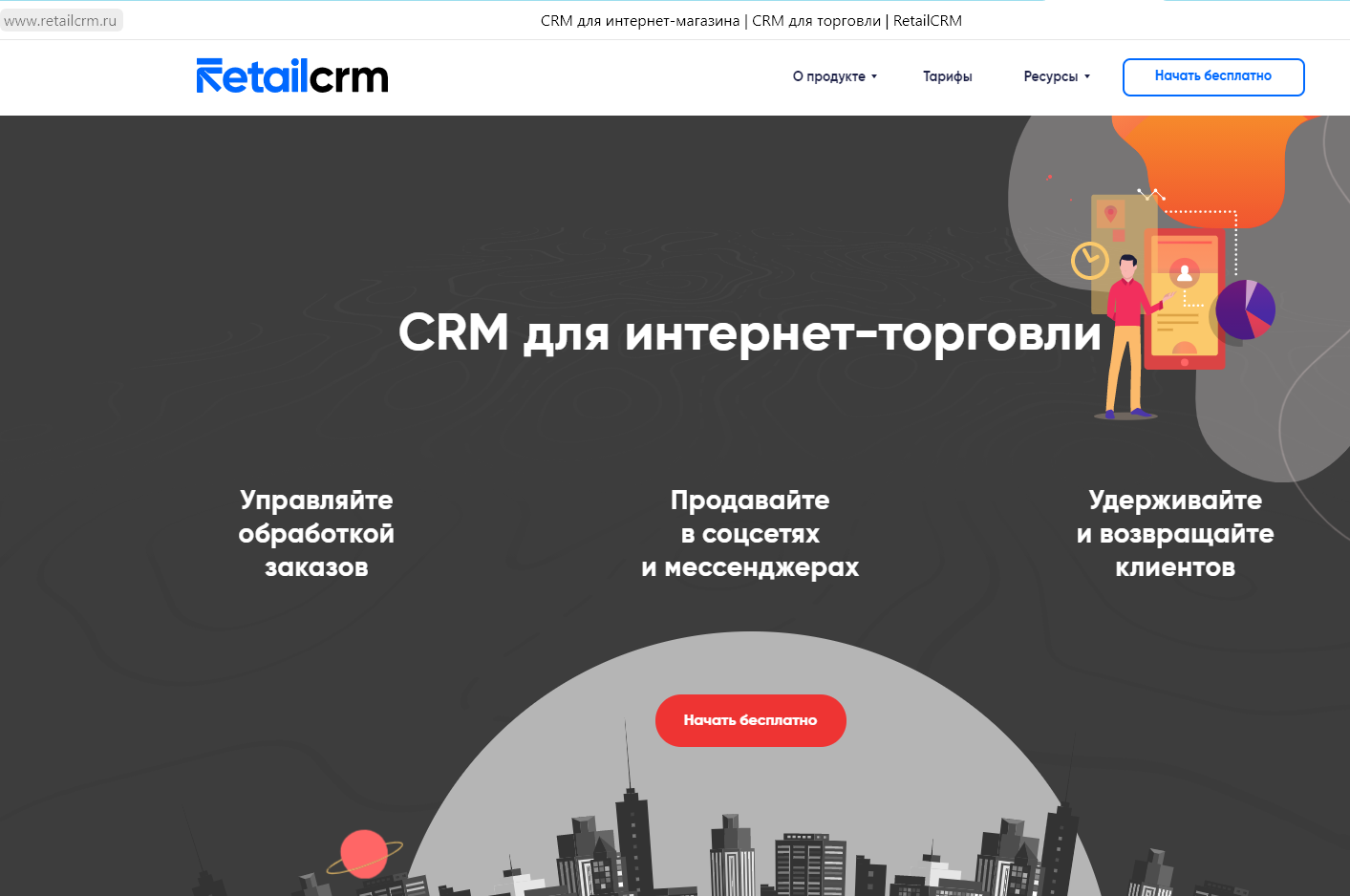


Рисунок 1. Окно сайта RetailCRM

* Ekam [2]

Возможность загружать товары из Excel или создавать в самой системе; Создавать оприходование или приемку товара; Управление заказами из вашего интернет-магазина; Отслеживайте остатки: фактические и с учетом резерва; Формирование отчётов по дням, сотрудникам, проданным позициям, контроль работы персонала.

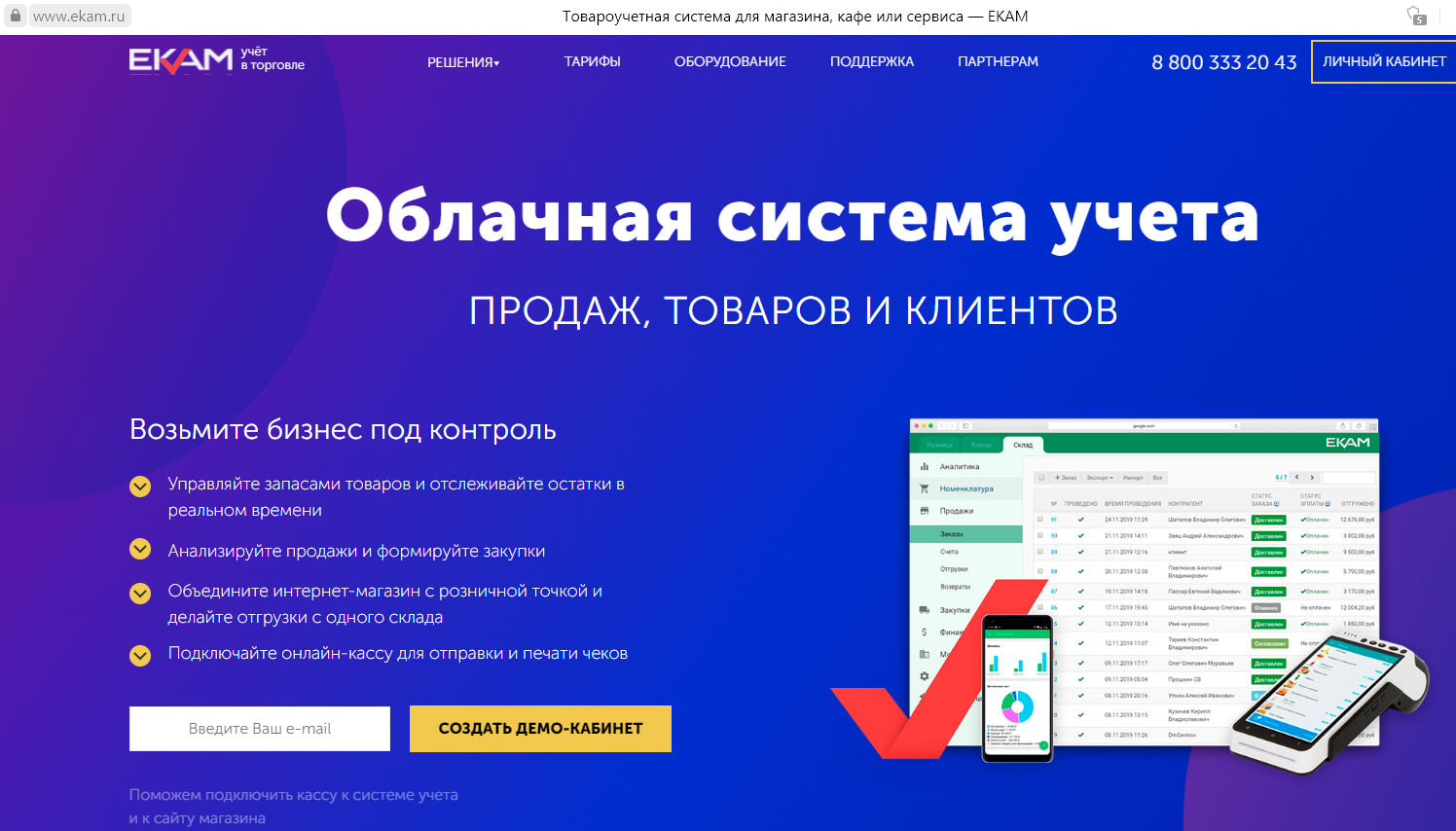


Рисунок 2. Окно сайта Ekam

* generixgroup [3]

визуальный доступ к запасам и остаткам для специалистов, помощь в учете и аналитике;

эффективное планирование и контроль логистических операций;

поддержку продажи, позволяющую в точности выполнить обязательства перед клиентом;

оптимизацию бизнес-процессов и документооборота, связанного с закрытием заявок;

снижение логистических издержек.

# 

Рисунок 3. Окно сайта generixgroup

# 2 Проектная часть

## Диаграмма прецедентов

В данном разделе содержится диаграмма прецедентов для приложения, на которой показаны возможные функциональные отношения.

Администратор

Рисунок 4. Диаграмма прецедентов приложения.

## Выбор инструментов

В разработке приложения “Управление заказами”, используется Python, в качестве СУБД использовался SQLite и в качестве IDE используется PyCharm и компилятор GCC, который встроен в PyCharm.

### Язык программирования

**Python** - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным - всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скрипто. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектноориентированное программирование, метапрограммирование и функциональное программирование. Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается через декораторы, более полноценная поддержка обеспечивается дополнительными фреймворками. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений.

Стандартная библиотека включает большой набор полезных переносимых функций, начиная от функционала для работы с текстом и заканчивая средствами для написания сетевых приложений. Дополнительные возможности, такие как математическое моделирование, работа с оборудованием, написание веб-приложений или разработка игр, могут реализовываться посредством обширного количества сторонних библиотек, а также интеграцией библиотек, написанных на Си или C++, при этом и сам интерпретатор Python может интегрироваться в проекты, написанные на этих языках. Существует и специализированный репозиторий программного обеспечения, написанного на Python, — PyPI. Данный репозиторий предоставляет средства для простой установки пакетов в операционную систему и стал стандартом де-факто для Python. По состоянию на 2019 год в нём содержалось более 175 тысяч пакетов.

### База данных

В разработке приложений под Ubuntu в основном в качестве базы данных используется **SQLite**. Причины использования:

1. **Минимальные затраты ресурсов.** Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера: база данных SQLite представляет собой обычный файл. Когда БД не используется, она не расходует процессорное время. Это особенно важно на мобильных устройствах, чтобы избежать разрядки аккумулятора.
2. **Оптимизация для одного пользователя.** С базой данных взаимодействует только наше приложение. Поэтому можно обойтись без идентификации с именем пользователя и паролем.
3. **Надежность и быстрота.** Базы данных SQLite поддерживают транзакции баз данных. Кроме того, операции чтения и записи данных реализуются на оптимизированном коде С.

## Проектирование сценария

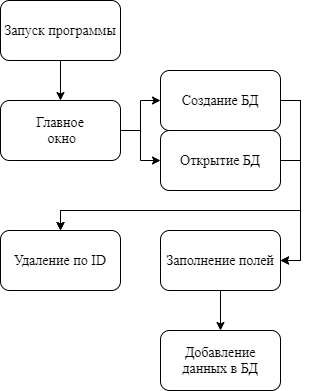
В данном разделе приведен сценарий использования приложения пользователем (Рисунок 5).

Рисунок 5. Сценарий использования сайта.

При запуске приложения открывается главный экран для ввода данных. Для работы с программой необходимо открыть или создать новый файл типа «Data Base File», сделать это можно на панели функций. После ввода данных и нажатия кнопки «Добавить» все введенные данные сохраняются в ранее открытую БД.

## Диаграммы классов

В данном разделе располагается диаграмма классов сайта. (Рисунок 6).

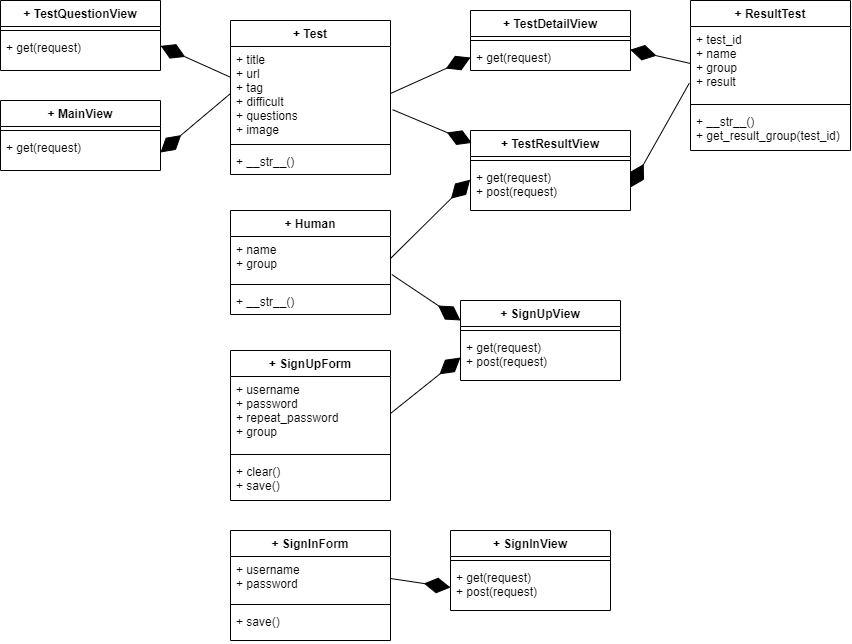


Рисунок 7. Диаграмма классов сайта

## Описание главного модуля.

Главный модуль состоит из класса Database. Он выполняется при запуске приложения. Данный модуль подключается к базе данных и создаёт основную таблицу и функции.

**Листинг 1**. Главный модуль

from tkinter import \*

from tkinter import messagebox

from tkinter import filedialog

import sqlite3

TEXT\_BUTTON\_DEL = "Удалить"

TEXT\_CREATE\_FILE = "Создать базу данных"

TEXT\_OPEN\_FILE = "Открыть базу данных"

nameDB = "database.db"

BUTTON\_COLOR = "#E6E6FA"

WINDOW\_COLOR = "#FF8C00"

ACT\_BUTTON\_COLOR = "#B0C4DE"

LABEL\_BG = "#FF8C00"

class Database:

def \_\_init\_\_(self, nameDB):

# подключение к базе данных

self.db = sqlite3.connect(nameDB)

self.cursor = self.db.cursor()

def create\_table(self, table):

# создание таблицы со столбцами в базе данных

self.cursor.execute(f"CREATE TABLE IF NOT EXISTS '{table}' (\

order\_id INTEGER, \

order\_name TEXT, \

customer\_name TEXT, \

number INTEGER, \

price INTEGER)")

## Описание спецификаций к модулям

Всего разработано 5 модулей:

* Аутентификация пользователя (SignUpView)
* Авторизация пользователя (SignInView)
* Страница теста (TestDetailView)
* Страница с вопросами по тесту (TestQuestionsView)
* Страница с результатами прохождения теста (TestResultView)

Взаимодействие между модулями можно наглядно увидеть по диаграммам классов из раздела 2.4.

## Описание модулей

**Аутентификация пользователя** – на данной странице находится форма для регистрации пользователя. Для регистрации нужно заполнить все поля, а именно:

* Имя пользователя
* Пароль
* Повторённый пароль
* Группа

После регистрации вы автоматически авторизуетесь. (Приложение 1)

Рассмотрим функцию регистрации пользователей и построим блок схему:

Листинг 2. Функция регистрации пользователей

def post(self, request, \*args, \*\*kwargs):

form = SignUpForm(request.POST) # Форма на регистрацию

# Если данные валидны

if form.is\_valid():

user = form.save() # Сохраняем пользователя в бд

# Если пользователь существует, то логинем этого пользователя

if user is not None:

human = dict()

human = Human(

name = request.POST['username'],

group = request.POST['group']

)

human.save()

login(request, user)

return HttpResponseRedirect('/') # Возращение на главную страницу

return render(request, 'moon\_test\_app/signup.html', context={

'form': form,

})

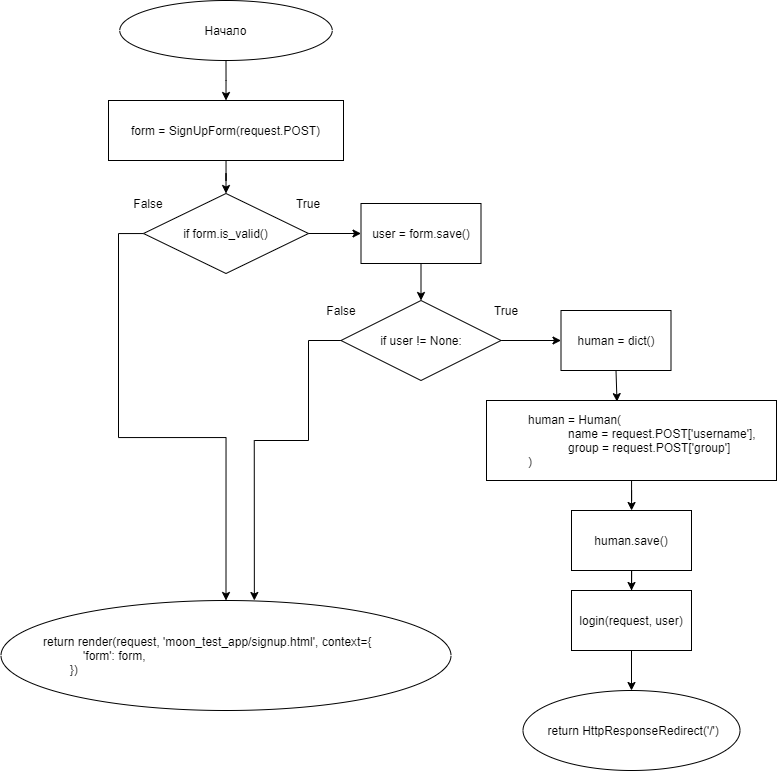


Рисунок 8. Блок схема функции post, класса SignUpView

**Авторизация пользователя** – содержит форму для входа пользователя (Приложение 2). Для этого нужно заполнить 2 поля:

Имя пользователя

Пароль

**Страница теста** – здесь располагается главное фото теста, название его, а также кнопка запуска теста. Также снизу есть таблица рекордов по тесту, который вы выбрали. Если тест ещё никто не проходил, то таблицы там не будет, а будет там некоторый текст (Приложение 3).

**Страница с вопросами по тесту** – на этой странице приведены вопросы (Приложение 4). Точнее будет сказать один вопрос. Но снизу есть кнопки:

Далее – открыть следующий вопрос

Назад – показать предыдущий вопрос

Закончить тест – окончить тест. Данная кнопка появляется, если вы находитесь на последнем вопросе.

**Страница с результатами прохождения теста** – на этой странице можно увидеть процент правильных ответов, по пройденному вами тесту, а также таблицу рекордов этого теста (Приложение 5).

## Описание тестовых наборов модулей

В некоторых модулях производится проверка данных на корректность, чтобы сообщить пользователю о проблеме и предотвратить аварийное завершение программы

**Тест 1. Проверка на правильный url теста**

Если пользователь решил самостоятельно вбить url нужного ему теста, и по каким-то причинам он вбил его с ошибкой, то выскачет ошибка 404. Данная ошибка скажет, что такой страницы не существует.

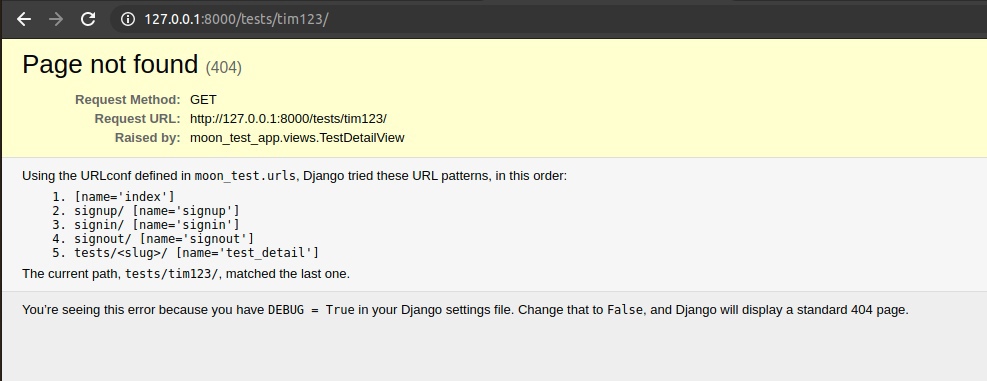


Рисунок 9. Ошбка при не правильном вводе url’a

**Тест 2. Проверка заполнения форм**

Если пользователь при аутентификации или авторизации допустил какую-либо ошибку, сверху ему выскочит плашка с ошибкой. Например, если пользователь неправильно повторил пароль, то плашка будет следующей:

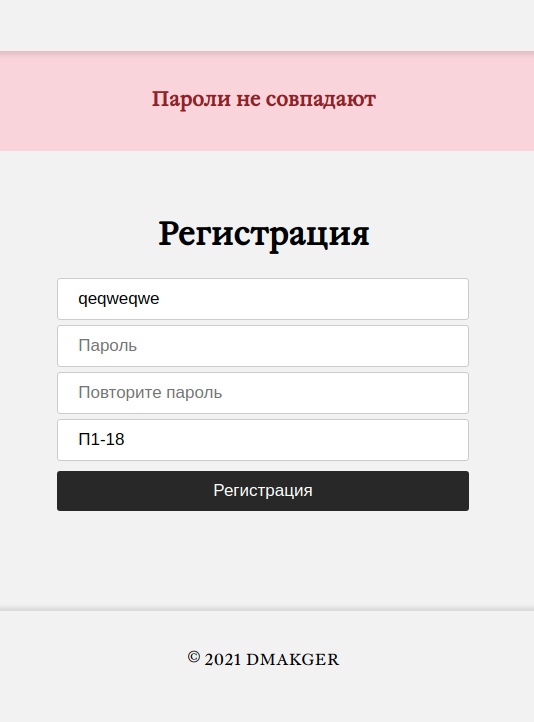


Рисунок 10. Ошибка при неправильном заполнении формы регистрации.

## Описание применения средств отладки

В ходе написания курсового проекта был найден баг. А именно при прохождении теста незарегистрированным пользователем (Рисунок 10):

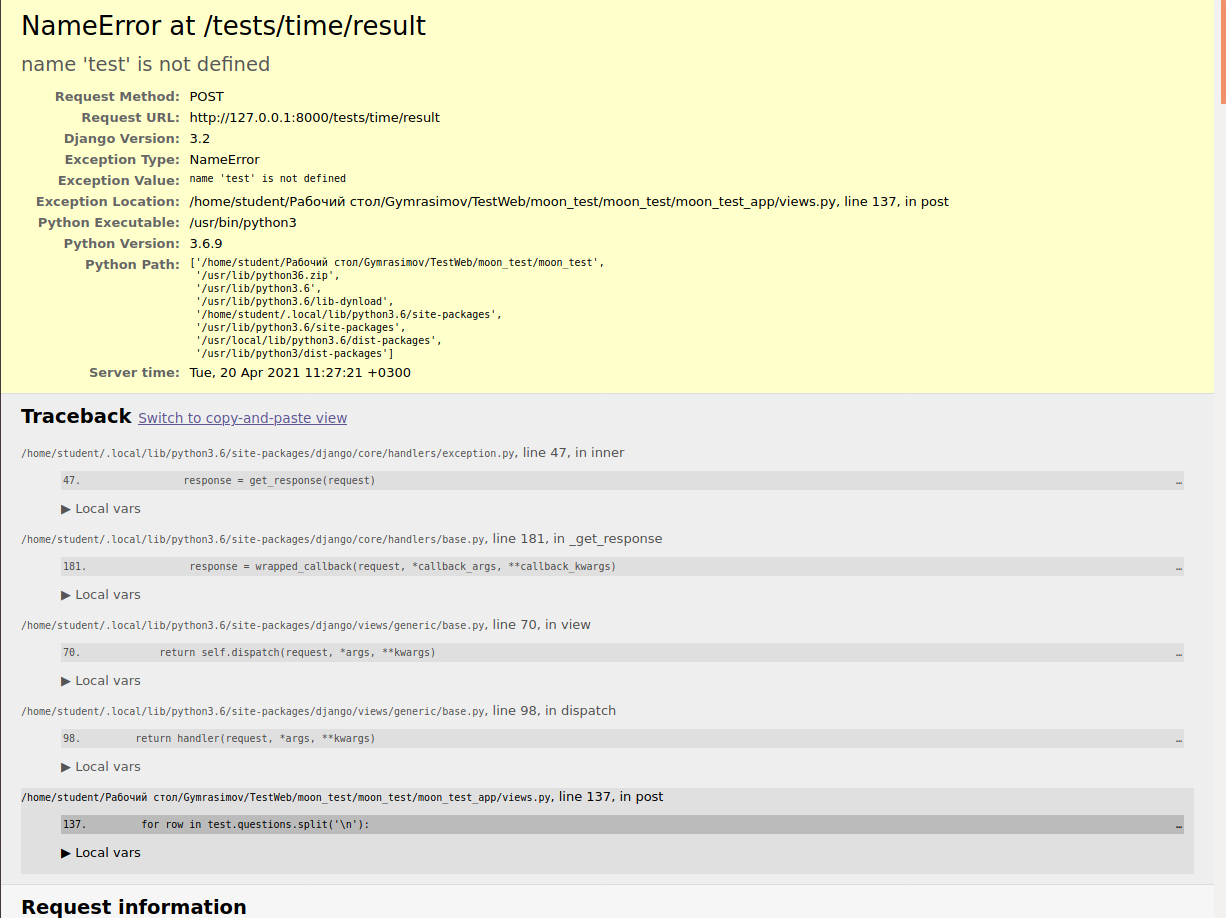


Рисунок 11. Ошибка при прохождении теста незарегистрированным пользователем

При проверке кода были исправлены найденные ошибки, в результате при запуске программы ошибок не было (Рисунок 11):

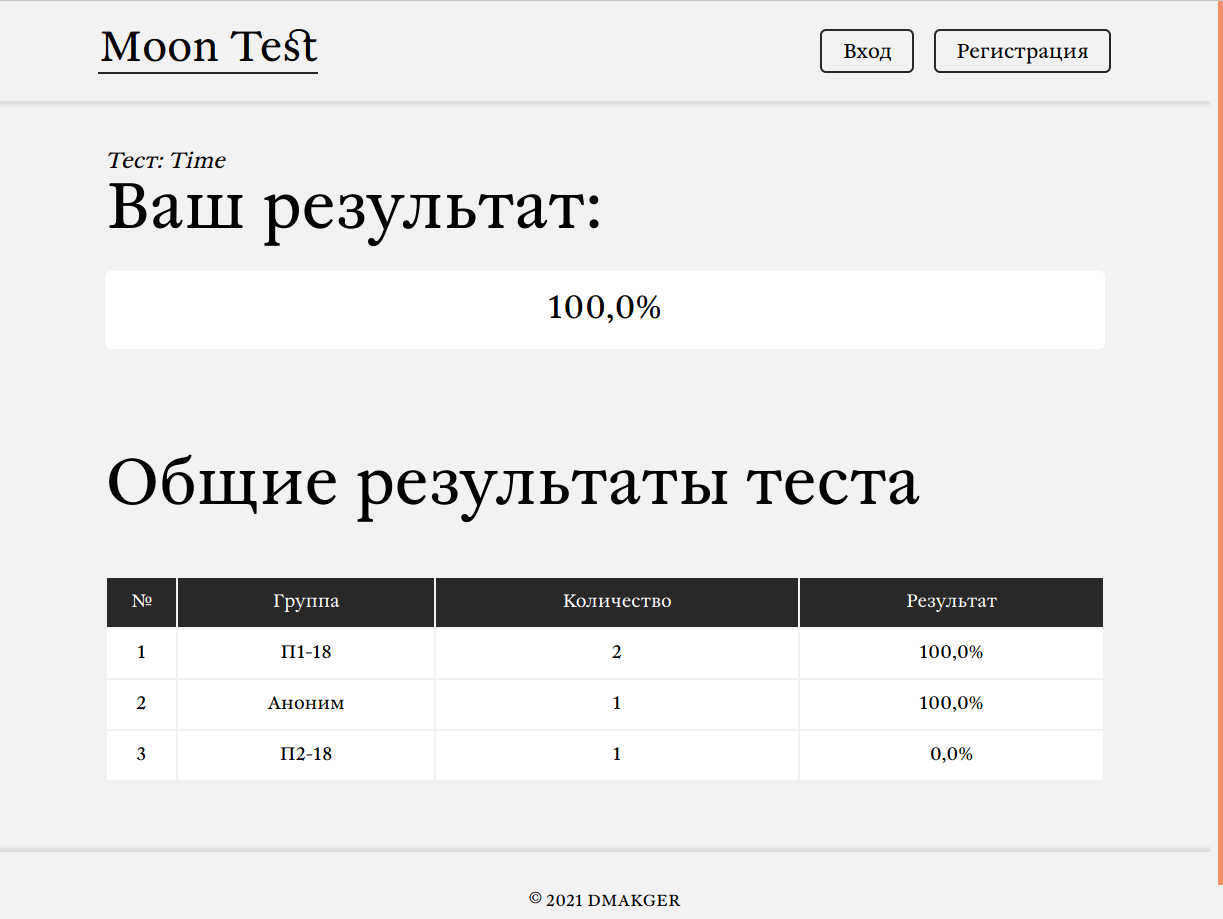


Рисунок 12. Успешная выдача результатов.

# Эксплуатационная часть

## Руководство оператора

### Назначение программы

Разработанный сайт позволяет проходить тесты, получать, а также с лёгкостью их создавать. Каждый тест хранится в базе данных, как и результаты пользователей и данные, которые ввели они.

### Условия выполнения программы

Любое устройство, выпущенное после 2012 года.

Браузеры: Edge, Firefox, Chrome, Safari, Opera, Safari on iOS, Opera Mini, Android Browser, Opera Mobile, Chrome for Android, Firefox for Android, UC Browser for Android, Samsung Internet, QQ Browser, Baidu Browser, KaiOS Browser.

### Выполнение программы

При запуске сайта вам откроется стартовая страница со всеми тестами:



Рисунок 13. Главная страница

Вы можете зарегистрировать пользователя, для этого надо заполнить данную форму:

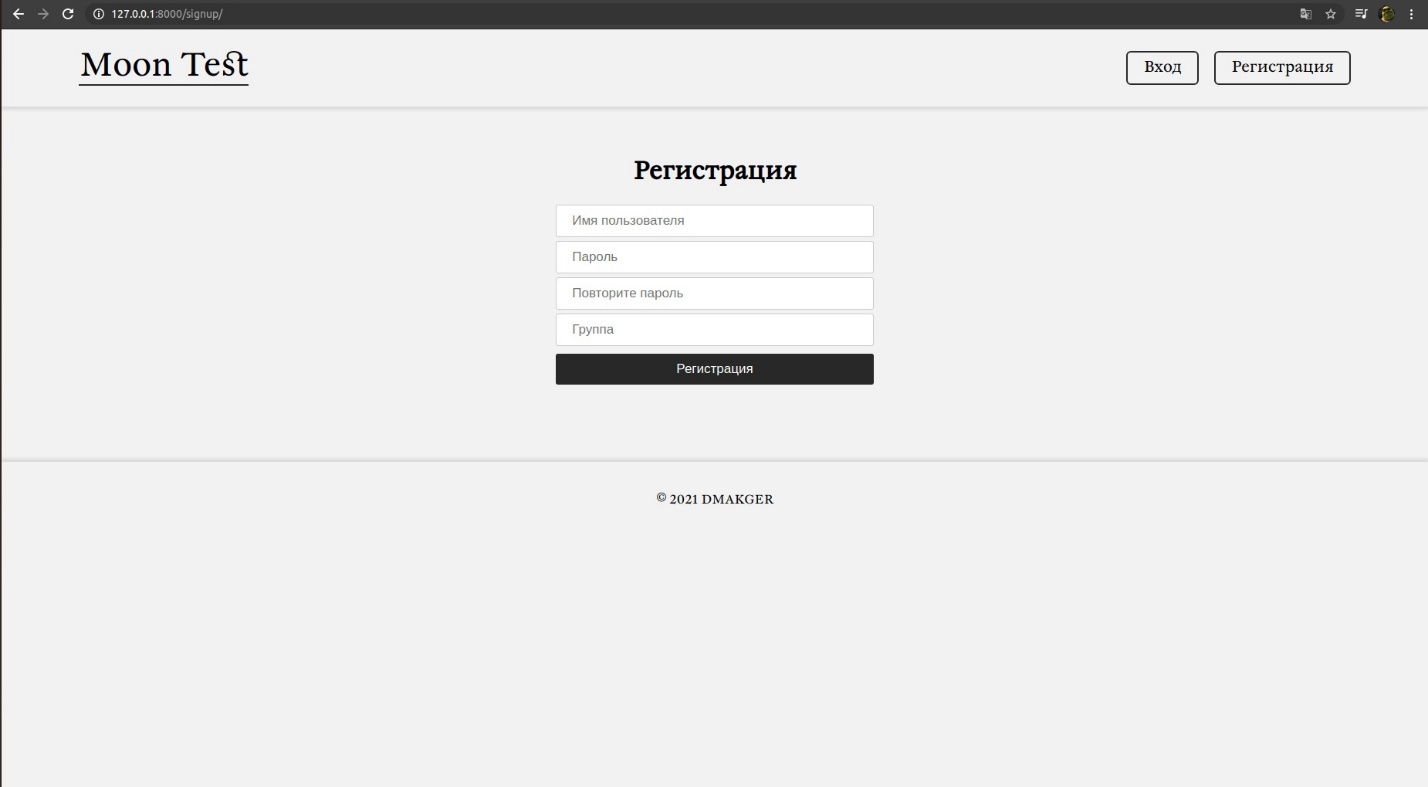


Рисунок 14. Форма аутентификации

Или войти в существующий аккаунт, для этого нужно ввести имя пользователя и пароль от него:

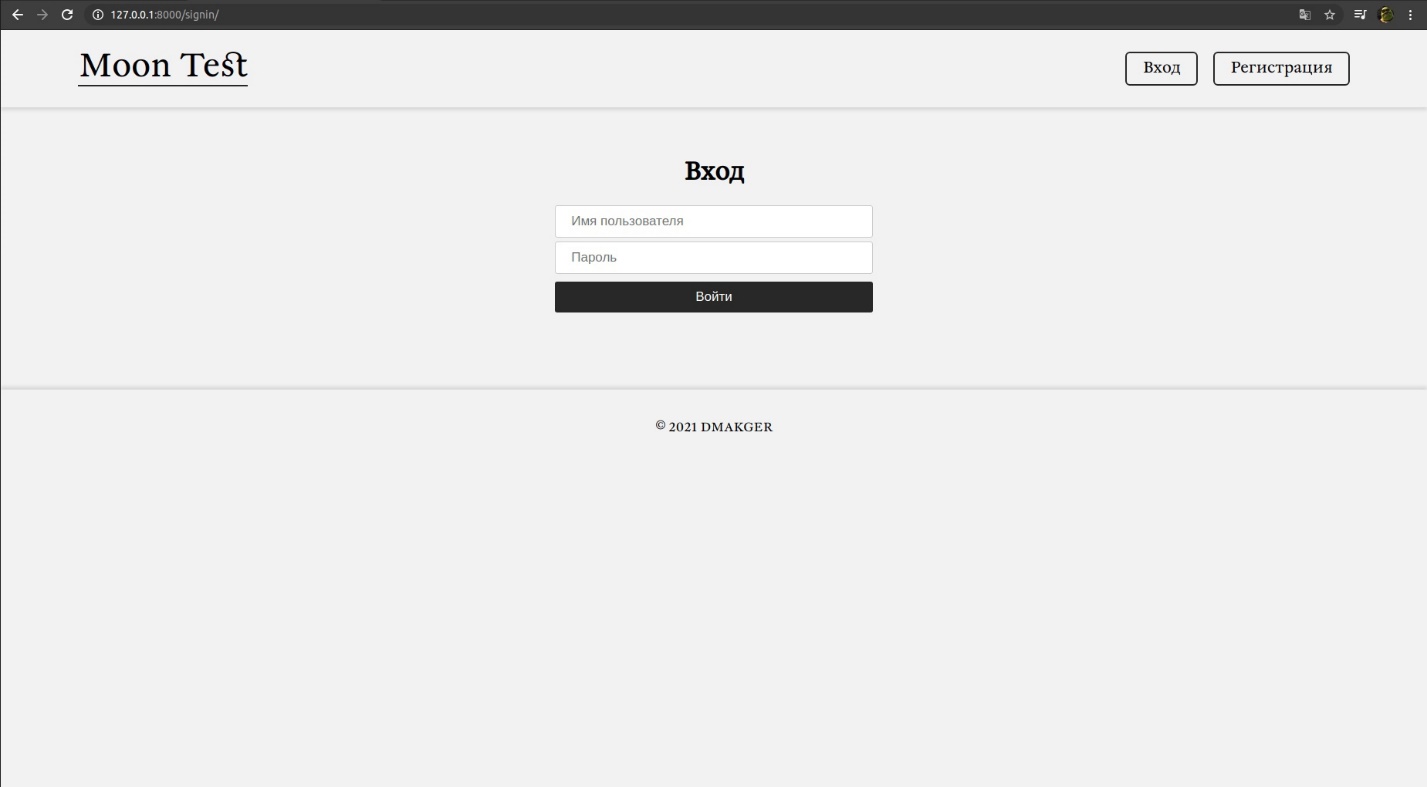


Рисунок 15. Форма авторизации

Либо же вы можете пропустить эти этапы. Тем не менее далее выбирается тест для прохождения, для этого нужно на него нажать.

После открытия теста вы можете увидеть таблицу результатов по данному тесту, а также кнопку «НАЧАТЬ ТЕСТ». Вам следует нажать на неё:

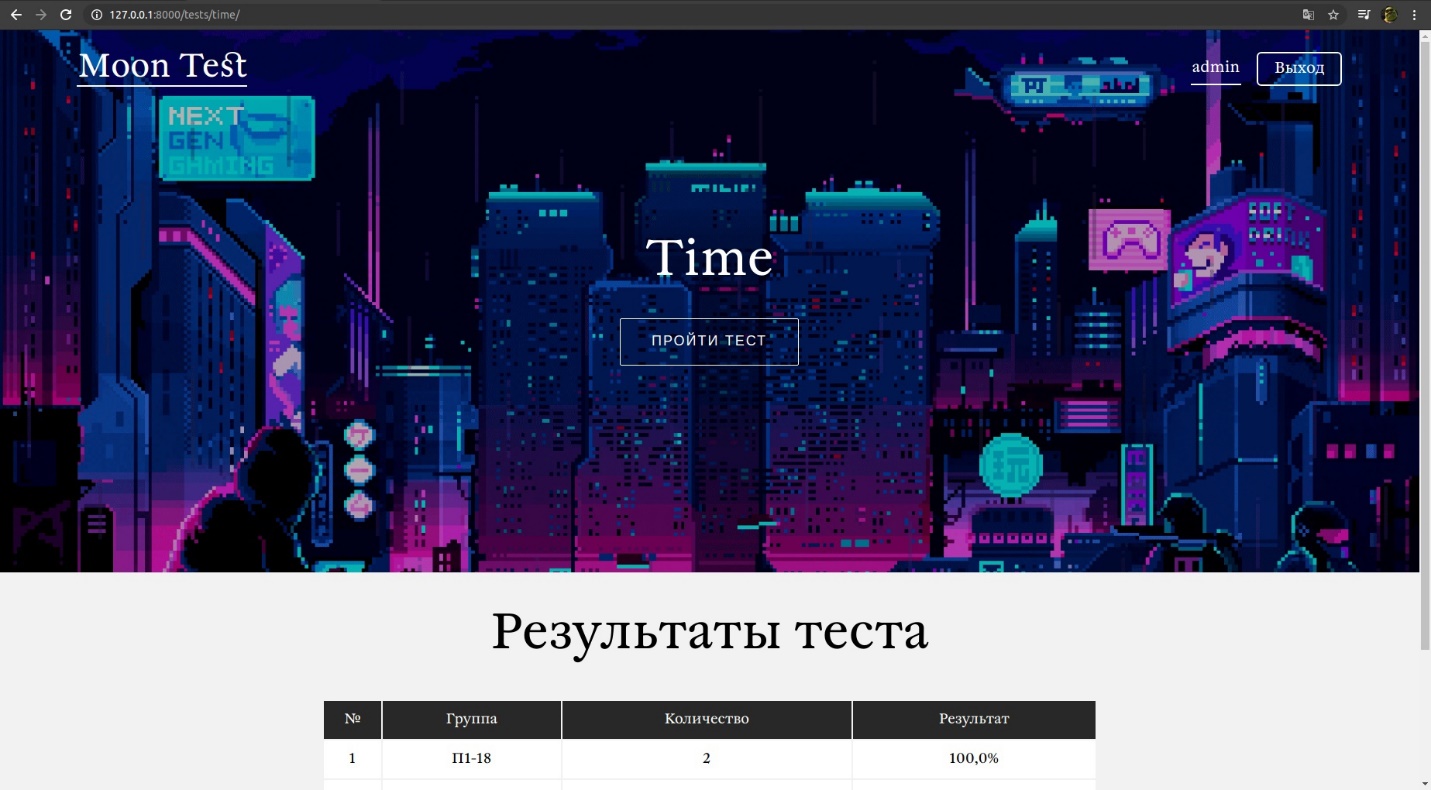


Рисунок 16. Страница теста

Теперь вам представлен вопросы к тесту. Чтобы их листать нажимайте кнопки «Далее» и «Назад».

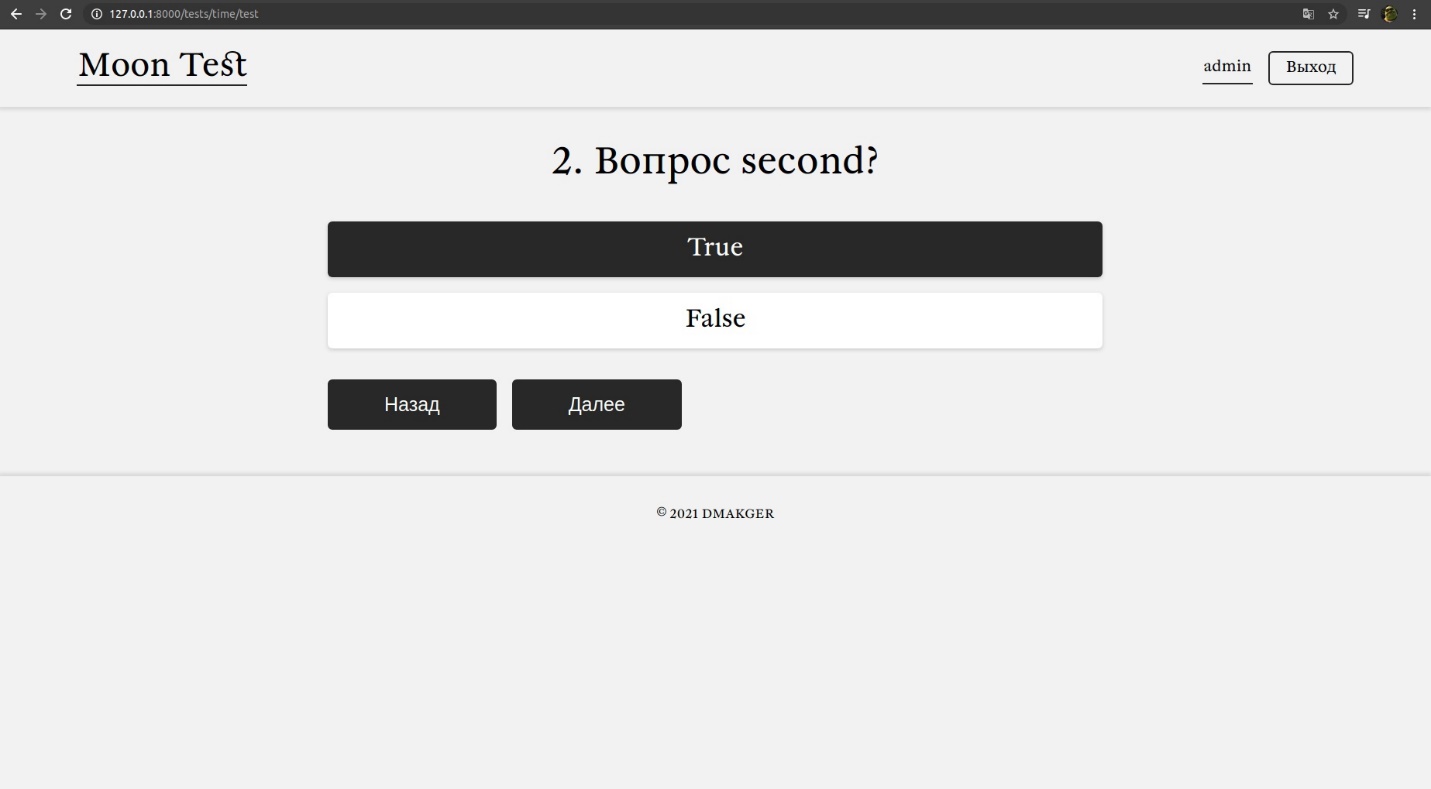


Рисунок 17. Вопросы к тесту

Для завершения теста нажмите кнопку, «Закончить тест». Она появляется при открытии последнего вопроса:

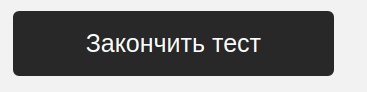


Рисунок 18. Кнопка завершения теста

После завершения теста вы можете увидеть процентное количество правильных ответов, а также таблицу результатов по группам.

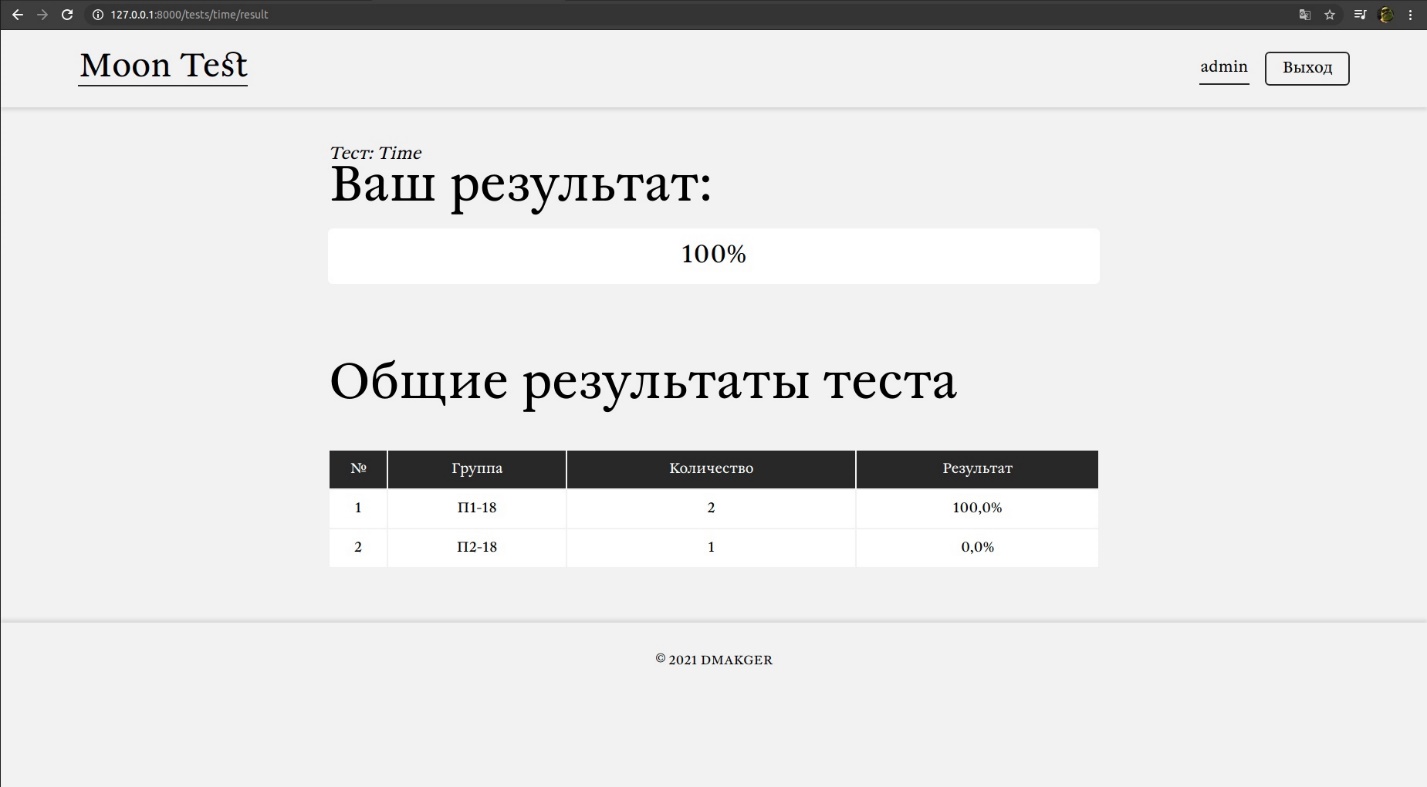


Рисунок 19. Результат теста

На любом из этих этапов, вы можете вернуться на главный экран. Для этого вам нужно нажать на логотип.



Рисунок 20. Логотип

### Сообщение оператору

При неправильном вводе данных в форму регистрации или авторизации появиться сообщение с допущенной ошибкой:

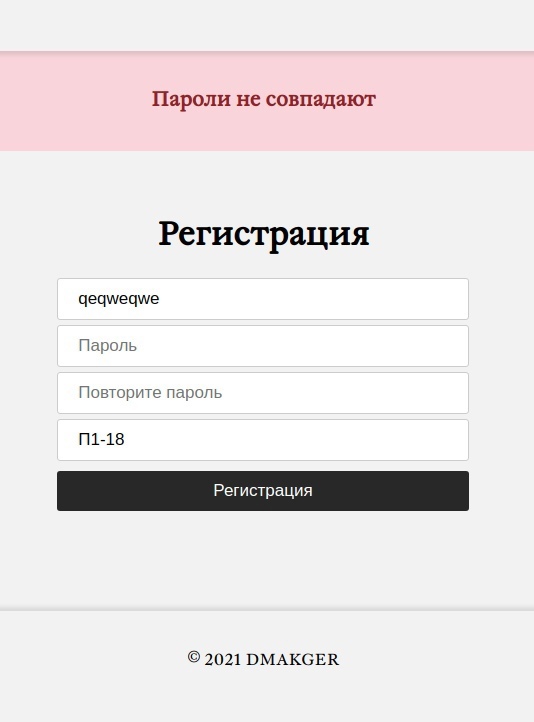


Рисунок 21. Сообщение об ошибке

**Причина:** неправильное заполнение форм.

**Действия сайта:** сайт не регистрирует/авторизует пользователя.

**Действия оператора:** правильно заполнить форму.

## To-Do лист

* Добавить профиль пользователю
* Пагинация
* Возможность создавать каждому пользователю тест на глобальном и локальном уровне.
* Добавить другие виды тестов
* Создание опросников
* Умное тестирование

# Заключение

Благодаря данному сайту проводить тестирование станет намного легче, так как не придётся тратить бумагу и чернила на написание теста. Также решение теста возможно будет в режиме реального времени из любой точки мира, то есть дистанционно. Данный проект может быть использован в учебных целях, а также для обучения созданию динамических сайтов.

# Список литературы и интернет ресурсов

1. 16Personalities

https://www.16personalities.com/

1. testometrika

<https://testometrika.com/tests/>

1. Викиум

https://wikium.ru/

1. HTML

<https://skillbox.ru/media/code/chto_takoe_html/>

1. **CSS**

<https://programfiles.info/css/chto-takoe-css/>

1. JavaScript

<https://promo.ingate.ru/seo-wikipedia/java-script/>

1. Django

<https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/Introduction>

1. Visual Studio Code

<https://code.visualstudio.com/docs>

1. Lucid – сайт для построения диаграмм

<https://lucid.app/>

# Приложения

**Приложение 1. views.py**

from django.http.response import HttpResponseRedirect

from .forms import SignUpForm

from django.shortcuts import render

from django.views import View

from .models import Human

from django.contrib.auth import login

class SignUpView(View):

def get(self, request, \*args, \*\*kwargs):

form = SignUpForm() # Форма на вход

return render(request, 'moon\_test\_app/signup.html', context={

'form': form,

})

def post(self, request, \*args, \*\*kwargs):

form = SignUpForm(request.POST) # Форма на регистрацию

# Если данные валидны

if form.is\_valid():

user = form.save() # Сохраняем пользователя в бд

# Если пользователь существует, то логинем этого пользователя

if user is not None:

human = dict()

human = Human(

name = request.POST['username'],

group = request.POST['group']

)

human.save()

login(request, user)

return HttpResponseRedirect('/') # Возращение на главную страницу

return render(request, 'moon\_test\_app/signup.html', context={

'form': form,

})

**Приложение 2. views.py**

from django.http.response import HttpResponseRedirect

from .forms import SignInForm

from django.shortcuts import render

from django.views import View

from django.contrib.auth import login

from django.contrib.auth import authenticate

class SignInView(View):

def get(self, request, \*args, \*\*kwargs):

form = SignInForm() # Форма на вход

return render(request, 'moon\_test\_app/signin.html', context={

'form': form,

})

def post(self, request, \*args, \*\*kwargs):

form = SignInForm(request.POST) # Форма на регистрацию

# Если данные валидны

if form.is\_valid():

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

# Если пользователь существует, то логинем этого пользователя

if user is not None:

login(request, user)

return HttpResponseRedirect('/')

return render(request, 'moon\_test\_app/signin.html', context={

'form': form,

})

**Приложение 3. views.py**

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

from django.views import View

from .models import Test, ResultTest

class TestDetailView(View):

def get(self, request, slug, \*args, \*\*kwargs):

test = get\_object\_or\_404(Test, url=slug)

result\_groups = ResultTest().get\_result\_group(test.id)

return render(request, 'moon\_test\_app/test\_detail.html', context={

'test': test,

'groups': result\_groups

})

**Приложение 4. views.py**

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

from django.views import View

from .models import Test

class TestQuestionsView(View):

def get(self, request, slug, \*args, \*\*kwargs):

test = get\_object\_or\_404(Test, url=slug)

questions = dict()

question = dict()

a\_true = dict()

count = 0

number = 1

for row in test.questions.split('\n'):

if row[-1] == '\r':

row = row[:-1]

if row == '':

questions[number] = question

question = dict()

count = -1

number += 1

else:

if count == 0:

question['q'] = row

else:

if row[0] == '\*':

row = row[1:]

a\_true[number] = count

if not question.get('a'):

question['a'] = dict()

question['a'][count] = row

count += 1

if question:

questions[number] = question

number += 1

return render(request, 'moon\_test\_app/test\_qa.html', context={

'questions': questions,

'a\_true': a\_true,

'test': test,

})

**Приложение 5. views.py**

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

from django.views import View

from .models import Test, Human, ResultTest

class TestResultView(View):

def get(self, request, \*args, \*\*kwargs):

test = ResultTest() # Форма на вход

return render(request, 'moon\_test\_app/<slug>/test\_result.html', context={

'test': test,

})

def post(self, request, slug, \*args, \*\*kwargs):

print('---')

print(request.POST)

# Получаем список ответов пользователя. (При этом удалив username и csrfmiddlewaretoken)

result = [key for key in sorted(request.POST.values()) if key != '' and key[0].isdigit()]

# Формируем список правельных ответов

a\_true = list()

test = get\_object\_or\_404(Test, url=slug)

count = 0

number = 1

for row in test.questions.split('\n'):

if row[-1] == '\r':

row = row[:-1]

if row == '':

count = -1

number += 1

elif row[0] == '\*':

a\_true.append(f"{number}-{count}")

count += 1

# Кол-во правельных ответов

n\_answer\_true = 0

for answer in result:

if answer in a\_true:

n\_answer\_true += 1

# Процент правильных ответов

percent\_true\_answer = round(n\_answer\_true/len(a\_true) \* 100, 2)

# Если незарегистрированный пользователь

if request.POST['username'] == '':

result\_test = {

'test\_id': test.id,

'result': percent\_true\_answer,

}

# Если зарегистрированный пользователь

else:

result\_test = ResultTest.objects.filter(name=request.POST['username'], test\_id=test.id)

print(result\_test)

# Если пользователь ещё не проходил тест, то регистрируем его

if not result\_test:

print('not')

group = Human.objects.filter(name=request.POST['username'])

# Пользователь не указал свою группу

if not group:

group = 'Аноним'

# Пользователь указал свою группу

else:

group = group[0].group

result\_test = ResultTest(

name=request.POST['username'],

test\_id=test.id,

group=group,

result=percent\_true\_answer,

)

# Пользователь уже проходил тест

else:

print('yes')

result\_test = result\_test[0]

result\_test.result = percent\_true\_answer

# Если десятичние числа равны 0, то приводим к int

if result\_test.result == int(result\_test.result):

result\_test.result = int(result\_test.result)

result\_test.save()

return render(request, 'moon\_test\_app/test\_result.html', context={

'test': test,

'result\_test': result\_test,

'all\_result\_test': ResultTest().get\_result\_group(test.id)

})