МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ШКОЛА КОМПЬЮТЕРНЫХ НАУК

КАФЕДРА ПРОГРАМММНОГО ОБЕСПЕЧЕНИЯ

ОТЧЕТ

О РАЗРАБОТКЕ ПРОЕКТА ПО ДИСЦИПЛИНЕ

«Интеллектуальные системы поддержки принятия решений»

на тему: “ Разработка веб-сервиса CheckUs для проверки программного кода студентов на языке Python”

|  |  |  |
| --- | --- | --- |
| Выполнил  обучающийся 4 курса,  МОиАИС-21.03 группы | (подпись) | Струнин Виталий Дмитриевич |
| Выполнил  обучающийся 4 курса,  МОиАИС-21.03 группы | (подпись) | Бовыкина Екатерина Евгеньевна |
| Научный руководитель  Проекта | (подпись) | Павлова Елена Александровна |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

Защищен: «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 202\_\_ г.

Результаты защиты \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  (подпись руководителя проекта)

Оглавление

[Введение 4](#_Toc185536393)

[Глава 1. Теоретическая часть проекта 5](#_Toc185536394)

[1.1. Описание специфики предметной области 5](#_Toc185536395)

[1.1.1 Ключевые пользователи 5](#_Toc185536396)

[1.1.2 Проблема 5](#_Toc185536397)

[1.1.3 Идея проекта 6](#_Toc185536398)

[1.1.4 Требования к проекту и учитываемые ограничения 6](#_Toc185536399)

[1.1.5 Теоретическое и практическое обоснование проекта 6](#_Toc185536400)

[1.1.6. Собственные теоретические и практические решения, найденные и реализованные в проекте 7](#_Toc185536401)

[1.2. Получение референсного кода 10](#_Toc185536402)

[1.2.1. Выбор модели 10](#_Toc185536403)

[1.2.1. Промт для генерации кода 10](#_Toc185536404)

[1.3. Анализ на основе AST 11](#_Toc185536405)

[1.3.1. Анализ последовательности узлов 12](#_Toc185536406)

[1.3.2. Метрика Tree Similarity Edit Distance (TSED) 14](#_Toc185536407)

[1.3.3 Введение весов узлов AST 16](#_Toc185536408)

[1.4. Генерация и использование тестов 18](#_Toc185536409)

[1.4.1. Тесты stdin/stdout 18](#_Toc185536410)

[1.4.2. Мутационное тестирование кода (MBFL) 19](#_Toc185536411)

[Глава 2. Практическая часть проекта 22](#_Toc185536412)

[2.1. Принципы и детали реализации проекта 22](#_Toc185536413)

[2.1.1. Этапы алгоритма проверки кода студента 22](#_Toc185536414)

[2.1.2. Описание работы сервиса 23](#_Toc185536415)

[2.1.3. Функциональные возможности проекта. 23](#_Toc185536416)

[2.2. Пользовательский (или иной внешний) интерфейс проекта. 23](#_Toc185536417)

[2.2.1. Интерфейс добавления заданий 23](#_Toc185536418)

[2.2.2. Основной интерфейс на этапе MVP 24](#_Toc185536419)

[2.2.3. Обратная связь о проверке кода 25](#_Toc185536420)

[2.3. Организация тестирования проекта 30](#_Toc185536421)

[2.3.1. Цели тестирования 30](#_Toc185536422)

[2.3.2. Исполнители 30](#_Toc185536423)

[2.3.3. Результаты тестирования на основе обратной связи от пользователей 31](#_Toc185536424)

[Заключение 32](#_Toc185536425)

[Список литературы 33](#_Toc185536426)

[Приложения 34](#_Toc185536427)

# **Введение**

Образование в области программирования требует постоянного взаимодействия между студентами и преподавателями, особенно в контексте проверки и анализа программного кода. Дисциплина «Программирование и основы алгоритмизации» предполагает выполнение практических заданий, которые необходимо оценивать не только с точки зрения корректности выполнения, но и с учётом качества написания кода, соответствия стилевым рекомендациям и полноты решения поставленной задачи.

В текущей системе взаимодействия процесс проверки программных заданий студентов имеет ряд несовершенств. Преподаватели вынуждены вручную анализировать большое количество работ, что требует временных затрат и усилий. Ограниченное время, отведённое на занятия, часто не позволяет уделить должное внимание каждому студенту, снижая качество обратной связи и объективность оценки. В свою очередь, студенты сталкиваются с трудностями получения оперативной и детализированной обратной связи по своим работам, что может замедлить процесс их обучения и повысить вероятность ошибок в дальнейшем.

Цель: разработать веб-сервис проверки программных кодов студентов, обеспечивающую проверку на соответствие заданию и установленным ограничениям на используемые методы, с последующей выдачей комментариев о проверке и указанием на возможные допущенные ошибки.

Задачи:

1. Провести обзор существующих подходов
2. Написать парсер для получения кодов из отчётов студентов для формирования датасета
3. Разметить коды для тестирования алгоритмов
4. Разработать алгоритм проверки программного кода студентов с генерацией тестов и сравнением структуры с референсным кодом
5. Разработать клиентскую и серверную части веб-сервиса

# **Глава 1. Теоретическая часть проекта**

## **1.1. Описание специфики предметной области**

### **1.1.1** **Ключевые пользователи**

Пользователями проекта являются две основные группы: студенты и преподаватели, вовлечённые в процесс обучения дисциплине «Программирование и основы алгоритмизации». Для студентов ключевыми потребностями являются возможность получения оперативной обратной связи по написанному коду и быстрая проверка решений на наличие ошибок, соответствие требованиям задачи, стиль написания и потенциальные недочёты. Они ожидают объективности в оценке их работы, минимизации влияния человеческого фактора, а также упрощения процесса сдачи и получения оценки.

Преподаватели, в свою очередь, сталкиваются с необходимостью проверки множества студенческих работ, что требует значительных временных затрат. Их основной запрос связан с автоматизацией проверки заданий, которая сократит время на рутинные задачи и обеспечит возможность настройки критериев оценки. Это позволит учитывать индивидуальные потребности студентов, уровень их подготовки и текущие темы, изучаемые в курсе.

### **1.1.2 Проблема**

В настоящее время система автоматической проверки заданий отсутствует. Существующий процесс проверки требует, чтобы студенты либо отправляли свои файлы через платформу LMS, либо демонстрировали выполнение задания лично на занятии. Преподаватель вручную проверяет каждую работу, тестирует её выполнение и предоставляет обратную связь.

Однако этот подход имеет ограничения. Во время стандартного занятия продолжительностью 90 минут преподаватель может уделить внимание лишь ограниченному числу студентов. Например, при 30 студентах каждому уделяется максимум 3 минуты. Если отдельные студенты требуют более детальной проверки или дополнительной обратной связи, остальные могут не успеть получить полноценную оценку. Отсутствие автоматизации также увеличивает нагрузку на преподавателя, снижая объективность оценки из-за усталости или нехватки времени.

### **1.1.3 Идея проекта**

Разработать веб-сервис для автоматической проверки программных кодов студентов, обучающихся предмету «Программирование и основы алгоритмизации» на Python, с использованием нейросетей. Сервис должен оценивать корректность решений в соответствии с заданиями, что позволит автоматизировать и ускорить процесс проверки заданий.

### **1.1.4 Требования к проекту и учитываемые ограничения**

Среди функциональных требований основное внимание уделено автоматической проверке студенческого кода. Сервис должен проверять корректность выполнения задач, давать обратную связь в виде комментария, а также, по возможности, указывать на допущенную студентом ошибку.

Для удобства взаимодействия с веб-сервисом, его интерфейс должен быть простым и интуитивно понятным.

К нефункциональным требованиям относится надежность: сервис должна корректно обрабатывать некорректный код, предоставляя соответствующие сообщения об ошибках.

Веб-сервис разрабатывается с учётом ряда ограничений, включая необходимость учитывать текущий уровень подготовки студентов. Он должен позволять задавать пройденные или ещё не изученные темы курса, а также контролировать использование методов, применяемых при решении задач.

### **1.1.5 Теоретическое и практическое обоснование проекта**

Основой разрабатываемого проекта является гипотеза о том, что абстрактное синтаксическое дерево (AST) программного кода может быть эффективно использовано для сравнения нескольких программ на предмет их структурного и логического сходства. Ключевая идея заключается в применении генеративных моделей для автоматической генерации референсного программного кода на основе подробного промта. Этот промт включает контекст задания и текст задачи из практической работы. Полученный референсный код будет служить эталоном для последующего сравнения с программными кодами студентов.

Используемые технологии:

1. Python 3
2. Django
3. Модуль AST (построение Abstract Syntax Tree)
4. Модуль difflib (сравнение AST)

### **1.1.6. Собственные теоретические и практические решения, найденные и реализованные в проекте**

Для реализации проекта был предоставлен доступ к курсу “Программирование и основы алгоритмизации” на платформе LMS, в котором имеется следующая информация:

1. Список практических работ
2. Информация, необходимая для практической работы (теория, задачи)
3. Отправленные студентами отчёты, содержащие программный код

На этапе разработки сервиса нам необходим программный код, написанный студентом. Для одной и той же задачи необходимо иметь несколько реализаций, чтобы обеспечить репрезентативность выборки. Для этого была проанализирована структура отчета студента и выявлена следующая закономерность (Рисунок 1): программный код для решения каждой задачи находится в отдельном блоке между шаблонными фрагментами «Текст программы на языке Python” и “Результат тестирования программы на языке Python»:

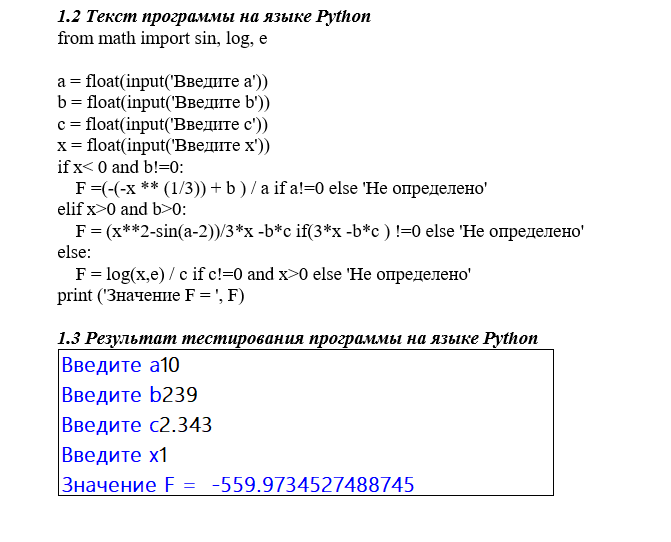


Рис.1 Блок между шаблонными фрагментами

Из структуры отчёта следует, что можно разработать программу для извлечения программного кода студента для каждой задачи.  Разработанный парсер принимает путь на папку с архивами с сайта LMS, в котором находятся отчеты, извлекает программный код каждого задания и сохраняет в папку соответствующей практической работы (Рисунок 2).

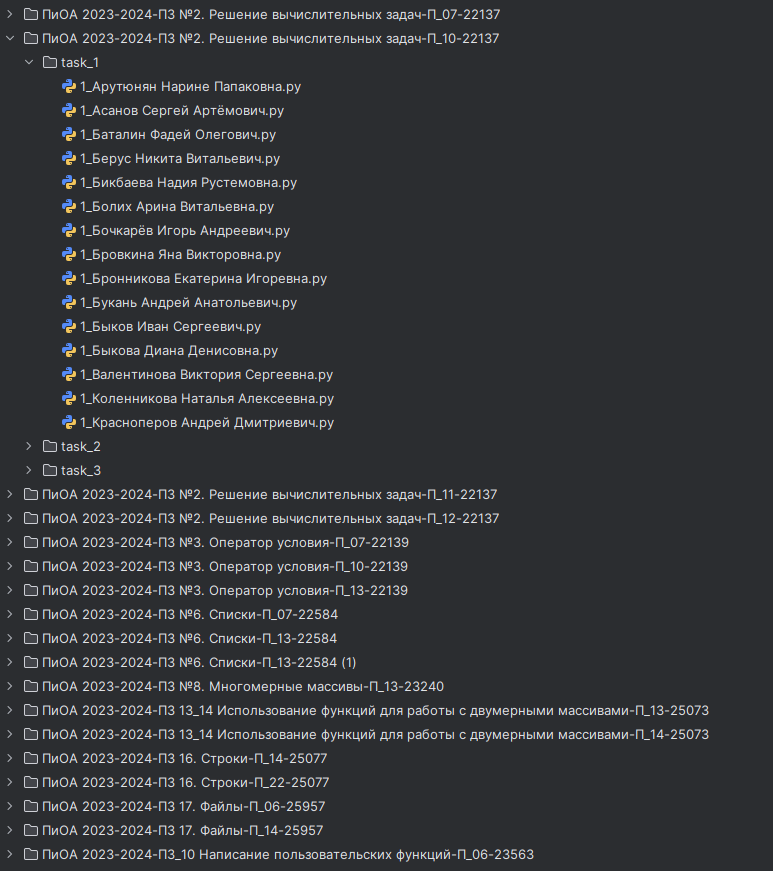


Рис.2 Полученные в результате работы парсера папки с программными кодами

В результате было получено 856 программных кодов по первой практической работе, написанные студентами первого курса различных направлений. Поскольку задание практической работы предусматривало выбор для решения любых 4 задач из 12 возможных, каждая папка могла содержать коды, относящиеся к различным заданиям.

Автоматическая классификация кодов оказалась затруднительной из-за разнообразия подходов к реализации. Поэтому было принято решение обработать данные вручную. В результате классифицировано 760 кодов, корректно решающих поставленные задачи.

В итоговый датасет для анализа не вошли:

* работы, не относящиеся к данной практической работе
* коды, поврежденные в процессе обработки парсером
* часто повторяющиеся коды с однотипным решением

## **1.2. Получение референсного кода**

### **1.2.1. Выбор модели**

Для проведения проверки требовался образец — код, который корректно решает поставленную задачу и структурно соответствует её требованиям. Первоначально, для получения референсного кода было принято решение использовать генеративные модели. Рассматривались открытые модели с возможностью локального развёртывания, такие как PolyCoder, CodeParrot, SantaCoder и другие. Однако тестирование показало, что из-за ограниченных вычислительных ресурсов эти модели не смогли обеспечить ожидаемый результат. В связи с этим было принято решение отказаться от данного подхода в пользу использования закрытых моделей с доступом через API. После анализа альтернатив было принято решение использовать модель GPT-4o . Составление подробного промта для генерации кода дало ожидаемый результат.

### **1.2.1. Промт для генерации кода**

Структура промта для генерации референсного кода:

* Задается роль**:** Модель выступает в роли части веб-сервиса для проверки студенческого кода, создавая референсные решения для сравнения.
* Определяется задача**:** написать эталонный код на языке Python, который будет использоваться для проверки правильности решений студентов.
* Устанавливаются ограничения**:**
  + Код должен быть кратким и простым, без сложных конструкций.
  + Использовать можно только те инструменты, которые указаны в контексте задания.
  + Объявление функций запрещено, если это не указано в контексте.
  + Данные вводятся с клавиатуры.
  + Формат ответа — чистый код на Python без комментариев.
* Задается контекст**:** определяет доступные инструменты, библиотеки и разрешенные конструкции в коде.
* Задается задание**:** описывает конкретную задачу, которую нужно решить с помощью эталонного кода.

GPT-4o продемонстрировал корректные результаты, обеспечивая коды, соответствующие заданию, которые можно использовать в качестве эталонных для сравнения.

После получения кода его можно просмотреть и проверить, чтобы определить, соответствует ли он требованиям задачи. В случае несоответствия можно отправить запрос на повторную генерацию.

На текущем этапе разработки получение кода через API не было добавлено в веб-сервис и выполнялось в отдельном консольном коде.

## **1.3. Анализ на основе AST**

Поскольку в основе лежала гипотеза возможности сравнения программных кодов на основе анализа их абстрактного синтаксического дерева (AST), необходимо было исследовать возможные подходы.

Абстрактное синтаксическое дерево (AST) — это структурированное представление исходного кода программы. Оно показывает:

* какие элементы кода присутствуют (например, вызовы функций, операции присваивания, вычисления);
* как они структурированы и взаимодействуют друг с другом.

Вместо текстового сравнения (которое чувствительно к форматированию, комментариям и незначительным изменениям) AST-сравнение анализирует семантические аспекты кода, такие как:

* используемые переменные,
* логика операций,
* структура выражений
  + 1. **Анализ** **последовательности узлов**

Исходный код анализировался с использованием библиотеки ast, которая преобразует текстовое представление Python-кода в абстрактное синтаксическое дерево (AST). Этот процесс позволил получить структурированное представление программы, независимое от деталей синтаксиса, таких как форматирование или комментарии.

Однако такой подход оказался чувствителен к минимальным изменениям в коде, например, к различиям в именах переменных или небольшим вариациям в реализации. Чтобы минимизировать эту чувствительность, было принято решение извлекать только ключевые элементы программы: вычислительные выражения, вызовы функций и операции над переменными. Это позволило сосредоточиться на логике кода, исключив второстепенные аспекты.

Извлечённые элементы логики преобразовывались в универсальное представление в виде строк, описывающих структуру дерева AST. Для оценки схожести использовался алгоритм SequenceMatcher, который вычисляет коэффициент сходства последовательностей [3].

*,*  (1.1)

где:

* |LCS∣ — длина наибольшей общей подпоследовательности (Longest Common Subsequence) между двумя строками.
* ∣Строка1​∣ и ∣Строка2∣ — длины строк, представляющих вычислительные выражения двух программ.

Определялся медианный процент сходства по всем кодам конкретного задания, а также доверительным интервалом. Доверительный интервал используется для оценки надежности среднего значения процента структурного сходства между эталонным кодом и кодами студентов. Это статистическая мера, показывающая диапазон, в котором, с определённой вероятностью (95%), находится истинное среднее значение.

*, (1.2)*

где:

* t — значение из t-распределения для уровня доверия (95%) и степени свободы (n−1);
* ​ — стандартная ошибка среднего.

Такой подход хорошо подходит для анализа задач с простыми и стандартными решениями, где используется общий алгоритм. Однако на заданиях, допускающих большую вариативность подходов к решению, метод показывал более низкую точность, так как незначительные различия в реализации могли приводить к занижению оценки схожести. Результаты представлены в таблице 1.

Таблица 1.

Результаты сравнения на основе анализа последовательности узлов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № задания и текст | Количество сравниваемых кодов | Медианный процент сходства | Средний процент сходства | Доверительный интервал |
| 1. вычислить | 227 | 77.14% | 67.53% | 64.39%-70.68% |
| 4. Вычислить углы треугольника, зная его стороны | 100 | 19.42% | 25.01% | 21.34%-28.68% |

### **1.3.2. Метрика Tree Similarity Edit Distance (TSED)**

Изучив статью «Similarity Evaluation with Abstract Syntax Tree Edit Distance», было найдено описание и применение метрики TSED при анализе AST.

Подход на основе анализа последовательности узлов: сравнивал извлечённые строки, описывающие структуру AST, с помощью SequenceMatcher, что основывалось на текстовом представлении дерева.

Подход на основе метрики TSED: вычисляет расстояние редактирования между двумя AST, анализируя их структуру на уровне узлов дерева.

Метод TSED использует обход дерева (ast.walk) для извлечения всех узлов из AST. На основе узлов двух деревьев формируется матрица стоимостей, где каждая стоимость отражает степень различия между узлами (например, разные типы узлов имеют максимальную стоимость) [2].

(1.3)

где:

* n1,i и n2,j**​** — узлы из двух деревьев (дерева 1 и дерева 2), которые сравниваются.
* 1 — максимальная стоимость, которая присваивается, если узлы имеют разные типы. Это означает, что они считаются полностью несоответствующими.
* 0.5⋅∣Вес(n1,i​)−Вес(n2,j​)∣ — половина абсолютной разницы между весами узлов n1,i и n2,j​. Применяется, если типы узлов совпадают, но их свойства отличаются. Эта часть формулы оценивает степень различий в их свойствах.
* 0 — стоимость равна нулю, если узлы идентичны по типу и свойствам. Это указывает на полное соответствие узлов.

(1.4)

где:

* C — матрица стоимостей, элементы которой представляют значения функции
* Cost(n1,i,n2,j) Cost(n1,i​,n2,j​) для всех возможных пар узлов n1n и n2n.

 С помощью алгоритма оптимального назначения рассчитывается минимальная общая стоимость преобразования одного дерева в другое. Для интерпретации результатов расстояние нормализуется, преобразуясь в метрику сходства, выраженную в процентах.

(1.5)

Это приводит значение сходства в диапазон от 0% (полностью разные деревья) до 100% (идентичные деревья).

Основное преимущество подхода заключается в его устойчивости к изменениям кода, таким как переименование переменных или изменение порядка строк. Анализ стал более чувствительным к логическим и структурным аспектам кода, что позволяет оценивать его семантическое сходство даже при существенных различиях в реализации.

Данный подход позволил поднять процент сходства в кодах с большей вариативностью решений (Таблица 2).

Таблица 2.

Результаты сравнения двух методов анализа на основе задания №4

|  |  |  |  |
| --- | --- | --- | --- |
| Метод анализа | Медианный процент сходства | Средний процент сходства | Доверительный интервал |
| Анализ последовательности узлов | 19.42% | 25.01% | 21.34%-28.68% |
| Метрики TSED | 63.61% | 61.74% | 60.06%-63.42% |

### **1.3.3 Введение весов узлов AST**

Введение весов узлов AST позволяет учитывать значимость различных конструкций кода при сравнении. Для этого каждому типу узла в абстрактном синтаксическом дереве (AST) был назначен вес, отражающий его вклад в семантику программы. Например, более значимые конструкции, такие как определение функций (FunctionDef), циклы (For, While) и условия (If), получают больший вес, чем менее значимые элементы, такие как имена переменных (Name) или константы (Constant). При расчёте расстояния между узлами разница в их весах используется для определения степени их сходства. Узлы одинакового типа, но с разным весом, вносят меньшее расстояние, в то время как узлы разных типов имеют максимальное расстояние, равное большему из их весов. Чем значительнее вес узлов при измерении суммы дистанции для преобразования, тем меньшем процент сходства. Результаты сравнения представлены в таблице 3.

Таблица 3.

Результаты сравнения после введения веса узлов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод анализа | № задания и текст | Количество сравниваемых кодов | Медианный процент сходства | Средний процент сходства | Доверительный интервал |
| Анализ последовательности узлов | 1. вычислить | 227 | 77.14% | 67.53% | 64.39%-70.68% |
| 4. Вычислить углы треугольника, зная его стороны | 100 | 19.42% | 25.01% | 21.34%-28.68% |
| Метрика TSED + разные веса для узлов | 1. вычислить | 227 | 98.86% | 95.80% | 93.89%-97.72% |
| 4. Вычислить углы треугольника, зная его стороны | 100 | 95.73% | 95.08% | 94.28%-95.88% |

Результат работы программы должен определить, прошел ли код пороговое значение при анализе. Первоначально был выбран медианных процент по результатам анализа всех имеющихся кодов в датасете 95.73%. Результат сравнения по всему датасету представлен на рисунке 3

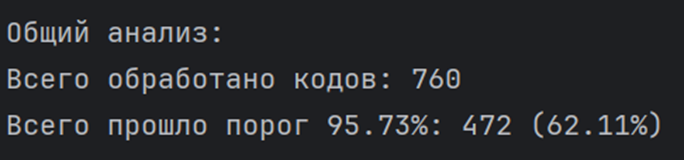


Рис.3 Результат прохождение анализа по медианному проценту кодов первой практической работы

Однако данный порог прошли всего 62% кодов с корректным решением задачи. В дальнейшем было выбрано медианное значение нижних границ доверительного интервала 90.87% (Рисунок 4).

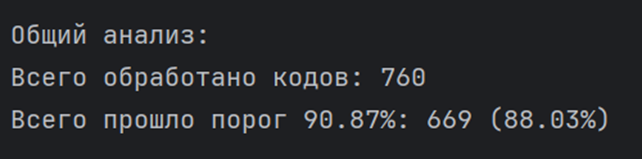


Рис.4 Результат прохождение анализа по медианному проценту нижней границы доверительного интервала кодов первой практической работы

Данный порог прошли уже 88% кодов.

Анализ на основе AST, несмотря на его структурный подход, оказался недостаточно эффективным для выявления незначительных изменений в коде, которые могли приводить к некорректным решениям задач. Например, такие изменения, как неправильный знак в формуле или неверное использование математической операции, не отражались в структуре дерева AST и, соответственно, не влияли на оценку сходства. Это создавало проблему: коды, логически неверные, могли быть оценены как высоко схожие с референсным.

* 1. **Генерация и использование тестов**
     1. **Тесты stdin/stdout**

Для решения проблемы из пункта 1.3.3. было принято решение дополнить анализ запуском кода с использованием тестов на основе стандартных потоков ввода и вывода (stdin и stdout). Этот подход позволил проверить фактическую корректность выполнения программы.

Для получения значений на ввод и вывод был написан инструмент для автоматической генерации тестовых данных и тестирования референсного кода с учётом заданных условий. На вход можно задать параметры тестов, включая количество переменных, их типы (целые или вещественные числа), диапазоны значений, знаки, индивидуальные ограничения для каждой переменной, глобальные ограничения на все переменные и отношения между ними. Например, можно задать такие ограничения: каждая переменная должна быть положительным числом с количеством цифр от 3 до 5; сумма всех переменных должна быть меньше 100; отношение между переменными может выражаться как x[0]≥x[1]x[0]. Генерация данных осуществляется случайным образом, при этом гарантируется соответствие всем заданным условиям. Референсный код тестируется с этими данными, входные значения подставляются вместо вызовов input(), а вывод программы перехватывается с помощью переопределения функции print(). В результате инструмент возвращает список данных в виде:

[

([вход1, … , входN], [вывод1, …,выводK]),

([вход1, … , входN], [вывод1, …,выводK]),

...] .

Количество входных и выходных данных может варьироваться в зависимости от специфики задания. Этот подход проверяет логическую корректность выполнения программы.

При каждом запуске проверки кода формируется 60 уникальных тестовых заданий, что исключает возможность “подстроить” решение под конкретные тестовые данные.

* + 1. **Мутационное тестирование кода (MBFL)**

MBFL (Mutation-Based Fault Localization) — это метод тестирования программного обеспечения, который использует мутационное тестирование для поиска ошибок и определения их местоположения в исходном коде. В отличие от традиционных подходов, MBFL направлен не только на проверку корректности тестов, но и на выявление конкретных строк кода, содержащих ошибки. Основная идея заключается в создании мутантов — изменённых версий кода, которые тестируются на заданных тестовых данных. Если выполнение мутанта приводит к ошибке, это указывает на возможную проблему в строке, где произошло изменение. [1]

Функция mutation\_based\_fault\_localization реализует этот подход, принимая в качестве параметров код студента, набор тестовых случаев с входными данными и ожидаемыми результатами, а также необязательное эталонное решение. В процессе работы функция создаёт мутанты, внося изменения в строки кода студента. Эти изменения охватывают арифметические операции, логические выражения, числовые значения, переменные, вызовы функций, условия, циклы и операторы возврата. Примеры таких мутаций включают замену математических операторов (+, -, \*, /), изменение операторов сравнения (>, <, ==, !=), модификацию числовых значений (например, x + 1 или -x), замену логических операторов (and, or, not) и т.д.

Если передано эталонное решение, функция выполняет посимвольное сравнение кода студента с эталонным кодом с использованием алгоритма SequenceMatcher. Строки, схожесть которых с эталоном превышает 80%, добавляются в список возможных мутаций для тестирования. Это позволяет учитывать заранее известные корректные решения.

После создания всех возможных мутантов функция выполняет тестирование, запуская каждую версию кода на заданных тестовых данных. Ввод и вывод программы перенаправляются через sys.stdin и sys.stdout, чтобы имитировать работу с консольным вводом и выводом. Исполняемый код выполняется с помощью функции exec(), а результаты сравниваются с ожидаемыми.

Если мутант выдаёт корректный результат, он считается «убитым» (PASS), что указывает на эффективность тестов. Если вывод отличается от ожидаемого или происходит ошибка выполнения, мутант «выживает» (FAIL/ERROR). Все результаты тестирования сохраняются в список с указанием номера строки, исходного кода, мутированной строки и статуса выполнения.

Если функция обнаруживает, что мутант исправляет ошибку и проходит все тесты, то ошибка может находиться в этой строке кода или в зависимых от неё строках. В таком случае выводится рекомендация о замене строки, что помогает локализовать ошибку и ускорить процесс отладки.

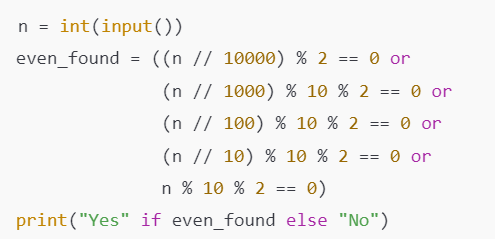
Количество мутаций для каждого кода можно рассчитать по формуле:

(1.6)

где:

* **M** — общее количество мутаций.
* **N** — количество строк кода.
* **Ai​** — количество мутаций арифм. операторов в строке i.
* **Li​** — количество мутаций логических операторов в строке i.
* **Ci​** — количество числовых мутаций в строке i.
* **Fi​** — количество мутаций функций в строке i.
* **Vi​** — количество мутаций переменных в строке i.
* **Ui​** — количество мутаций условий в строке i.
* **Ri​** — количество мутаций возвратов значений в строке i

Для условного кода



M=(2)+(20+16+30+10+50)+(6) M=2+126+6=134

M = 2 + 126 + 6 = 134M=2+126+6=134

Максимальное количество мутаций для данного кода = 134

# **Глава 2. Практическая часть проекта**

## **2.1. Принципы и детали реализации проекта**

### **2.1.1. Этапы алгоритма проверки кода студента**

1. Отправка кода студента на проверку (процесс начинается с отправки кода для анализа)

2. Прохождение 60 (кол-во тестов по умолчанию) случайно сгенерированных тестов.

3.Код проверяется на корректность с помощью автоматических тестов. Далее возможны два сценария:

1. Если код проходит все тесты, то выполняется анализ соответствия структуры через сравнение AST (Abstract Syntax Tree), чтобы проверить, соответствует ли структура кода заданным требованиям.
2. Если код не проходит тесты, то проводится мутационное тестирование для локализации возможной ошибки, чтобы определить конкретное место в коде, которое вызывает проблему.

На основе результатов тестирования и анализа формируется обратная связь с замечаниями и рекомендациями. Процесс проверки кода завершается после предоставления комментариев. Схема алгоритма представлена на рисунке 5.

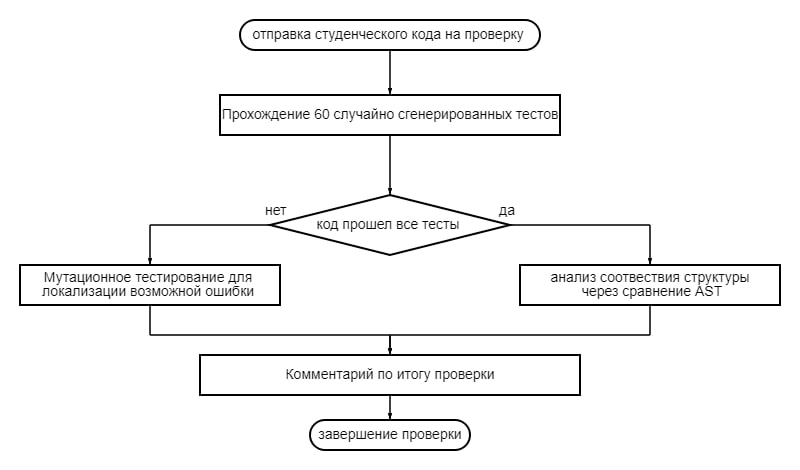


Рис.5 Схема алгоритма проверки кода

**2.1.2. Описание работы сервиса**

Пользователь отправляет запрос через интерфейс (Frontend) для получения задания или проверки кода. Frontend взаимодействует с backend (на базе Django), который состоит из двух модулей: модуля выдачи задания и модуля проверки решения. Модуль выдачи задач выбирает данные из базы данных (PostgreSQL) с использованием Django ORM и отправляет их на frontend. Модуль проверки решений выполняет тестирование и структурный анализ кода, переданного пользователем, и возвращает результат на frontend. В итоге пользователь получает задание и результаты проверки в интерфейсе сервиса. Архитектура сервиса представлена в приложение 1.

### **2.1.3. Функциональные возможности проекта**.

* Добавление заданий через админ-панель Django
* Генерация тестов с использованием критериев входного значения через инструмент генерации
* Проверка кодов студентов с использованием тестов и сравнения по AST+ анализ MBFL
* Выдача комментария по итогу проверки

## **2.2. Пользовательский (или иной внешний) интерфейс проекта**.

### **2.2.1. Интерфейс добавления заданий**

Добавление, удаление и редактирование заданий осуществляется через встроенную админ-панель (Рисунок 6) Django (на этапе MVP это делается вручную).



Рис.6 Добавление задания через Админ-панель

### **2.2.2. Основной интерфейс на этапе MVP**

На главной панели сервиса отображается список практических работ, каждая из которых представляет отдельную тему:

Только первая практическая работа работа активна и позволяет перейти к заданиям, в то время как остальные практические работы недоступны (кнопки для перехода к заданиям серые и неактивны).

Интерфейс выполнен в минималистичном стиле с синей верхней панелью, на которой размещено название сервиса CheckUs и логотип ТюмГУ (Рисунок 7)

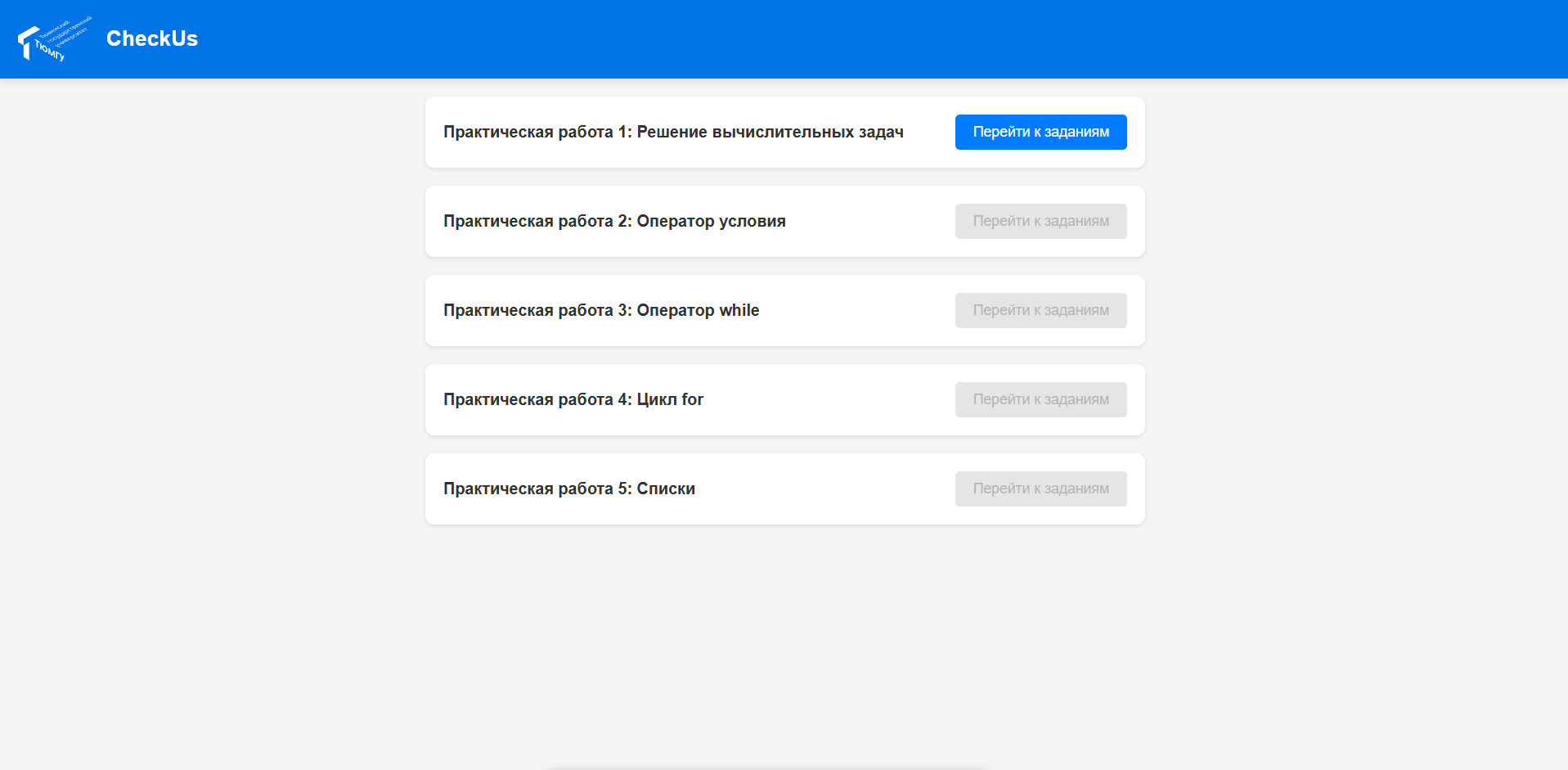


Рис.7 Главная панель

Интерфейс заданий (Рисунок 8), открывающийся после выбора практической работы. В тексте задания указана сама постановка задания, а также критерии к записи кода (Тип входных данных и правила для вывода).

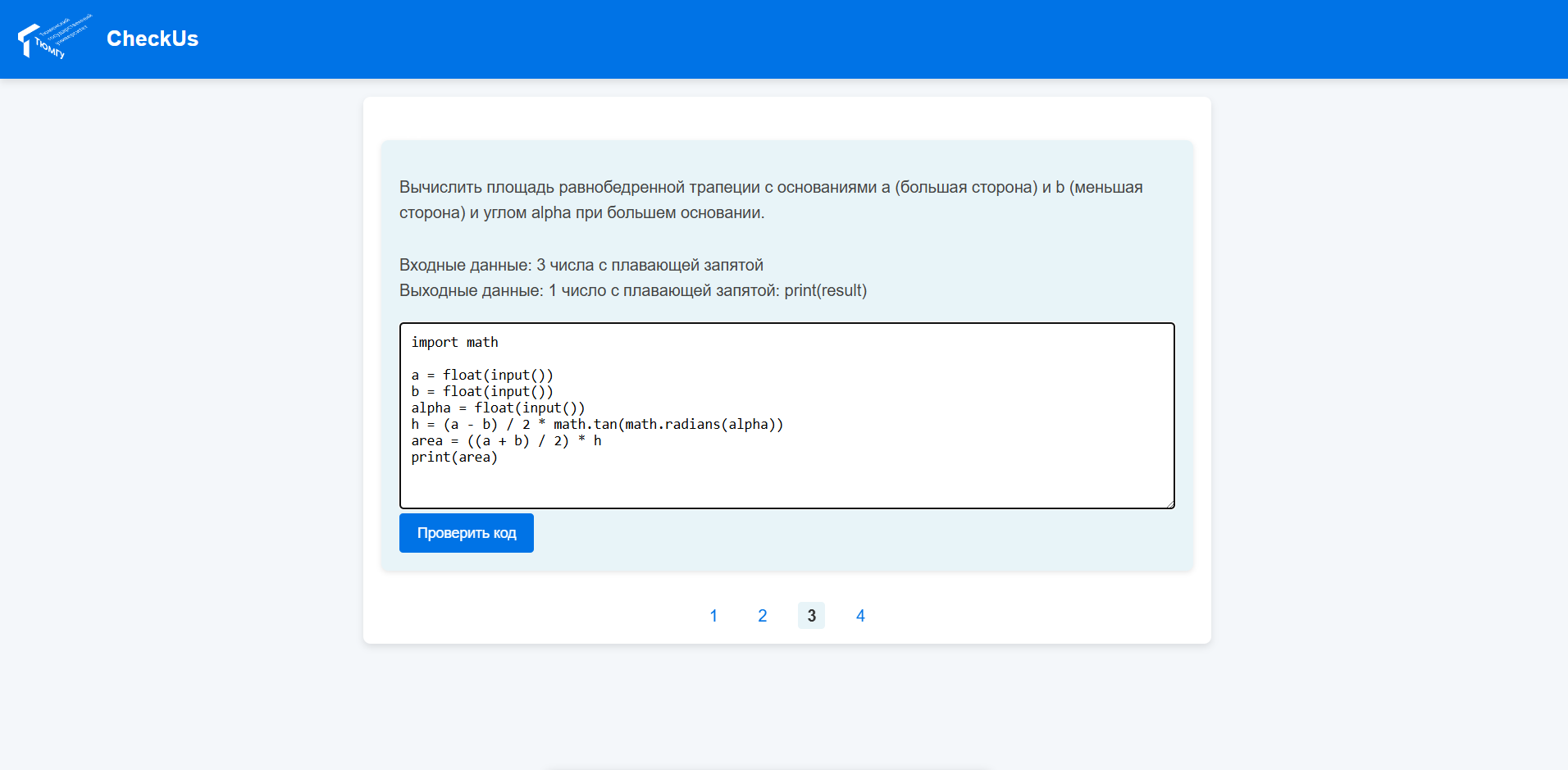


Рис.8 Интерфейс с заданием

### **2.2.3. Обратная связь о проверке кода**

После выбора задания студент может ввести в форму код для решения указанной задачи. Возможные исходы:

1. Студент написал правильный код в соответствии с поставленной задачей. В таком случае выводится следующего вида (Рисунок 9-10):

* Сообщение о том, что код прошел все тесты
* Сообщение о том, что код соответствует по структуре поставленной задаче
* 5 тестов из 60 с детализацией, какие входные данные были использованы, какие ожидались и фактически получение.

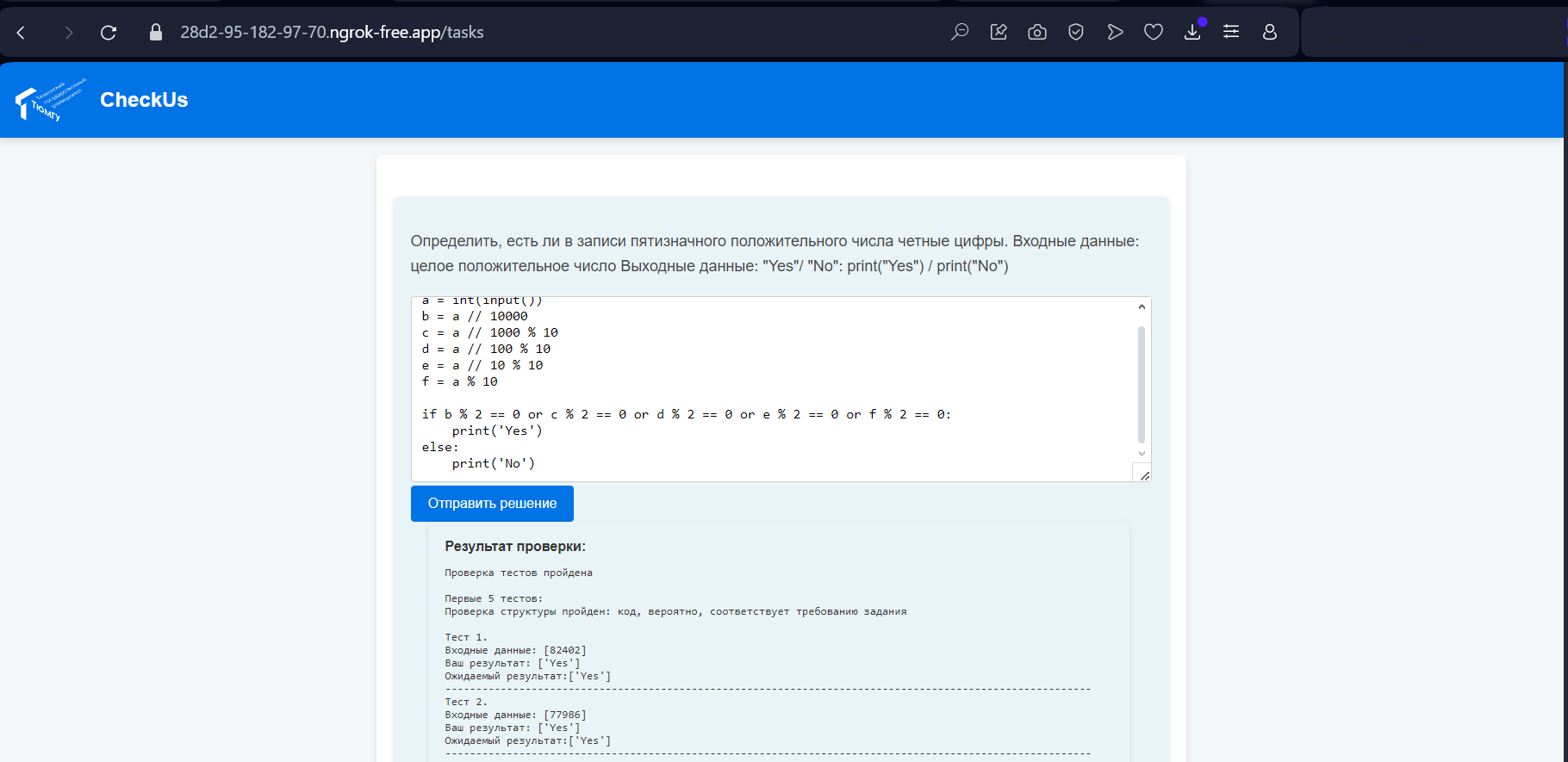


Рис.9 Пример результата проверки

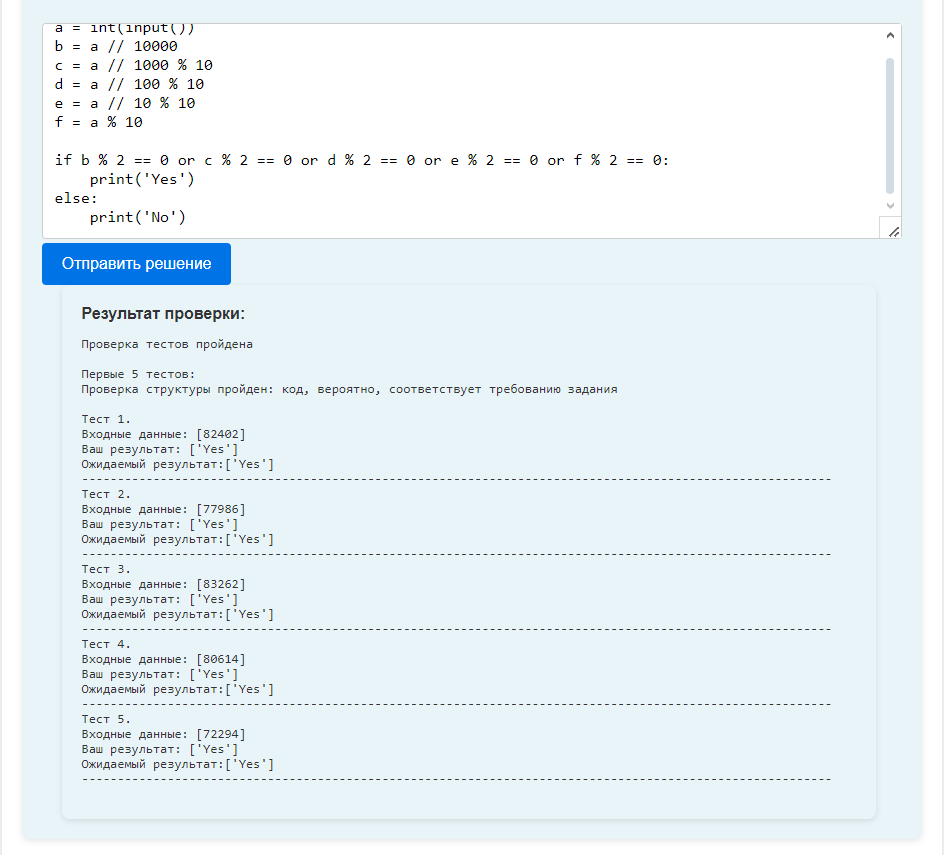


Рис.10 Текст комментария проверки ,все правильно

2. Студент написал правильный код, однако его структура отличается от структуры референсного кода (вероятно, код не соответствует постановке задачи). В таком случае выводится следующего вида (Рисунок 11-12):

* Сообщение о том, что код прошел все тесты
* Сообщение о том, что код не соответствует по структуре поставленной задаче
* 5 тестов из 60 с детализацией, какие входные данные были использованы, какие ожидались и фактически получение.

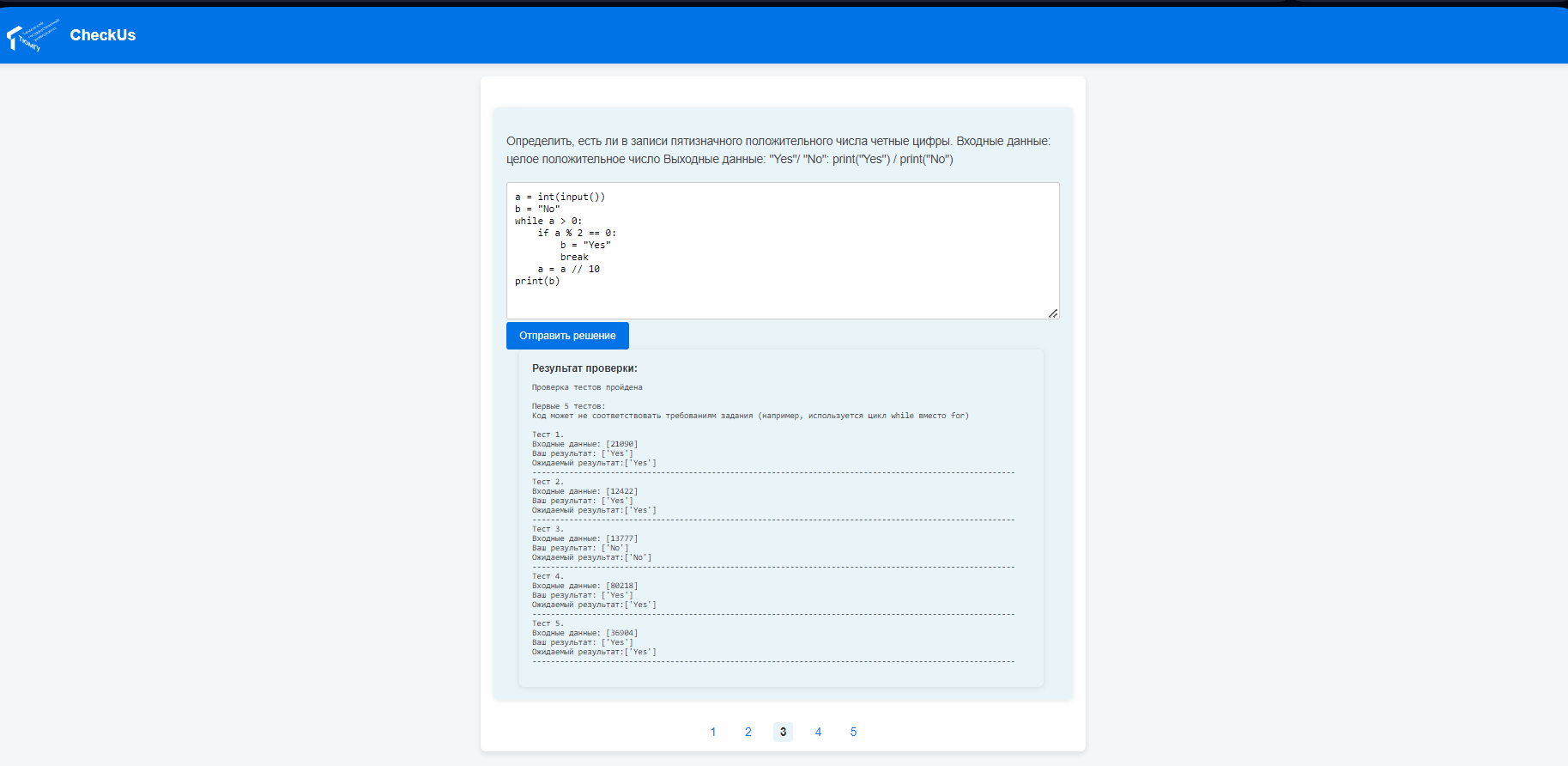


Рис.11 Пример проверки №2

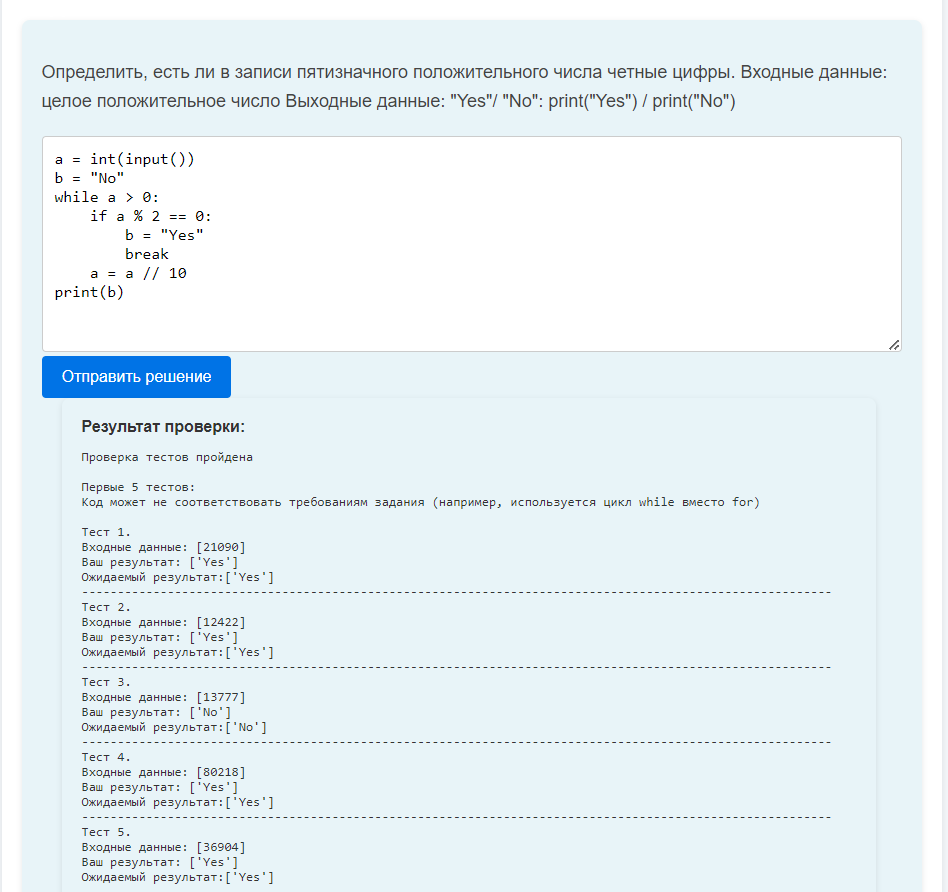


Рис.12 Текст комментария проверки, пройдены тесты, но не соответствие требованиям по структуре задания

3. Студент допустил небольшую ошибку в коде, а алгоритм мутаций смог её распознать. В таком случае выводится следующего вида (Рисунок 13):

* Сообщение о том, что код не прошёл тесты
* Возможная замена в коде для исправления ошибки
* 5 тестов из 60 с детализацией, какие входные данные были использованы, какие ожидались и фактически получение. Из них не менее одного – проваленные тесты

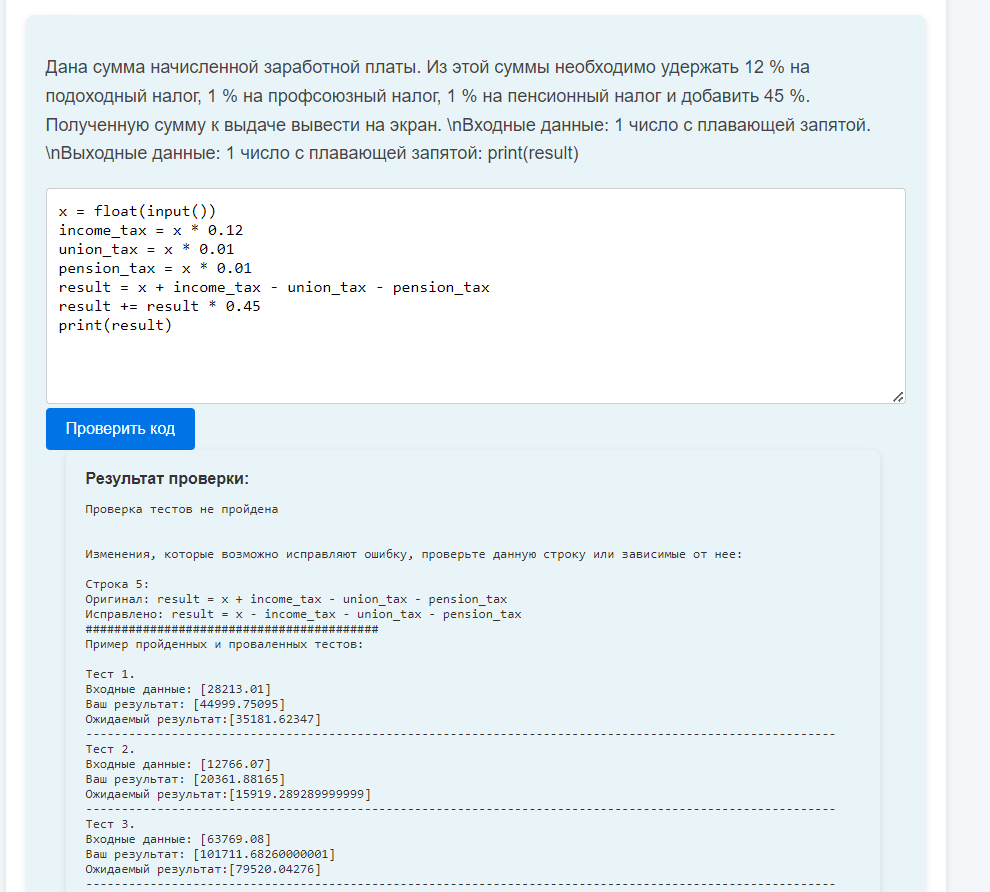


Рис.13 Алгоритм мутаций нашел ошибку и вывел ее в комментарии

4. Студент написал неправильный код, сервис не смог помочь найти ошибку. В таком случае выводится следующего вида (Рисунок 14):

* Сообщение о том, что код не прошёл тесты
* Сообщение о том, найти возможное исправление не удалось
* 5 тестов из 60 с детализацией, какие входные данные были использованы, какие ожидались и фактически получение. Из них не менее одного – проваленные тесты

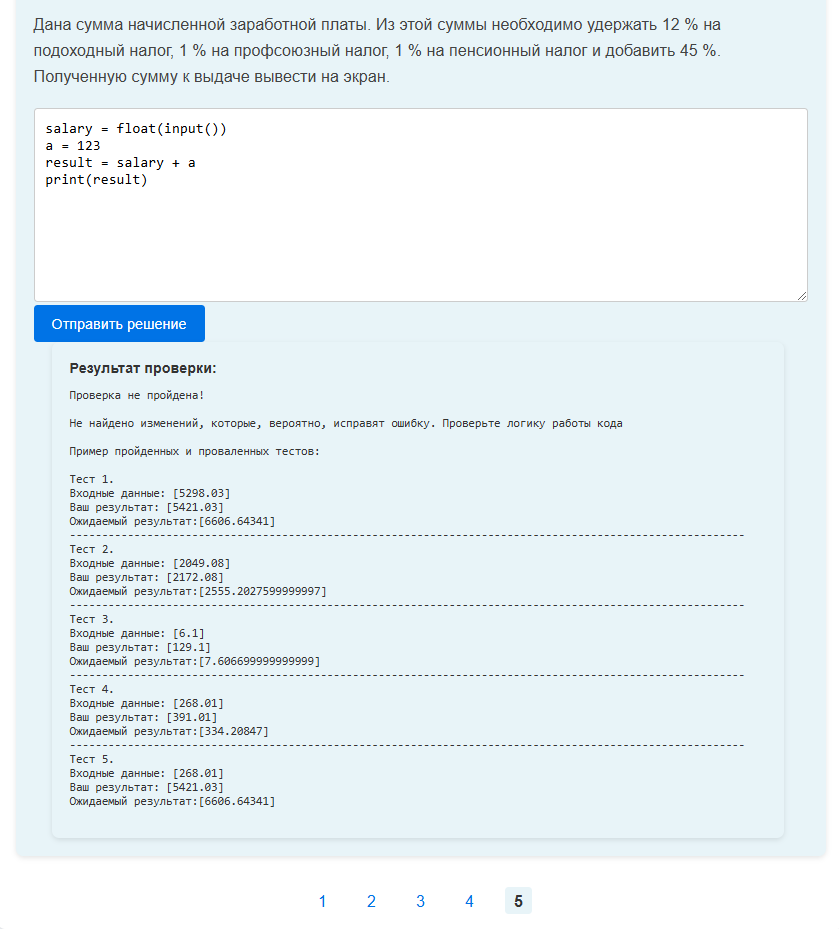


Рис.14 Алгоритм не смог найти ошибку рекомендованную к исправлению

## **2.3. Организация тестирования проекта**

### **2.3.1. Цели тестирования**

1. Проверка интерфейса (понятность, удобность)​
2. Проверка работоспособности сервиса (выявление возможных ошибок)​
3. Проверка качества обратной связи (насколько полезна проверка от сервиса

### **2.3.2. Исполнители**

1. Эксперты (разработчики проекта)
2. 6 студентов 1 курса направления МОиАИС
3. 2 студента старших курсов не IT-направлений, но изучающих Python в университете

### **2.3.3. Результаты тестирования на основе обратной связи от пользователей**

1. Для улучшения пользовательского опыта нужно будет поработать над интерфейсом, формой ввода кода и прочим (хороший пример – это Codewars)  
2. Обратная связь по проверке кода, по мнению пользователей, адекватная  
3. Пользователи отмечают удобство написания и отправки практических работ в сервисе  
4. Преподаватель даёт более качественную обратную связь, но в том случае, когда нет возможности быстро её получить, сервис может помочь

# **Заключение**

В ходе работы был разработан MVP сервиса, который прошёл тестирование на реальных пользователях, что подтвердило его минимальную работоспособность и актуальность для целевой аудитории. Были определены планы для дальнейшего развития проекта, направленные на его улучшение и расширение функциональности. Также был собран датасет программных кодов студентов, который позволит усовершенствовать алгоритмы анализа и проверки в будущем. Эти результаты создают основу для дальнейшего развития проекта и его интеграции в образовательные процессы.

# **Список литературы**

1. Rezaalipour M., Furia C. A. An Empirical Study of Fault Localization in Python Programs., 2024, arXiv:2305.19834v3. URL: https://arxiv.org/abs/2305.19834v3 (дата обращения: 05.12.2024).

2. Song Y., Lothritz, C., Tang, D., Bissyandé, T. F., Klein, J. Revisiting Code Similarity Evaluation with Abstract Syntax Tree Edit Distance., 2024, arXiv:2404.08817v2. URL: https://arxiv.org/abs/2404.08817v2 (дата обращения: 14.11.2024).

3. Doe J., Smith A. Empirical Analysis of Abstract Syntax Tree Metrics for Code Similarity. \*Proceedings of the 19th IEEE International Working Conference on Source Code Analysis and Manipulation\*, 2019, pp. 45-56. URL: https://doi.org/10.1109/SCAM.2019.00021 (дата обращения: 28.10.2024).

# **Приложения**

Приложение 1.

Архитектура сервиса

