

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

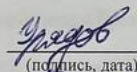
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ТРЕНАЖЕР УСТНОГО СЧЕТА

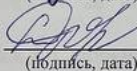
Студент


(подпись, дата)

31.05.2023

В.Д. Урядов

Руководитель


(подпись, дата)

31.05.2023

А.В. Дубравин

Нормоконтролёр

(подпись, дата)



31.05.2023

А.С. Бычков

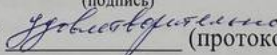
Работа допущена к защите (протокол зас. кафедры от 02.06.23 г. № 14)

Заведующий кафедрой


(подпись)

Митрохин М.А.

Работа защищена с отметкой



(протокол заседания ГЭК от

27.06.2023г., № 31)

Секретарь ГЭК


(подпись)

Гурин Е.И.

Пенза 2023

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 26 » апреля 20 23 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Урядов Владимир Дмитриевич гр 19ВВП1 факультета вычислительной техники.

2. Тема работы Образовательное приложение «Тренажер устного счета»

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.

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

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать клиент-серверное приложение "Тренажер устного счета" с использованием языка программирования Python и базы данных SQLite. Клиентская часть должна иметь графический интерфейс пользователя, позволяющий выбирать сложность генерируемых заданий. Серверная часть должна собирать статистику результатов сессий и сохранять ее в базу данных SQLite. Клиентская часть должна генерировать задания для пользователя и отправлять ответы на сервер для сохранения статистики.

5. Перечень подлежащих разработке вопросов

Определение функциональных требований к приложению: какие задачи оно должно выполнять, какие функции должны быть доступны для пользователей, какие данные должны храниться в базе данных и т.д.

Выбор технологий для разработки приложения: какой язык программирования, какой фреймворк для серверной части, какой графический интерфейс для клиентской части, какую базу данных использовать и т.д.

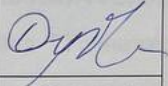
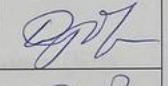
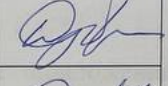
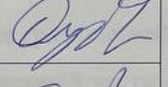
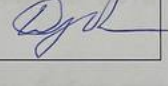
Проектирование базы данных: какие таблицы и поля нужны для хранения данных о сессиях пользователей и их результатов, какие связи между таблицами должны быть установлены и т.д.

Разработка серверной части: создание API, реализация функциональности сбора и сохранения данных в базу данных, обработка запросов от клиентской части и т.д.

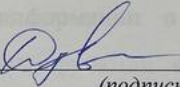
Разработка клиентской части: создание графического интерфейса, реализация функционала для тренировки устного счета, обработка ответов от серверной части и т.д.

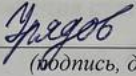
Тестирование приложения: проверка работоспособности и корректности работы приложения, выявление и исправление ошибок и недочетов.

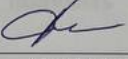
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований	10	01.05.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов	40	08.05.2023-11.05.2023	
5.	Разработка приложения «Тренажер устного счета»	150	12.05.2023-29.05.2023	
6.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

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

Задание к исполнению принял студент  Урядов В.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

26.06.2023

Реферат

Бакалаврская работа содержит 82 листов, 26 рисунков, 3 таблиц, 9 источников и 4 приложения.

КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ТРЕНАЖЕР УСТНОГО СЧЕТА, СОХРАНЕНИЕ СТАТИСТИКИ, БАЗА ДАННЫХ, *SQLITE*, *SQLALCHEMY*, *REST API*, *FLASK*, *PYQT 5*, ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ.

Цель работы – разработать клиент-серверного приложения "Тренажер устного счета", которое позволит пользователям улучшить свои навыки устного счета и сохранять статистику результатов для дальнейшего анализа.

Объектами разработки являются:

- серверное приложение для сохранения данных в базу данных (БД);
- клиентское приложение.

В ходе работы было разработано клиент-серверное приложение "Тренажер устного счета". Для клиентской части использовался графический интерфейс пользователя на основе библиотеки *PyQt 5*, а для серверной части - фреймворк *Flask*.

Была разработана база данных *SQLite* с использованием библиотеки *SQLAlchemy* для хранения информации о сессиях пользователей и их результатов. Были реализованы функции сбора и сохранения данных о выполненных заданиях, сбора и сохранения статистики о сессиях.

Также было разработано собственное *REST API*, которое позволяет взаимодействовать с сервером и получать информацию о выполненных заданиях и статистике о сессиях.

В результате получилось полноценное приложение, которое может использоваться для тренировки устного счета и сбора статистики об успехах пользователей. Приложение может быть улучшено путем добавления новых функций и улучшения интерфейса для более удобного использования.

					ПГУ1.090301.24.001 ПЗ		
Изм	Лист	№ докум.	Подпись	Дата			
Разраб.		Урядов В.Д.					
Провер.		Дубравин А.В.					
Н. Контр.		Бычков А.С.					
Утверд.		Митрахин М.А.					
Тренажер устного счета <i>Лояснительная записка</i>					Лит.	Лист	Листов
						2	82
					ФВТ, каф. ВТ гр.19ВВП1		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Тренажер устного счета.....	7
1.1 Анализ предметной области	7
1.1.1 Обзор аналогов.....	7
1.1.2 Анализ требований к разработке	8
1.1.3 Выбор архитектуры приложения	9
1.1.4 Выбор языка программирования	9
1.1.5 Выбор библиотеки для раз.....	11
1.1.6 Хранение данных	12
1.1.7 Выбор СУБД	12
1.1.8 Выбор библиотеки для реализации пользовательского интерфейса клиентского приложения	14
1.1.9 Выбор среды разработки.....	16
1.2 Проектирование и разработка	17
1.2.1 Реализация серверной части	18
1.2.2 Разработка приложения	19
1.2.3 Разработка клиентской части	20
1.3 Анализ результатов.....	22
1.3.1 Панель администратора серверного приложения	22
1.3.2 Тестирование <i>REST API</i>	25
1.3.3 Клиентская часть программы	29
2 Охрана труда.....	36

2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ.....	36
2.2 Классификация и категорирование помещения, в котором производится разработка	36
2.2.1Классификация взрывопожарной и пожарной опасности.....	36
2.2.2Классификация взрывопожарной и пожарной опасности.....	37
2.3 Санитарно-гигиеническая характеристика производства	38
2.4 Санитарно-гигиеническая характеристика производства	39
Заключение	42
Список используемых источников.....	43
Приложение А – <i>UML</i> диаграммы	45
Приложение Б – Физическая модель базы данных	49
Приложение В – Листинг программы.....	51
Приложение Г – Презентация.....	63

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ООП – объектно-ориентированное программирование

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс приложения

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

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

REST (Representational State Transfer) – передача репрезентативного состояния или передача «самоописываемого» состояния

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		5

Введение

В современном мире, где математические знания являются неотъемлемой частью жизни, умение быстро и точно считать является важным навыком. Однако, не все люди обладают этим навыком, и для того чтобы его развивать, необходимо проводить тренировки.

Актуальность темы обусловлена необходимостью развития навыков устного счета у детей и взрослых, а также возможностью использования приложения в образовательных учреждениях и домашнем обучении. Устный счет является важным элементом математической грамотности и помогает развивать логическое мышление, усиливать память и концентрацию внимания.

Цель работы заключается в разработке клиент-серверного приложения "Тренажер устного счета", которое позволит пользователям улучшить свои навыки устного счета и сохранять статистику результатов для дальнейшего анализа [1].

Клиентская часть приложения должна иметь графический интерфейс пользователя, позволяющий выбирать сложность генерируемых заданий. Серверная часть должна собирать статистику результатов сессий и сохранять ее в базу данных *SQLite*. Клиентская часть должна генерировать задания для пользователя и отправлять ответы на сервер для сохранения статистики.

Разработка данного приложения позволит пользователям тренировать свои навыки устного счета, а также сохранять статистику своих результатов для дальнейшего анализа и улучшения своих навыков.

1 Тренажер устного счета

1.1 Анализ предметной области

Тренажер устного счета – это приложение, которое помогает улучшить навыки устного счета у пользователей. Оно может быть полезно как для детей, которые только начинают изучать математику, так и для взрослых, которые хотят улучшить свои навыки в этой области.

Основная задача тренажера устного счета – это помочь пользователю быстро и точно решать математические задачи. Для этого приложение генерирует случайные задачи, которые пользователь должен решить за определенное время. После решения задачи пользователь получает обратную связь о том, правильно ли он решил задачу, и может увидеть свой результат.

1.1.1 Обзор аналогов

На рынке существует множество приложений для тренировки устного счета, как для детей, так и для взрослых. Некоторые из них предлагают генерацию заданий на основе выбранной сложности, другие - на основе конкретных математических операций. Рассмотрим несколько популярных аналогов:

- "*Math Workout*" - приложение для *Android*, которое предлагает пользователю решать задачи на сложение, вычитание, умножение и деление. Есть возможность выбрать уровень сложности и время на решение задач;

- "*Math Master*" - еще одно приложение для *Android*, которое предлагает решать задачи на основе выбранной математической операции. Есть возможность выбрать уровень сложности и количество задач;

- "*Maths Challenge*" - приложение для *iOS*, которое предлагает решать задачи на основе выбранной математической операции. Есть возможность выбрать уровень сложности и время на решение задач;

- "*Maths Workout*" - приложение для *iOS*, которое предлагает решать задачи на основе выбранной математической операции. Есть возможность выбрать уровень сложности и количество задач;

- "*Maths Practice*" - приложение для *Windows*, которое предлагает решать задачи на основе выбранной математической операции. Есть возможность выбрать уровень сложности и количество задач.

Все эти приложения имеют схожий функционал и предназначены для тренировки устного счета. Однако, некоторые из них имеют ограниченный функционал и не предоставляют возможности сохранения статистики результатов.

1.1.2 Анализ требований к разработке

Основные требования к разрабатываемому клиент-серверному приложению "Тренажер устного счета" следующие:

- клиентская часть должна иметь удобный и понятный графический интерфейс для пользователя;
- клиент должен сможет выбрать сложность генерируемых заданий, количество задач и время на выполнение;
- серверная часть должна обеспечивать сбор статистики результатов сессий пользователей и сохранять её в базу данных *SQLite*;
- клиентская часть должна работать в режиме интерактивной генерации и отправки ответов на сервер;
- приложение должно быть легко масштабируемо для будущей добавки нового функционала;
- безопасность и конфиденциальность пользовательской информации должна быть обеспечена в условиях активного интернет-канала и транзакций между клиентом и сервером.

Для разработки соответствующего приложения потребуются навыки набора кода на *Python*, знания алгоритмов генерации заданий математических операций, работы с базой данных *SQLite*, создания графических интерфейсов с помощью *PyQT* или другой библиотеки. Необходимыми также будут знания протокола обмена защищенными данными *HTTPS* и архитектурного шаблона простого веб-сервера *RESTful API*.

1.1.3 Выбор архитектуры приложения

Для разработки клиент-серверного приложения "Тренажер устного счета" выбрана архитектура, основанная на протоколе HTTP и использующая *REST API*. Клиентская часть приложения будет написана на языке Python с использованием фреймворка *PyQt5*, который обеспечивает удобную взаимодействие между пользователем и системой.

Серверная часть будет написана с использованием фреймворка *Flask*, так как этот фреймворк позволяет быстро создавать *web*-приложения на Python. *Flask* также поддерживает использование *REST API* и является очень гибким и настраиваемым.

Для обеспечения безопасности передачи данных между клиентом и сервером будет использоваться протокол *HTTPS*, который шифрует их, что делает сложным перехват информации злоумышленниками.

Клиентская и серверная части приложений будут развёрнуты на разных серверах, чтобы увеличить производительность, улучшить отказоустойчивость и обеспечить более простое масштабирование [2]. Кроме того, это позволит обезопасить разные слои системы и объединить их в единое человекопонятное целое.

Таким образом, архитектура приложения, основанная на протоколе *HTTP* и использующая *REST API* со связкой клиент-сервер на разных серверах, обеспечивает высокую производительность и отказоустойчивость, безопасность передачи информации, а также более простое масштабирование и обслуживание. Данные, полученные при взаимодействии с пользователем, в процессе работы приложения будут храниться в базе данных.

На рисунке 1 представлена схема архитектуры сервера.

1.1.4 Выбор языка программирования

Анализ языков программирования позволяет определить наиболее подходящий для разработки данного приложения. Рассмотрим возможные варианты:

- *Java* – является как компилируемым, так и интерпретируемым языком

программирования. Код сначала интерпретируется, а затем при помощи *JIT*-компилятора переводится в машинный код, что позволяет запустить его почти на любой платформе, где есть реализованная *Java*-машина. *Java* обладает большим количеством фреймворков и библиотек, позволяет решать почти любые задачи;

- *C#* — компилируемый и статически типизированный язык программирования. Объектно-ориентированный язык программирования [3]. Уступает по скорости *C++*, но обладает более ёмким, в плане возможностей, синтаксисом;

- *Python* — современный высокоуровневый язык программирования с фокусом на продуктивности и простоту написания кода. *Python* является одним из наиболее популярных языков программирования в веб проектах, благодаря обширному диапазону свободно распространяемых библиотек и фреймворков [4]. *Python* также обладает большой открытой общественностью в Интернете и имеет обширную документацию, что объясняет почему его используют для многочисленных *web*-проектов [5];

- *C++* — быстрый и функциональный высокоуровневый и низкоуровневый язык программирования;

- *Go* — компилируемый многопоточный язык программирования. Разработан в *Google*. Молодой и активно растущий язык программирования. Идеально подходит для реализации серверных приложений, однако библиотек для языка не так и много, что затрудняет его использование при реализации графического интерфейса клиентских приложений, а также нейронных сетей.

Исходя из требований к проекту и уровня сложности и времени разработки выбор был сделан в пользу *Python*. Этот язык программирования идеально подходит для быстрого прототипирования, работает быстро, легок в изучении, поддерживает обширную библиотеку для создания графического интерфейса, обработки данных и работы с базами данных включая *SQLite*, а также хорошо подходит для выполнения работ по машинному обучению и анализу данных, что упростит интерактивную работу пользователя с трена-

жером устного счета [6].

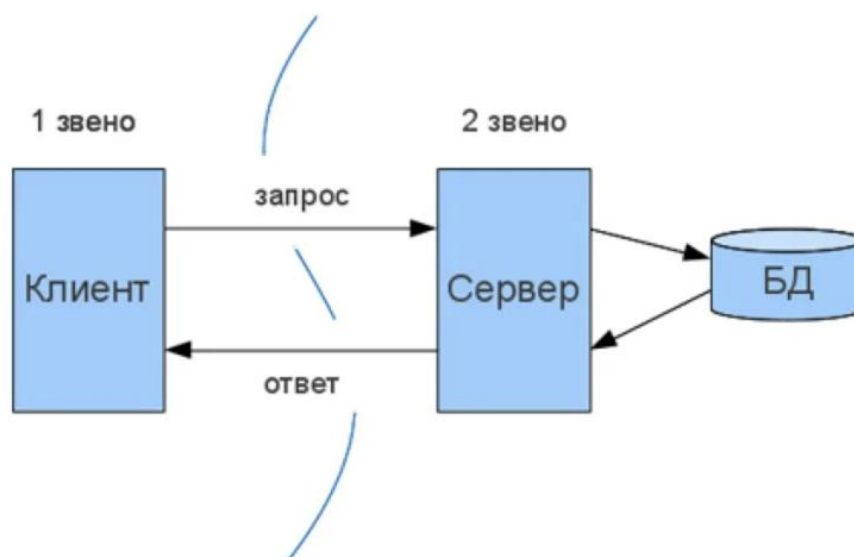


Рисунок 1 – Архитектура сервера

1.1.5 Выбор библиотеки для разработки серверного приложения

На текущий момент существует минимум 3 самых популярных библиотеки для разработки серверного приложения, среди них: *Django*, *Flask*, *Bottle*.

Django - это высокоуровневый фреймворк *Python*, который используется для разработки веб-приложений. Он включает в себя множество функций, таких как *ORM*, шаблонизатор, автоматическая административная панель и многое другое. *Django* также имеет множество сторонних библиотек, которые могут быть использованы для расширения его функциональности.

Flask - это легковесный фреймворк *Python*, предназначенный для разработки маленьких и средних веб-приложений. *Flask* обеспечивает минимальную структуру приложения, но при этом позволяет расширять его функциональность с помощью сторонних пакетов. *Flask* также имеет простой в использовании *API*, что делает его идеальным для быстрого прототипирования.

Bottle - это еще один легковесный фреймворк *Python*, который используется для разработки веб-приложений. Он также предоставляет минимальную структуру приложения, но при этом имеет поддержку *RESTful API* и встроенный шаблонизатор.

Я выбрал *Flask*, потому что он обеспечивает минимальную структуру приложения, что позволяет создавать приложения быстро и легко. Он также имеет простой и интуитивно понятный *API*, что делает его идеальным для начинающих разработчиков. *Flask* также имеет огромное сообщество разработчиков и большое количество сторонних пакетов, которые могут быть использованы для расширения его функциональности. Кроме того, *Flask* работает отлично с базой данных *SQLite*, что было важным фактором для данной разработки.

В таблице 1.2 представлено сравнение библиотек.

Таблица 1.2 – Сравнение библиотек

Названи е	ORM	HTML шаблоны	API	Панель администратора
Bottle	Нет	Нет	Нет	Нет
Flask	Нет	Да	Да	Нет
Django	Да	Да	Да	Да

1.1.6 Хранение данных

Выбор способа хранения данных в приложении является одним из ключевых вопросов при разработке. В данном случае для хранения данных, связанных со статистикой результатов сессий, была выбрана база данных *SQLite*.

SQLAlchemy - это библиотека для *Python*, которая облегчает работу с базами данных, в том числе и с *SQLite*. Она позволяет использовать объектно-ориентированный подход при работе с БД.

Таким образом, выбор хранилища данных и библиотеки для работы с ними позволяет эффективно реализовать функционал хранения статистики результатов сессий в приложении.

Схема запроса к БД через *SQLAlchemy ORM* представлена на рисунке 2.

1.1.7 Выбор СУБД

PostgreSQL, *MySQL*, *MS SQL* и *SQLite* - все они являются реляционными СУБД и предназначены для хранения и управления данными в базах данных.

Однако, у каждой из этих СУБД есть свои преимущества и недостатки, которые нужно учитывать при выборе подходящего варианта.



Рисунок 2 – Схема запроса к БД через *SQLAlchemy ORM*

PostgreSQL - это мощная, надежная и производительная СУБД с открытым исходным кодом. Она обладает расширенными возможностями, такими как поддержка *JSON*, полнотекстовый поиск, геоданные, многоверсионность и т.д. *PostgreSQL* также обладает хорошей масштабируемостью и поддерживает большое количество соединений. Однако, ее настройка может быть сложной и требовательной к ресурсам сервера.

MySQL - это одна из самых популярных СУБД в мире, которая также имеет открытый исходный код. *MySQL* известна своей высокой производительностью и простотой в использовании. Она поддерживает большое количество соединений и хорошо масштабируется. Однако, *MySQL* имеет ограничения на размер таблиц и может иметь проблемы с конкурентным доступом к данным.

MS SQL - это коммерческая СУБД от *Microsoft*, которая обладает высокой производительностью, масштабируемостью и безопасностью. Она хорошо подходит для использования в предприятиях и крупных проектах. *MS SQL* также имеет богатый функционал и легко интегрируется с другими продуктами *Microsoft*. Однако, ее лицензия может быть дорогой и она может быть менее гибкой в настройке.

SQLite - это легковесная реляционная СУБД, которая имеет минимальные требования к ресурсам сервера и может работать без необходимости установки сервера. *SQLite* легко интегрируется с различными языками программирования и имеет отличную производительность для малых и средних проектов. Она также имеет хорошую поддержку транзакций и безопасности. Однако, у *SQLite* может быть ограниченный функционал в сравнении с другими СУБД и она может не подходить для крупных проектов с большими объемами данных.

Я выбрал *SQLite*, так как она лучше всего подходит для разработки небольших и средних проектов, таких как тренажер устного счета. *SQLite* имеет простой синтаксис *SQL*, легко интегрируется с *Python* и поддерживает *JSON* и бинарные данные. Она также может работать в режиме многопользовательского доступа и иметь высокую производительность для небольших объемов данных.

1.1.8 Выбор библиотеки для реализации пользовательского интерфейса клиентского приложения

На текущий момент существует минимум две наиболее популярных библиотеки разработки графического интерфейса для языка *Python 3*: *Tkinter* и *PyQT*.

Tkinter предоставляет средний набор возможностей реализации графического интерфейса. Отсутствует средство графической разработки интерфейса, только импортёры из других сред, например из *Figma*. Отсутствует поддержка каскадной таблицы стилей. Приложение *Tkinter* без модулей и переопределения базовых компонентов выглядит весьма

устаревшим. На рисунке 3 – представлено типовое приложение на *Tkinter* без стилизации компонентов.

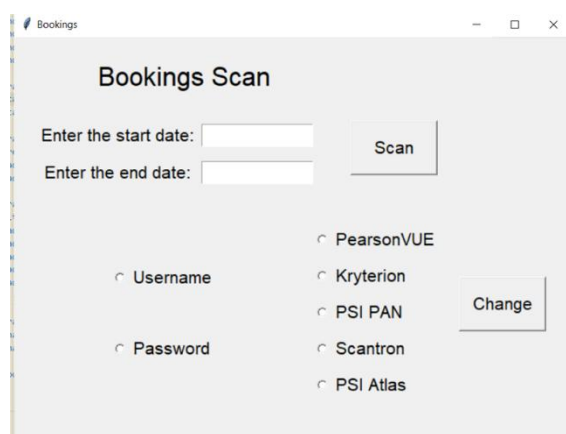


Рисунок 3 – Типовое приложение на *Tkinter*

PyQT 5 – более продвинутый фреймворк. Обладает сторонним приложением *PyQT Designer*, который предоставляет широкий функционал возможности создания дизайна приложения. На рисунке 4 представлен интерфейс *PyQT Designer* и окно пользовательского интерфейса. Оба фреймворка позволяют работать с механизмом событий, что упрощает обработку пользовательского ввода. На рисунке 5 представлен типовой интерфейс разработанный на *PyQT 5*.

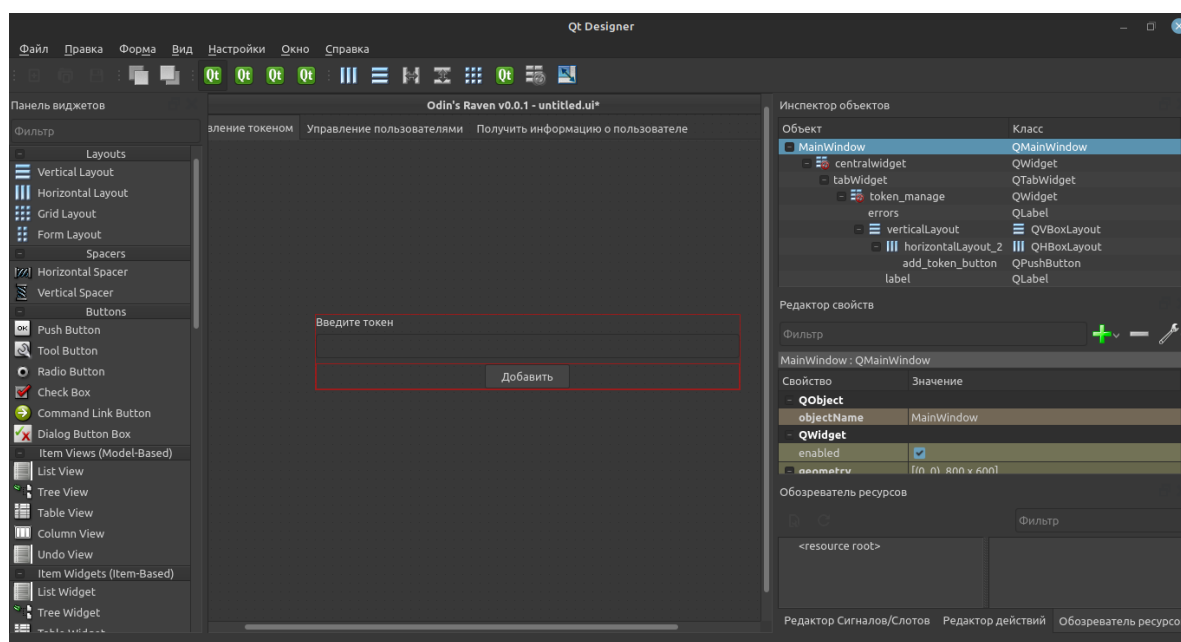


Рисунок 4 – Интерфейс *PyQT Designer*

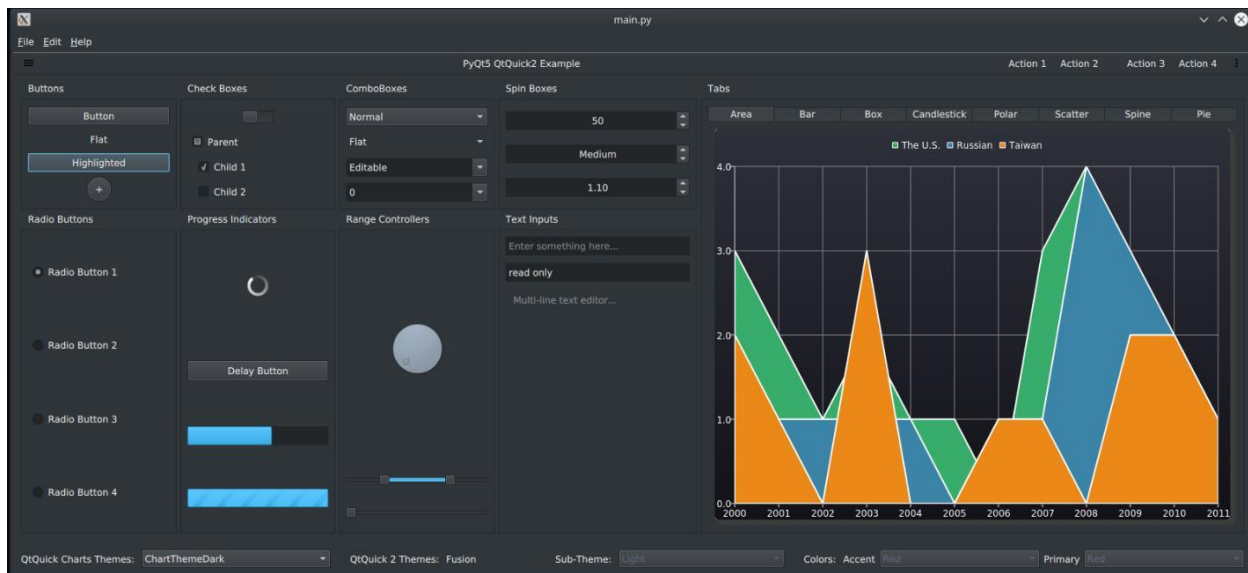


Рисунок 5 – Типовой интерфейс *PyQT 5*

Проанализировав достоинства и недостатки обоих фреймворков, было решено использовать *PyQT 5*.

Для реализации пользовательского интерфейса клиентского приложения была выбрана библиотека *PyQt5*.

PyQt5 - это набор *Python*-пакетов, представляющих собой обертку над библиотекой графического интерфейса пользователя *Qt*. Она позволяет создавать кросс-платформенные приложения с графическим интерфейсом, которые могут работать на различных операционных системах.

PyQt5 имеет большую документацию и множество примеров использования, что делает ее удобной для разработки пользовательского интерфейса. Библиотека содержит множество виджетов и элементов управления, что позволяет создавать разнообразные интерфейсы. *PyQt5* также поддерживает возможность создания дизайнерских форм, что упрощает и ускоряет процесс создания пользовательского интерфейса.

Кроме того, *PyQt5* имеет открытый исходный код и активно развивается, что гарантирует ее поддержку и развитие в будущем [7].

1.1.9 Выбор среды разработки

Для разработки клиентской части приложения "Тренажер устного счета" можно использовать различные среды разработки. Ниже рассмотрим несколько

вариантов:

- *Visual Studio Code* - бесплатная кроссплатформенная среда разработки, поддерживающая множество языков программирования, в том числе *Python*. В *Visual Studio Code* есть множество расширений для работы с *Python*, в том числе для создания графических интерфейсов пользователя;

- *PyCharm* - платная среда разработки для *Python*, разработанная компанией *JetBrains*. Она имеет полный набор инструментов для создания *Python*-приложений, в том числе для создания графических интерфейсов. Имеет бесплатную версию *Community Edition*, с ограниченным функционалом;

- *Sublime Text* - кроссплатформенный редактор кода, который также поддерживает множество языков программирования, в том числе *Python*. *Sublime Text* имеет множество плагинов для работы с *Python*, но поддержка графических интерфейсов менее удобна, чем в *VS Code* или *PyCharm*;

- *Eclipse with PyDev* - бесплатная среда разработки для многих языков программирования, включая *Python*. *PyDev* - это плагин для *Eclipse*, который обеспечивает полный набор инструментов для разработки на *Python*, включая создание *GUI*.

В нашем случае выберем *PyCharm*, так как он обеспечивает полный набор инструментов для разработки на *Python*, включая создание *GUI*, и имеет бесплатную версию *Community Edition*. Кроме того, *PyCharm* имеет удобный интерфейс и высокую степень автоматизации, что позволяет ускорить процесс разработки.

1.2 Проектирование и разработка

Состав приложения включает серверную и клиентскую части. Сервер собирает, обрабатывает и хранит информацию о пользователях, а клиентская часть предоставляет доступ только к тренажеру. Результаты пройденных сессий можно просмотреть только на серверной части. Перед началом работы сервера происходит инициализация всех компонентов и модулей системы,

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		17

включая сам сервер и компонент, который определяет фрагменты на изображении. Также в

это время происходит подключение к СУБД.

1.2.1 Реализация серверной части

Серверная часть приложения реализована на языке *Python* с использованием *Flask*-фреймворка.

В начале файла *import*-ами подключены необходимые библиотеки и модули: *Flask*, *datetime*, *jsonify* из библиотеки *flask*, а также модули баз данных, настроек и инициализации базы данных.

Далее идет инициализация приложения и настройка секретного ключа:

```
app = Flask(__name__)  
app.config['SECRET_KEY'] = 'secret_key'
```

Затем идет инициализация базы данных с помощью функции *init_db()*.

Далее описываются обработчики запросов на сервер:

- *start_session()*: обработчик запроса на запуск тренажера. В этом обработчике парсятся данные, переданные в запросе, и создается запись в таблице *Session* базы данных. В этой записи сохраняются все переданные данные о сессии, а также дата и время ее запуска. В ответ на запрос возвращается *id* созданной сессии;

- *save_task()*: обработчик запроса на сохранение задания и ответа. При получении запроса парсится *id* сессии, задание, ответ и результат ответа. Далее создается запись в таблице *History* базы данных, в которой сохраняются все переданные данные о задании и ответе, а также дата и время его выполнения;

- *finish_session()*: обработчик запроса на окончание сессии. При получении запроса парсится *id* сессии и количество правильных ответов. Далее происходит обновление записи в таблице *Session* базы данных, в которой сохраняются все переданные данные о сессии, а также дата и время ее окончания;

- *get_stats()*: обработчик запроса на получение статистики. В этом обработчике происходит выборка всех записей из таблицы *Session* и

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		18

формирование списка словарей со всеми данными о каждой сессии. В ответ на запрос возвращается этот список в формате *JSON*;

- *status_server()*: обработчик запроса на проверку статуса сервера. В ответ на запрос возвращается *JSON*-объект со статусом сервера.

Эти обработчики принимают запросы от клиентской части приложения и обрабатывают их. Также, в основном файле проекта *server.py* реализован запуск сервера на хосте и порте, заданных в конфигурационном файле:

```
if __name__ == '__main__':  
    app.run(debug=True, host=HOST, port=PORT)
```

1.2.2 Разработка БД

Файл *database.py* содержит код для создания базы данных на основе моделей данных, созданных с использованием библиотеки *SQLAlchemy*.

Первым делом создается подключение к базе данных с помощью функции *create_engine* из модуля *SQLAlchemy*. В данном случае используется база данных *SQLite*, название которой определяется в файле конфигурации *DBNAME*.

Далее создается сессия для взаимодействия с базой данных с помощью функции *sessionmaker*. Также используется *scoped_session*, чтобы гарантировать, что каждый поток будет иметь свою собственную сессию.

Для создания таблиц в базе данных используется класс *Base* из модуля *declarative_base*, который создает новый класс-модель, от которого будут наследоваться остальные модели.

В файле *database.py* определены две модели: *Session* и *History*.

Модель *Session* содержит информацию о сессии пользователя: имя пользователя, уровень сложности, время прохождения сессии, количество заданий в сессии, количество правильно решенных заданий, статус (завершена или нет) и дату создания. Также у *Session* есть связь с моделью *History* через поле *history*.

Модель *History* содержит информацию о каждом задании, которое было решено в рамках сессии. Модель содержит поля: *id*, *session_id*, *task*, *answer*, *result* и *created_at*. Поле *session_id* связывает таблицы *Session* и *History*.

Наконец, функция `init_db()` вызывает метод `metadata.create_all()` у объекта *Base*, который создает все таблицы, описанные в файлах моделей.

Таким образом, этот файл содержит определения моделей данных и код для создания соответствующих таблиц в базе данных.

Для хранения результатов сессии была разработана следующая модель, *UML*-диаграмма которой представлена на рисунке 6.

Session
id = Column(Integer, primary_key=True) user_name = Column(String(50)) level = Column(Integer) time = Column(Integer) count_tasks = Column(Integer) count_correct = Column(Integer) status = Column(String(10)) created_at = Column(TIMESTAMP)

Рисунок 6 – *UML*-диаграмма класса для хранения результатов сессии

Для хранения ответов задач была разработана следующая модель, *UML*-диаграмма которой представлена на рисунке 7

History
id = Column(Integer, primary_key=True) user_name = Column(String(50)) level = Column(Integer) time = Column(Integer) count_tasks = Column(Integer) count_correct = Column(Integer) status = Column(String(10)) created_at = Column(TIMESTAMP)

Рисунок 7 – *UML*-диаграмма для хранения задач и ответов

На рисунке 8 представлена схема базы данных.

1.2.3 Разработка клиентской части

Для разработки клиентской части был использован язык программирования Python и фреймворк *PyQt5* для создания графического интерфейса [8].

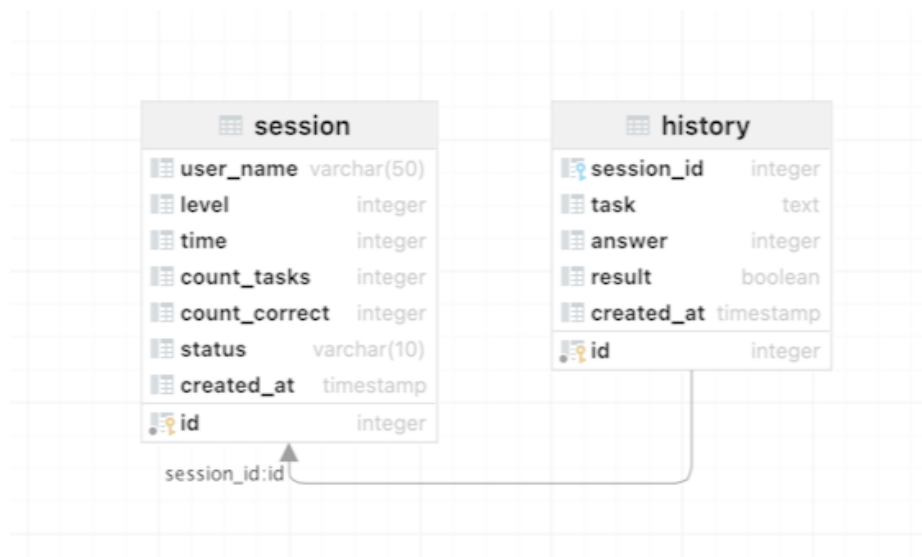


Рисунок 8 – Схема базы данных

Главный класс клиентской части — это *StartPage*, который отображает основное окно приложения. В этом окне пользователь может ввести свое имя, выбрать уровень сложности, время выполнения задач и количество задач. После заполнения всех полей пользователь нажимает на кнопку "Запустить тренажер".

При нажатии на кнопку, происходит проверка заполнения всех полей и отправка данных на сервер. Если сервер доступен и данные корректны, то создается объект *Trainer* и открывается окно с тренажером. Если же что-то пошло не так, то выводится сообщение об ошибке.

Для отправки данных на сервер используется библиотека *requests* [9]. Для проверки доступности сервера используется запрос на эндпоинт */status_server*. Если сервер вернул статус 200, значит сервер доступен и можно отправлять данные. Если сервер не доступен, выводится сообщение об ошибке.

Класс *Trainer* отвечает за отображение окна с тренажером. Он принимает на вход идентификатор сессии, время выполнения задач, количество задач и уровень сложности. При инициализации окна генерируется первая задача.

Для генерации задач используется метод *generate_task*, который генерирует два случайных числа и выводит вопрос на экран. Также он сохраняет правильный ответ для последующей проверки.

Пользователь вводит ответ на вопрос в поле ввода и нажимает на кнопку "Далее". При нажатии происходит проверка корректности введенных данных. Если данные корректны, то создается объект класса *ResultPage* и открывается окно с результатами. При закрытии окна *Trainer*, вызывается метод *update_status*, который отправляет на сервер данные о завершении сессии.

Класс *ResultPage* отвечает за отображение окна с результатами. Он принимает на вход количество правильных ответов и общее количество задач. После отображения результатов пользователь может нажать на кнопку "Начать заново", и откроется окно *StartPage* с возможностью начать новую сессию.

Для запуска приложения создается объект класса *QApplication*, затем создается объект *StartPage* и отображается на экране. Окно приложения закрывается при выходе из приложения.

Метод *closeEvent* - это метод класса *QWidget*, который вызывается при закрытии окна. Он позволяет обрабатывать события закрытия окна и выполнять дополнительные действия перед закрытием окна.

В данном проекте метод *closeEvent* используется для отправки на сервер данных о завершении сессии тренировки перед закрытием окна *Trainer*. Он вызывается автоматически, когда пользователь пытается закрыть окно. Если пользователь подтверждает закрытие окна, метод выполняет отправку данных на сервер и закрывает окно.

1.3 Анализ результатов

1.3.1 Панель администратора серверного приложения

Для анализа результатов работы тренажера необходимо создать панель администратора серверного приложения. На этой панели администратор сможет просмотреть статистику по всем сессиям тренажера, а также отсортировать сессии по различным параметрам, таким как уровень сложности, количество задач, время выполнения и т.д. (рисунок 9).

В окне содержатся следующие элементы:

- кнопка "Запустить сервер" - позволяет запустить серверную часть приложения. При нажатии на эту кнопку сервер начинает работу и готов принимать запросы от клиентов;

- кнопка "Отключить сервер" - позволяет отключить серверную часть приложения. При нажатии на эту кнопку сервер прекращает работу и перестает принимать запросы от клиентов;

- индикатор статуса сервера - отображает текущий статус сервера (запущен или отключен). Если сервер запущен, то индикатор "Запустить сервер" будет не активен, как показано на рисунке 10, если сервер отключен, то индикатор "Отключить сервер" будет не активен;

- кнопка "Обновить таблицу" - позволяет обновить таблицу статистики сессий. При нажатии на эту кнопку таблица будет обновлена и отобразит актуальную информацию о сессиях;

- таблица статистики сессий - отображает информацию о всех сессиях, которые были запущены на сервере. В таблице отображаются следующие поля:

- 1) *Date* - дата и время создания сессии. Это поле показывает, когда была запущена сессия;

- 2) *User* - имя пользователя, который запустил сессию. Это поле показывает, кто запустил сессию;

3) *Timer* - время, отведенное на выполнение заданий в сессии. Это поле показывает, сколько времени было отведено на выполнение заданий в сессии;

4) *Tasks* - количество задач в сессии и количество правильных ответов на них. Это поле показывает, сколько задач было в сессии и сколько из них было решено правильно. На рисунке 12 видим, что пользователь Андрей не выполнил ни одного задания из пяти;

5) *Status* - текущий статус сессии (Выполняется или Завершен). Это поле показывает, в каком состоянии находится сессия - запущена или завершена. Если сессия завершена, то в этом поле будет отображено "Завершен". На рисунке 13 показано, что пользователь Ирина завершила выполнение теста и сессия завершена, а пользователь Владимир выполняет тест и сессия имеет статус "Выполняется".

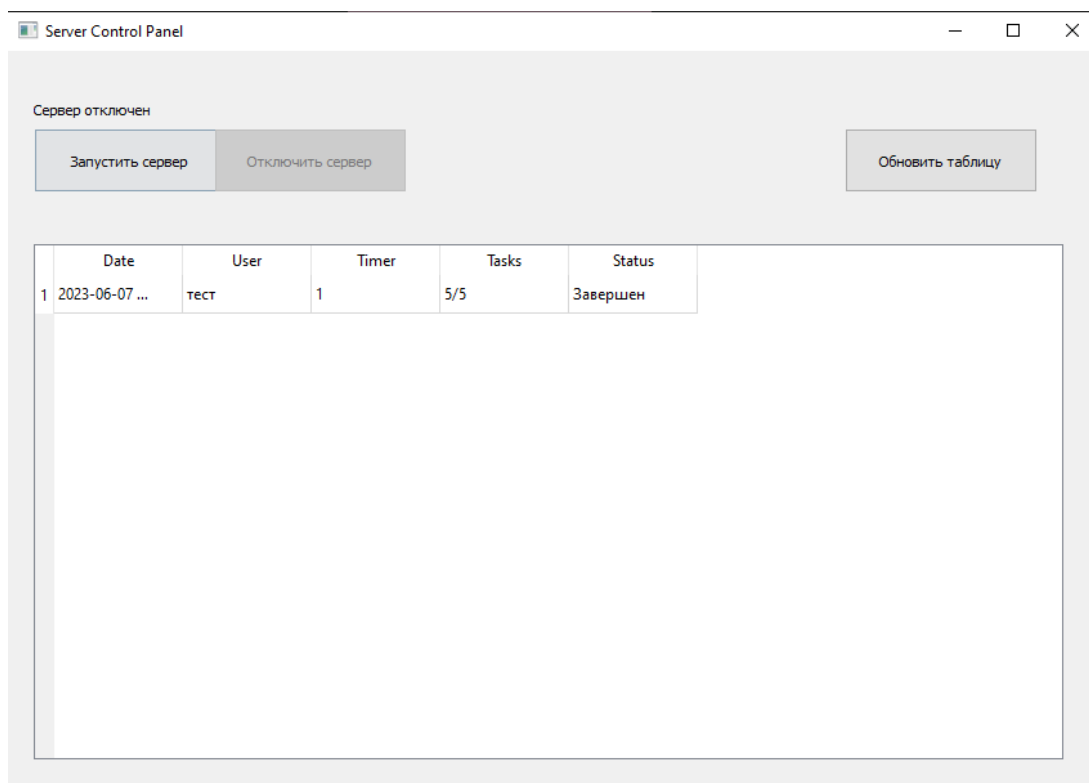


Рисунок 9 - Окно администратора серверной части

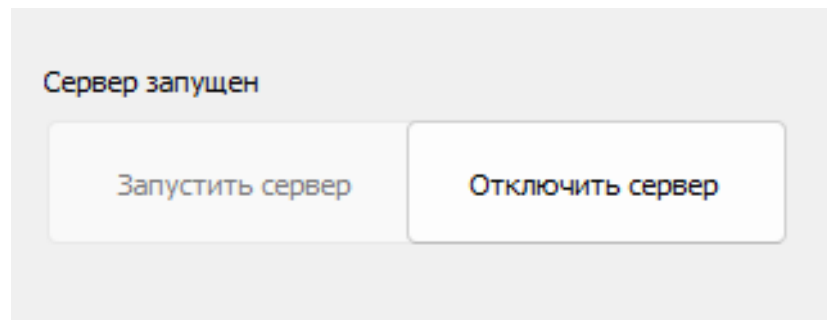


Рисунок 10 - Кнопки запуска и отключения сервера

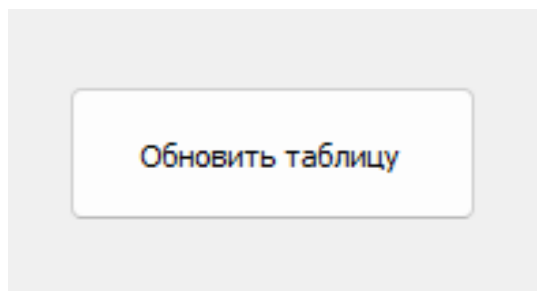


Рисунок 11 - Кнопка "Обновить таблицу"

	Date	User	Timer	Tasks	Status
1	2023-06-07 ...	тест	1	5/5	Завершен
2	2023-06-16 ...	Андрей	1	<u>0/5</u>	Завершен
3	2023-06-16 ...	Ирина	1	5/5	Завершен
4	2023-06-16 ...	Владимир	1	0/5	Выполняется

Рисунок 12 - Таблица статистики

	Date	User	Timer	Tasks	Status
1	2023-06-07 ...	тест	1	5/5	Завершен
2	2023-06-16 ...	Андрей	1	0/5	Завершен
3	2023-06-16 ...	Ирина	1	5/5	<u>Завершен</u>
4	2023-06-16 ...	Владимир	1	0/5	<u>Выполняется</u>

Рисунок 13 - Таблица статистики

1.3.2 Тестирование *REST API*

Для тестирования было использовано *Postman*.

Postman - это инструмент для тестирования и разработки *API*. Он позволяет пользователям отправлять запросы к *API*, а затем просматривать и анализировать ответы. *Postman* также позволяет создавать коллекции запросов и документацию для *API*.

Тестирование метода *start_session*:

- формируем *POST* запрос методом *start_session* на *http://127.0.0.1:5002/start*;
- указываем параметры: *user_name*, *level*, *time*, *count_tasks*;
- если все поля заполнены, то ожидаем в ответе статус 200 и *session_id* в формате *JSON*;
- если одно из полей не заполнено, то ожидаем в ответе статус 400 и сообщение об ошибке в формате *JSON*;
- если поле *time* или *count_tasks* содержит нечисловое значение, то ожидаем в ответе статус 400 и сообщение об ошибке в формате *JSON*.

На рисунке 14 представлен ответ *API* на запрос *http://127.0.0.1:5002/start*

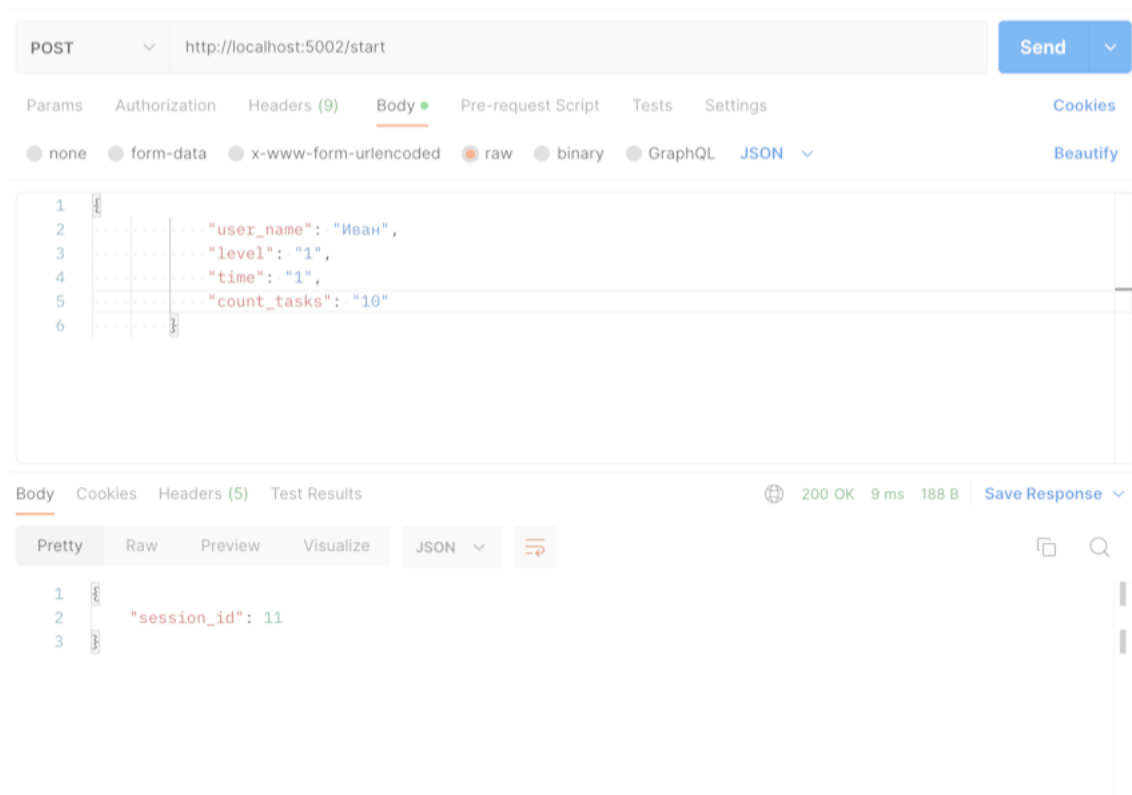


Рисунок 14 – Результат запроса `http://127.0.0.1:5002/start`

Тестирование метода `save_task`:

- формируем `POST` запрос методом `save_task` на `http://127.0.0.1:5002/save_task`;
- указываем параметры: `session_id`, `task`, `answer`;
- если все поля заполнены, то ожидаем в ответе статус 200 и сообщение об успешном сохранении в формате `JSON`;
- если одно из полей не заполнено, то ожидаем в ответе статус 400 и сообщение об ошибке в формате `JSON`.

На рисунке 15 представлен ответ `API` на запрос `http://127.0.0.1:5002/save_task`

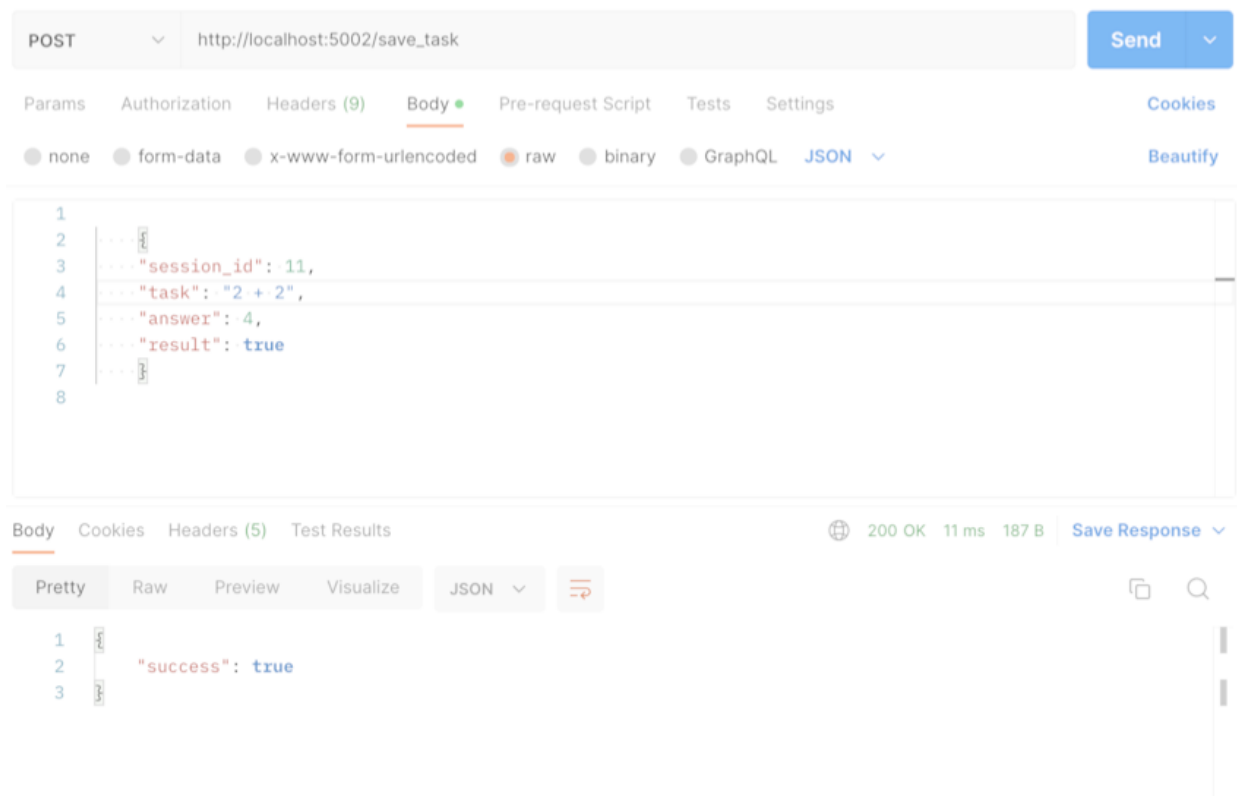


Рисунок 15 – Результат запроса `http://127.0.0.1:5002/ save_task`

Тестирование метода `finish_session`:

- формируем `POST`-запрос методом `finish_session` на `http://127.0.0.1:5000/finish_session`
- указываем параметры: `session_id`, `status`, `count_correct`;
- если все поля заполнены, то ожидаем в ответе статус 200 и сообщение об успешном завершении сессии в формате `JSON`;
- если поле `session_id` не заполнено или сессия с данным `id` не найдена, то ожидаем в ответе статус 404 и сообщение об ошибке в формате `JSON`;
- если одно из полей не заполнено, то ожидаем в ответе статус 400 и сообщение об ошибке в формате `JSON`.

На рисунке 16 представлен ответ `API` на запрос `http://127.0.0.1:5002/finish_session`

Тестирование метода `status_server`:

- формируем `GET` запрос методом `status_server` на `http://127.0.0.1:5000/status_server`;

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		28

- ожидаем в ответе статус 200 и сообщение об успешном подключении в формате *JSON*.

На рисунке 17 представлен ответ *API* на запрос *http://127.0.0.1:5002/status_server*.

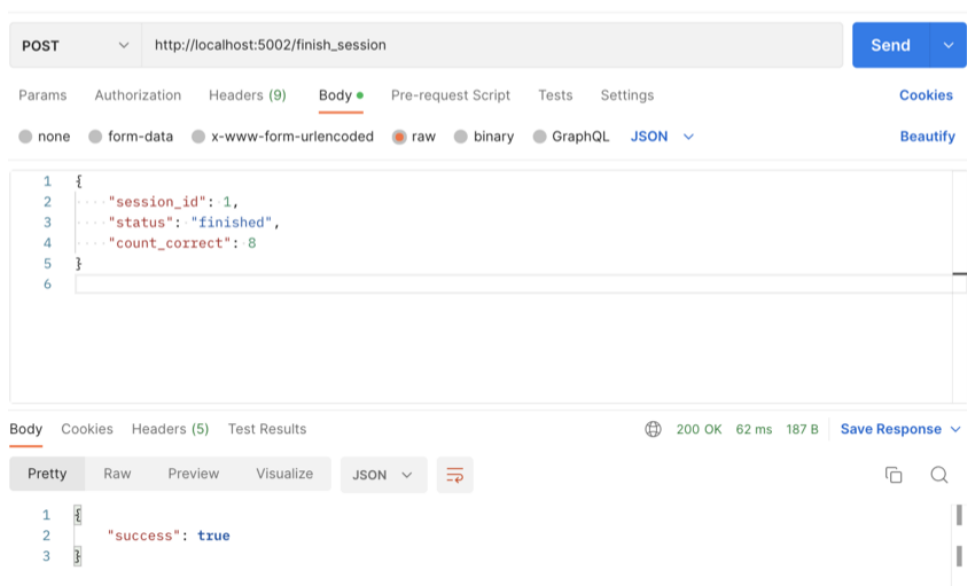


Рисунок 16 – Результат запроса *http://127.0.0.1:5002/finish_session*

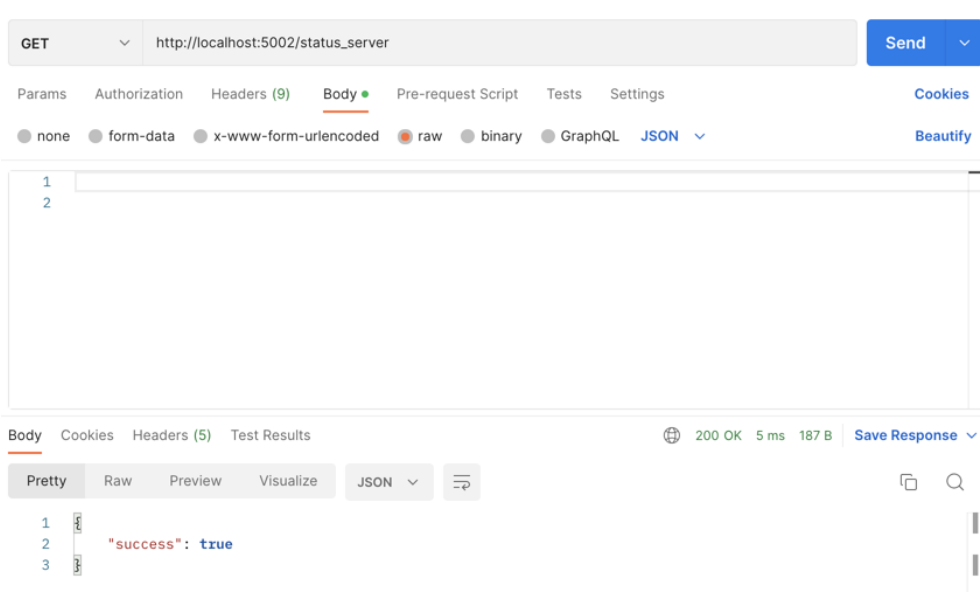


Рисунок 17 – Результат запроса *http://127.0.0.1:5002/status_server*.

1.3.3 Клиентская часть программы

На рисунке 18 представлена стартовое окно клиентской части

Рисунок 18 - стартовое окно клиентской части

Стартовое окно клиентской части имеет следующие поля ввода.

Поле для ввода имени пользователя. Это поле обязательно для заполнения и должно содержать от 3 до 20 символов, включая буквы латинского алфавита, цифры и знак подчеркивания. Если пользователь ввел некорректное значение, то должно появиться сообщение об ошибке и кнопка запуска тренажера должна быть заблокирована, пока пользователь не исправит ошибку.

Выбор уровня сложности от 1 до 5. Пользователь может выбрать уровень сложности задач, который будет определять количество и сложность задач, предлагаемых во время работы тренажера. По умолчанию выбран уровень сложности 1.

На рисунке 19 представлена выбор уровня сложности.

Выбор времени выполнения задач в минутах от 1 до 60. Пользователь может выбрать время, которое тренажер будет отводить на выполнение задач. По умолчанию выбрано время 1 минута.

На рисунке 20 представлена выбор времени выполнения задач в минутах.

Тренажер устного счета

Введите имя:

Тестовый

Выберите уровень сложности:

- 1
- 1
- 2
- 3
- 4
- 5

Введите количество задач:

5

Запустить тренажер

Рисунок 19 - Выбор уровня сложности.

Тренажер устного счета

Введите имя:

Тестовый

Выберите уровень сложности:

1

Введите время выполнения задач (в минутах):

- 1
- 1
- 5
- 10
- 15
- 20
- 30
- 45
- 60

Рисунок 20 - Выбор времени выполнения задач в минутах

Выбор количества задач от 5 до 100. Пользователь может выбрать количество задач, которые будут предложены во время работы тренажера. По умолчанию выбрано 10 задач.

На рисунке 21 представлена выбор количества задач.

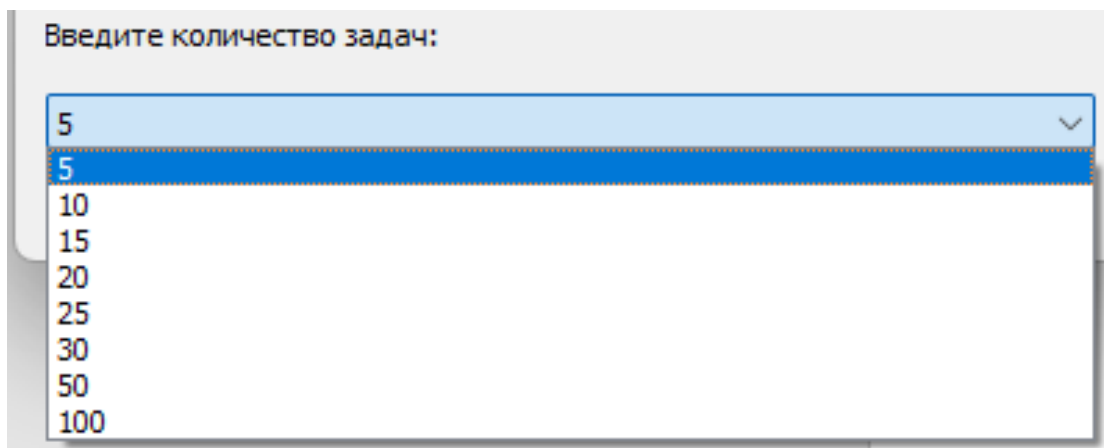


Рисунок 21 - Выбор количества задач

Кнопка запуска тренажера. Пользователь может нажать на эту кнопку, чтобы начать работу тренажера с выбранными параметрами. Если какое-то из полей ввода не заполнено или заполнено некорректно, то кнопка запуска тренажера не должна быть активной.

На рисунке 22 указано кнопка «Запустить тренажер».

Если при заполнении полей ввода стартового окна пользователь не заполнил какое-то из обязательных полей или заполнил его некорректно, то при попытке нажать на кнопку запуска тренажера должно появляться сообщение "Заполните все поля". Это сообщение должно быть ярко выделено на экране и привлекать внимание пользователя.

Сообщение должно появляться в том случае, если условия заполнения полей не соблюдены. Например, если пользователь не ввел имя пользователя или ввел имя, содержащее недопустимые символы, то должно появляться сообщение "Заполните все поля". Также сообщение должно появляться в том случае, если пользователь не выбрал уровень сложности, время выполнения задач или количество задач.

Рисунок 22 - Кнопка «Запустить тренажер».

Сообщение "Заполните все поля" должно исчезнуть, когда пользователь исправит ошибку и заполнит все обязательные поля ввода. После этого кнопка запуска тренажера должна стать активной и пользователь сможет начать работу с тренажером.

На рисунке 23 представлен сообщение «Заполните все поля».

После того, как пользователь заполнил все необходимые поля и нажал на кнопку "Запустить тренажер", открывается новое окно, которое содержит следующие элементы:

- таймер, отображающий оставшееся время выполнения задания;
- индикатор задач, показывающий, сколько задач осталось выполнить и сколько уже выполнено;
- задача, которую необходимо решить;
- поля для ввода ответа на задачу;
- кнопка "Далее", которая позволяет перейти к следующей задаче.

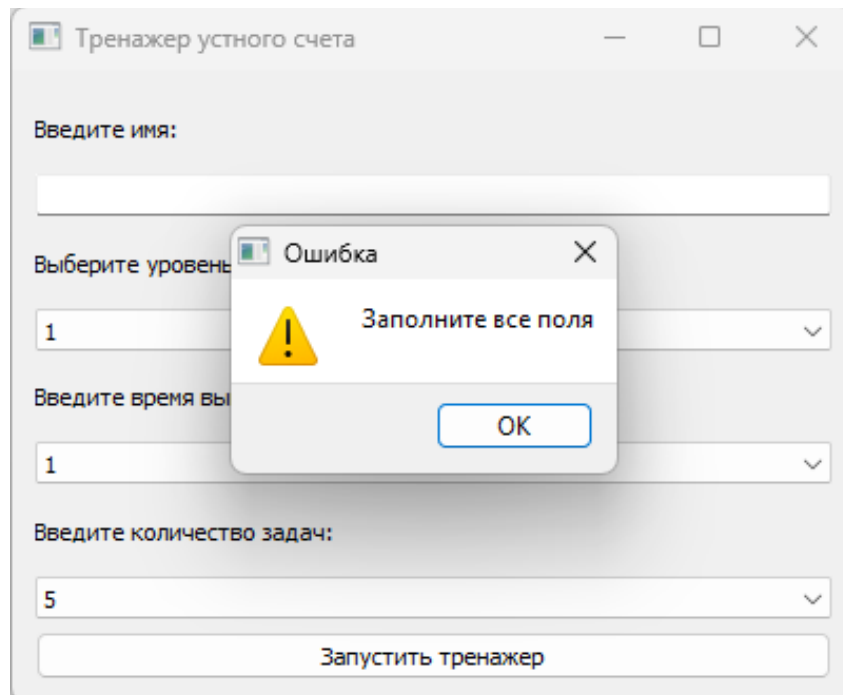


Рисунок 23 - Модальное окно сообщение «Заполните все поля»

Таймер отображает оставшееся время выполнения задания, которое пользователь выбрал на стартовой странице. Когда время истекает, тренажер автоматически переводит пользователя на следующую задачу.

На рисунке 24 показано окно тренажера.

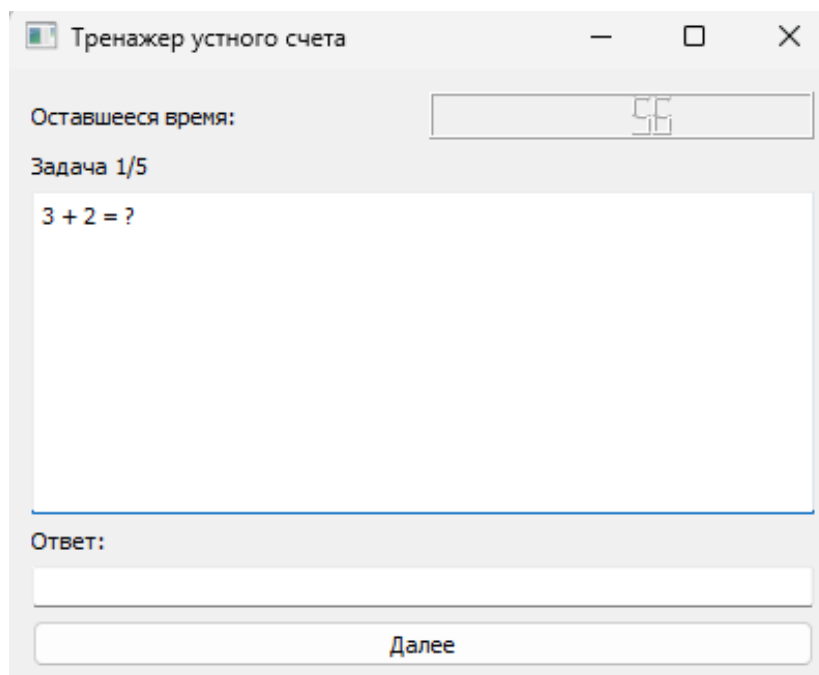


Рисунок 24 - Окно тренажера

Индикатор задач показывает, сколько задач осталось выполнить и сколько уже выполнено. Например, на старте индикатор может показывать "0/5", что означает, что пользователю предстоит выполнить 5 задач, но на данный момент ни одна из них не выполнена.

Задача, которую необходимо решить, отображается в соответствующем поле. Это может быть математическая задача, логическая задача, задача на знание иностранного языка и т.д.

Поля для ввода ответа на задачу позволяют пользователю ввести свой ответ. Например, для математической задачи это может быть поле для ввода числа или выражения.

Кнопка "Далее" позволяет перейти к следующей задаче. Когда пользователь вводит ответ на текущую задачу и нажимает на эту кнопку, тренажер проверяет правильность ответа и переводит пользователя на следующую задачу.

Таким образом, окно тренажера содержит все необходимые элементы, чтобы пользователь мог эффективно выполнять задания и улучшать свои навыки в выбранной области.

Модальное окно "Введите ответ" появляется, когда пользователь нажимает на кнопку "Далее" после ввода ответа на задачу. На рисунке 25 представлено модальное окно «Введите ответ».

Когда время выполнения задания истекает или пользователь выполнил все задания, сессия завершается, и открывается окно с результатами выполнения задания. Окно содержит следующую информацию:

- количество решенных задач;
- общее количество задач;
- кнопка "Попробовать еще раз".

Например, окно может содержать следующий текст: "Вы решили 1 из 5 задач. На рисунке 26 представлено окно «Результат».

Кнопка "Попробовать еще раз" позволяет пользователю повторить задание с теми же параметрами, которые он выбрал на стартовой странице.

Кнопка "Закрыть" позволяет пользователю закрыть окно и вернуться на стартовую страницу.

Таким образом, окно результатов выполнения задания позволяет пользователю узнать свой результат, оценить свои навыки и принять решение о том, повторить задание или вернуться к выбору параметров на стартовой странице.

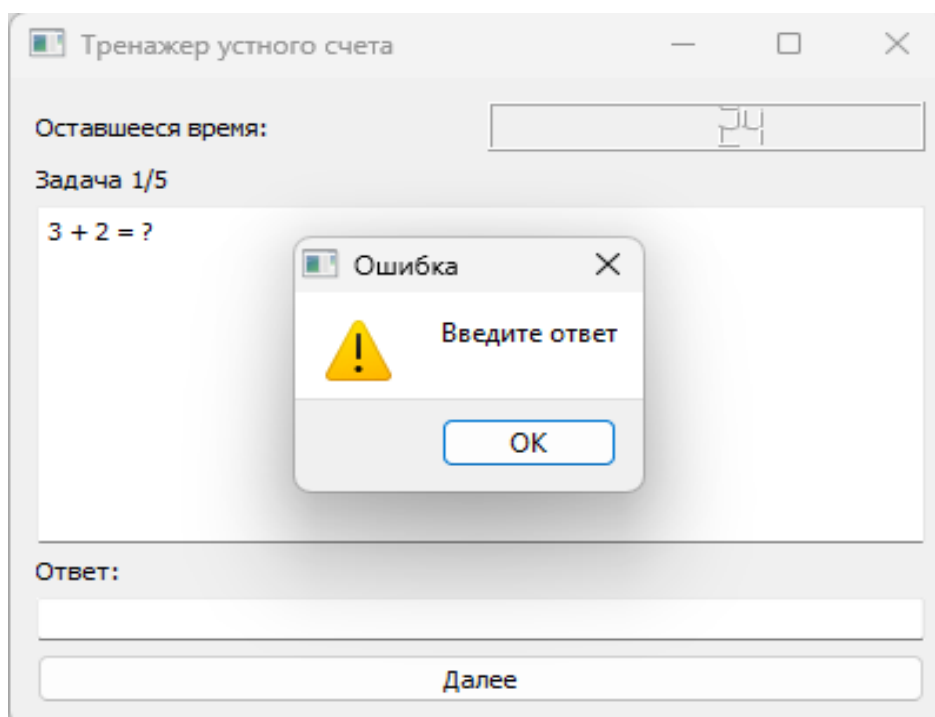


Рисунок 25 - Окно сообщение «Введите ответ»

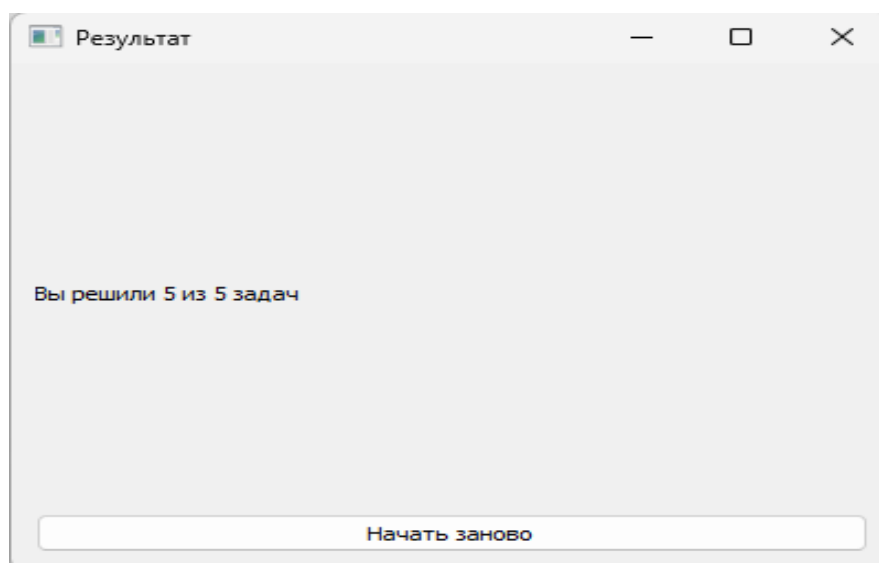


Рисунок 26 – Окно «Результат»

2 Охрана труда

В этом разделе описываются требования по обеспечению безопасности жизнедеятельности и охраны труда для пользователей компьютеров, которые участвуют в создании и использовании программного обеспечения, а также методы для снижения вредных воздействий на здоровье.

2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Компьютеры помогают людям решать задачи, однако они могут быть опасны для здоровья, как для пользователя, так и для окружающих людей. Чтобы предотвратить травмы, необходимо соблюдать рекомендации по безопасности, такие как ограничение времени работы за компьютером, правильное расположение рабочего места, соблюдение норм освещения и уровня шума.

2.2 Классификация и категорирование помещения, в котором производится разработка

2.2.1 Классификация по взрывопожарной и пожарной опасности

Категории помещений и зданий зависят от взрыво- и пожароопасности, которую определяют горючие вещества и материалы, их количество, а также планировочные решения и характеристики процессов. Помещения подразделяются на категории А, Б, В1-В4, Г и Д, а здания - на категории А, Б, В, Г и Д. Ниже приведены характеристики веществ и материалов, которые находятся в помещении, определяющие его категорию:

- «А» - опасность взрыва и пожара высокая, включая газы и легковоспламеняющиеся жидкости с температурой вспышки не выше 28 °С в количестве, способном образовать взрывоопасные смеси, а также вещества и материалы, способные взрываться или гореть при контакте с водой, кислородом, воздухом или друг с другом, при этом создавая избыточное

давление взрыва в помещении более 5 кПа;

- «Б» - опасность взрыва и пожара средняя, включая горючие пыли и волокна, легковоспламеняющиеся жидкости с температурой вспышки выше 28 °С, и горючие жидкости, которые могут образовывать взрывоопасные пылевоздушные и паровоздушные смеси, приводящие к избыточному давлению взрыва в помещении более 5 кПа;

- «В1-В4» - опасность пожара разная для горючих и трудногорючих жидкостей, веществ и материалов, которые могут гореть при взаимодействии с водой, кислородом, воздухом или друг с другом;

- «Г» - опасность пожара умеренная для негорючих веществ и материалов в раскаленном состоянии, которые могут выделять лучистое тепло, искры и пламя при обработке;

- «Д» - пониженная опасность пожара для материалов в холодном состоянии и негорючих веществ.

2.2.2 Классификация по опасности поражения электрическим током

Условия повышенной опасности поражения людей электрическим током:

- воздух с высокой влажностью (свыше 75%), где образуется конденсат или пар;

- наличие пыли, которая проводит электричество и оседает на проводах, внутри аппаратов и машин, ухудшая изоляцию и охлаждение, однако не является источником опасности для возникновения пожара или взрыва;

- использование металлических, земляных, железобетонных кирпичных оснований;

- высокая температура (в течение всего года более 35°С или кратковременно более 40°С).

Условия особой опасности поражения людей электрическим током:

- наличие сырости (дождь, снег, частое опрыскивание и покрытие влагой потолка, стен и т.д.);

- наличие химически активной среды;

– наличие одновременно двух или более условий повышенной опасности.

Таким образом можно выделить три категории опасности помещений: безопасные, с повышенной опасностью и особо опасные.

Для минимизации рисков поражения электрическим током рекомендуется работать в помещениях, соответствующих следующим критериям:

- температура воздуха не ниже 30°C, рекомендованный диапазон 18-20°C;
- полы должны быть выполнены из диэлектрического материала;
- помещение должно проветриваться;
- влажность воздуха не превышает 60%.

2.3 Санитарно-гигиеническая характеристика производства

Обязательными санитарно-гигиеническими критериями являются: сухость помещения, температура внутри него и количество света.

Также существует перечень основных требований:

- каждое рабочее место должно быть в объёме не менее 15 м³;
- высота помещения не менее 3,2 м;
- ширина дверей не менее 0,8 м;
- ширина коридоров не менее 1,4 м.

Для обеспечения максимальной освещенности каждое рабочее место должно быть освещено окном, которое должно регулярно чиститься от пыли и грязи. В случае недостаточной естественной освещенности, необходимо установить электрические лампы. Фонари следует изготавливать из армированного стекла и защищать металлическими сетками.

Оптимальная температура в помещении составляет 18-21°C, а оптимальный уровень влажности - 40-60%. Руководство производства должно обеспечить чистую питьевую воду в каждом рабочем помещении и рекомендуется менять ее не реже, чем раз в сутки.

					ПГУ1.090301.24.001 ПЗ	
ИЗ	Лист	№ докум.	Подпись	Дата		39

Помещения, в которых уровень шума превышает норму, должны быть помечены специальными знаками, а каждый работник должен быть проинструктирован и знать правила безопасности при работе в шумных условиях.

2.4 Расчет контура заземления производственного помещения, где установлены ПЭВМ

Для безопасного использования электроприборов, подключенных к электрической сети, необходимо учитывать высокий риск поражения током. В помещениях, где установлены компьютеры, используют заземляющие контуры, которые обеспечивают защиту пользователей. Для создания заземляющей магистрали используют полосовую или круглую сталь с сечением около 48 мм². Минимальные размеры стальных заземлителей и заземляющих проводников приведены в таблице 2.1.

Таблица 2.1 - Минимальные размеры стальных заземлителей и заземляющих проводников, мм

Заземлитель	Место расположения		
	В зданиях	В наружных установках	В земле
Круглые, диаметром, мм	5	6	10
Прямоугольные, сечением, мм ²	24	48	48
Прямоугольные, толщиной, мм ²	3	4	4
Угловая сталь с толщиной полос, мм	2	2,5	4
Стальные водопроводные (некондиционные) трубы с толщиной стенок, мм	2,5	2,5	3,5

Для соединения заземляющего проводника с заземлением необходимо использовать сварку в двух местах на нахлестке с длиной, равной удвоенной

ширине проводника. В зависимости от номинального тока потребителя, диаметр болта может быть 4 мм при токе до 16 А или 10 мм для потребителей от 250 до 30 А.

В таблице 2.2 приведены допустимые значения сопротивления заземляющих устройств в электроустановках. I_p – расчетный ток замыкания на землю. Сопротивление заземляющего устройства для электроустановки сетей с учетом повторных заземлений нулевого провода должно быть не более 2, 4 и 8 Ом при линейных напряжениях соответственно 660, 380 и 220В источника трехфазного тока и напряжениях 380, 220 и 127 В источника однофазного тока.

Таблица 2.2 - Наибольшие допустимые значения сопротивлений заземляющих устройств электроустановок

Характеристика объекта	Сопротивление, Ом
Электроустановки напряжением 110кВ и выше сетей с эффективным заземлением нейтрали, выполненные по нормам на сопротивление	0,5 $0,002 \cdot 0,5$
Электроустановки 3-35 кВ сетей с изолированной нейтралью	
Электроустановки сетей напряжением до 1000 В с глухозаземленной нейтралью напряжением: 660/380 В 380/220 В 220/127 В	15 ($15 \cdot 0,01$) 30 ($30 \cdot 0,01$) 60 ($60 \cdot 0,01$)
Электроустановки сетей напряжением до 1000 В с изолированной нейтралью при мощности источника питания: более 100 кВА До 100 кВА	$50/I_p$, но не более 4 Ом $50/I_p$, но не более 10 Ом

Имеется заземлитель диаметром 25...30 мм. Рассчитаем сопротивление растеканию тока R по формуле, где p – сопротивление грунта, l – длина трубы,:

$$R = 0,9 \cdot (p / l), \quad (1)$$

В нашем случае:

$$R = 0,9 \cdot (420 / 4) = 94,5 \text{ (Ом)}.$$

Определим примерное число заземлителей без учёта коэффициента экранирования при допустимом сопротивлении заземляющего устройства 2 Ом:

$$n = 94,5 / 2 = 47,25 \text{ (шт)}.$$

Определим число вертикальных заземлителей n_1 с учетом коэффициента экранирования:

$$n_1 = 47,25 / 0,36 = 131,25 \text{ (шт)}.$$

Рассчитаем длину соединительной полосы l_n , где a – расстояние между вертикальными заземлителями, при расстоянии между трубами 2,75 м:

$$l_n = n_1 \cdot a = 131,25 \cdot 2,75 = 360,94, \quad (2)$$

Периметр помещения с ПЭВМ 200 (м). Расчетная длина соединительной полосы не менее периметра помещения. Рассчитаем сопротивление растеканию электрического тока через соединительную полосу:

$$R_n = 2,1 \cdot (420 / 360,94) = 2,44.$$

Определим результирующее сопротивление растеканию тока всего устройства R_3 , Ом.

$$R_3 = (94,5 \cdot 2,44) / (0,21 \cdot 94,5 + 0,36 \cdot 2,44 \cdot 131,25) = 230,58 / (19,85 + 115,29) = 230,58 / 135,14 = 1,71.$$

Допустимое сопротивление заземляющего устройства на электрических установках напряжением до 1000 В равно 1,71 Ом. Это значение не превышает 4 Ом, следовательно, результирующее сопротивление растеканию тока установлено верно и соответствует норме.

Заключение

В ходе работы было разработано клиент-серверное приложение "Тренажер устного счета". Для клиентской части использовался графический интерфейс пользователя на основе библиотеки *PyQt 5*, а для серверной части - фреймворк *Flask*.

Была разработана база данных *SQLite* с использованием библиотеки *SQLAlchemy* для хранения информации о сессиях пользователей и их результатов. Были реализованы функции сбора и сохранения данных о выполненных заданиях, сбора и сохранения статистики о сессиях.

Были реализованы системы анализа текстовых и графических данных, которые позволяют анализировать данные о выполненных заданиях и сессиях пользователей.

Также было разработано собственное *REST API*, которое позволяет взаимодействовать с сервером и получать информацию о выполненных заданиях и статистике о сессиях.

В результате получилось полноценное приложение, которое может использоваться для тренировки устного счета и сбора статистики об успехах пользователей. Приложение может быть улучшено путем добавления новых функций и улучшения интерфейса для более удобного использования.

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		43

Список использованных источников

1. Гурвиц Г. Разработка реального приложения в среде клиент-сервер – ДВГУПС, 2005, 206 с.
2. Дубаков А.А. Сетевое программирование: учебное пособие / А.А. Дубаков – СПб: НИУ ИТМО, 2013. – 248 с.
3. Мейер Б. Объектно-ориентированное программирование и программная инженерия / Мейер Б.— Электрон. текстовые данные.— М.: Интернет Университет Информационных Технологий (ИНТУИТ), 2016.— 285с.
4. Федоров, Д. Ю. Программирование на языке высокого уровня *Python* : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с. – (Бакалавр. Прикладной курс). – ISBN 978-5-534-10971-9. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: <https://urait.ru/bcode/437489> (дата обращения: 13.05.2023).
5. Шелудько, В. М. Основы программирования на языке высокого уровня *Python*: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 146 с. – ISBN 978-5-9275-2649-9. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: <http://www.iprbookshop.ru/87461.html> (дата обращения: 13.05.2023). – Режим доступа: для авторизир. пользователей
6. Шелудько, В. М. Язык программирования высокого уровня *Python*. Функции, структуры данных, дополнительные модули: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 107 с. – ISBN 978-5-9275-2648-2. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: <http://www.iprbookshop.ru/87530.html> (дата обращения: 13.05.2023). – Режим доступа: для авторизир. Пользователей
7. Доусон М. Програмируем на *Python*. – СПб.: Питер, 2014. – 416 с.

8. Прохоренок Н.А. *Python 3* и *PyQt*. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.

9. Пилгрим Марк. Погружение в *Python 3* (*Dive into Python 3* на русском)

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		45

Приложение А

(обязательное)

Тренажер устного счета

UML диаграммы

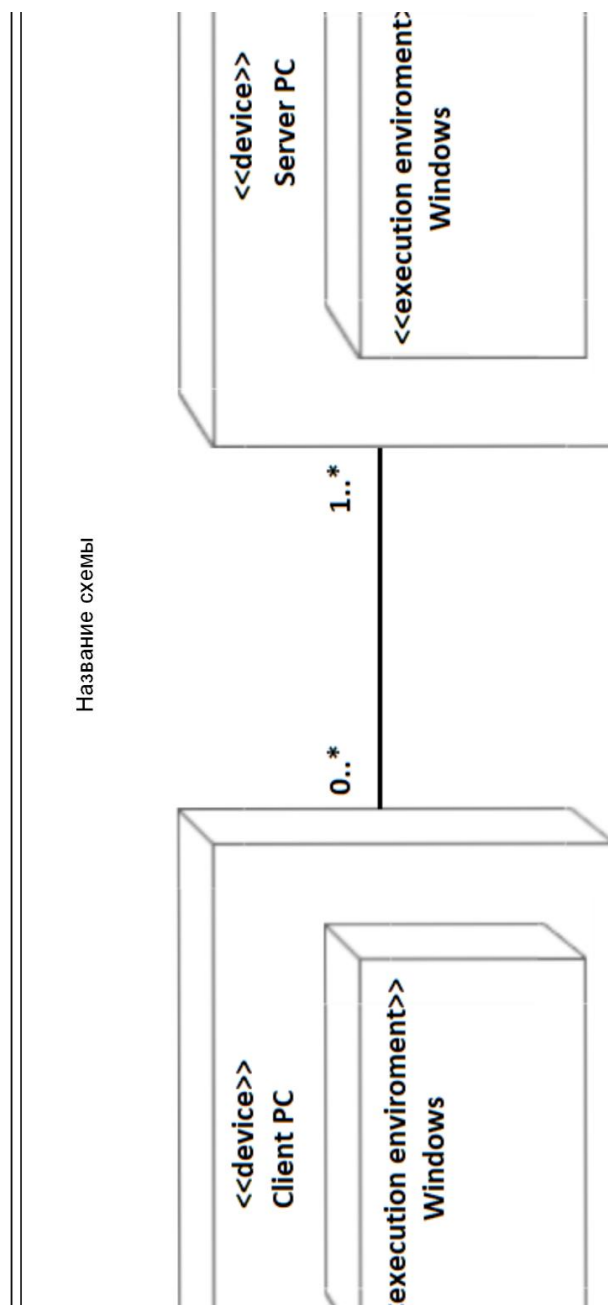
					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		46

Приложение А.1 – *UML* – диаграмма классов

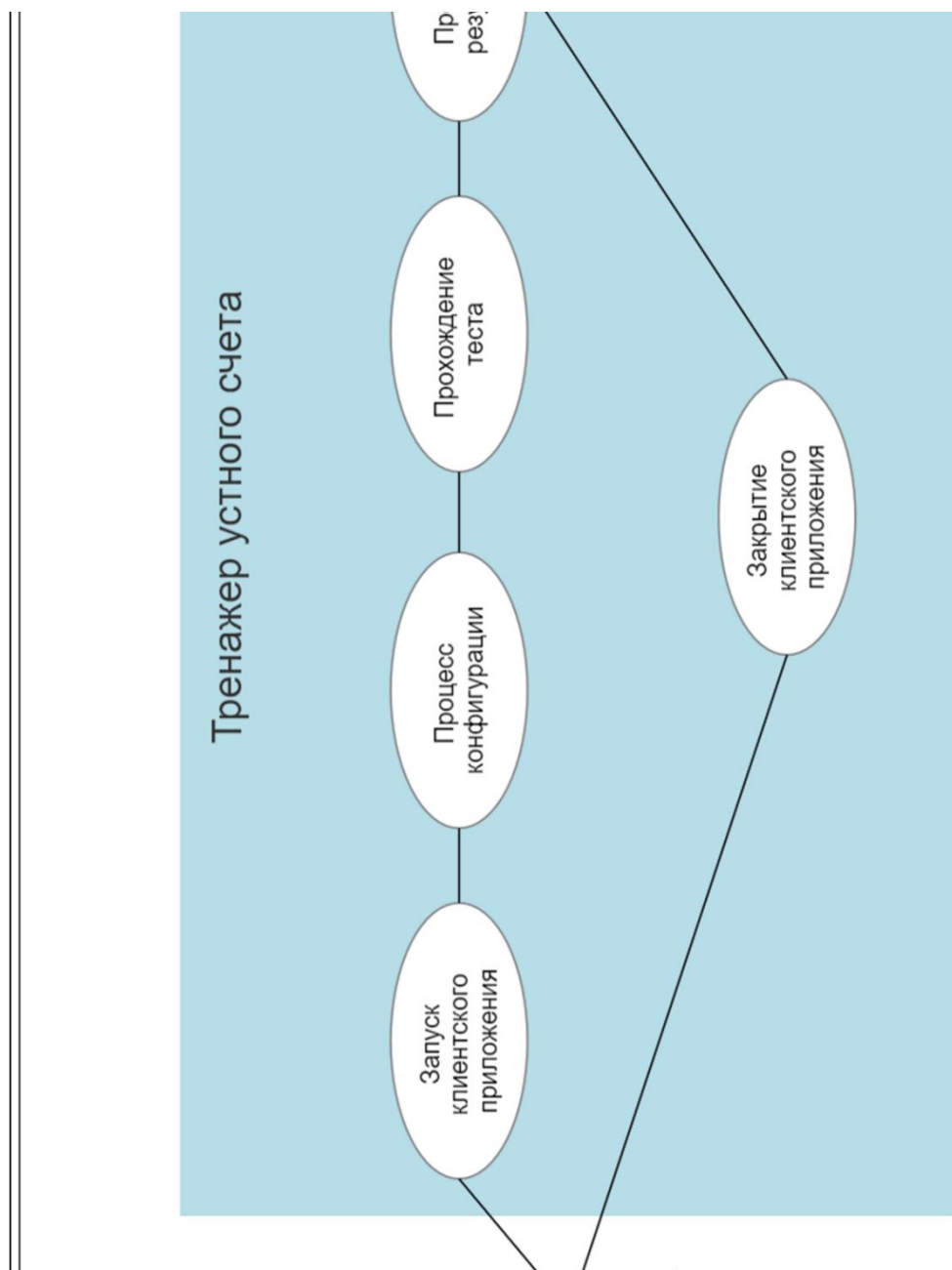


					ПУУ1.090301.24.002 Д11		
Имя	Фамилия	Имя	Фамилия	Оценки	Лист	Место	Maximum of
Тренер	Тренер А.Б.						
Тренер	Дубровский А.В.						
Тренер					Лист	Листов	
Место	Выход А.С.						

Приложение А.2 – UML – диаграмма развертывания



Приложение А.3 – UML – диаграмма использования



Приложение Б

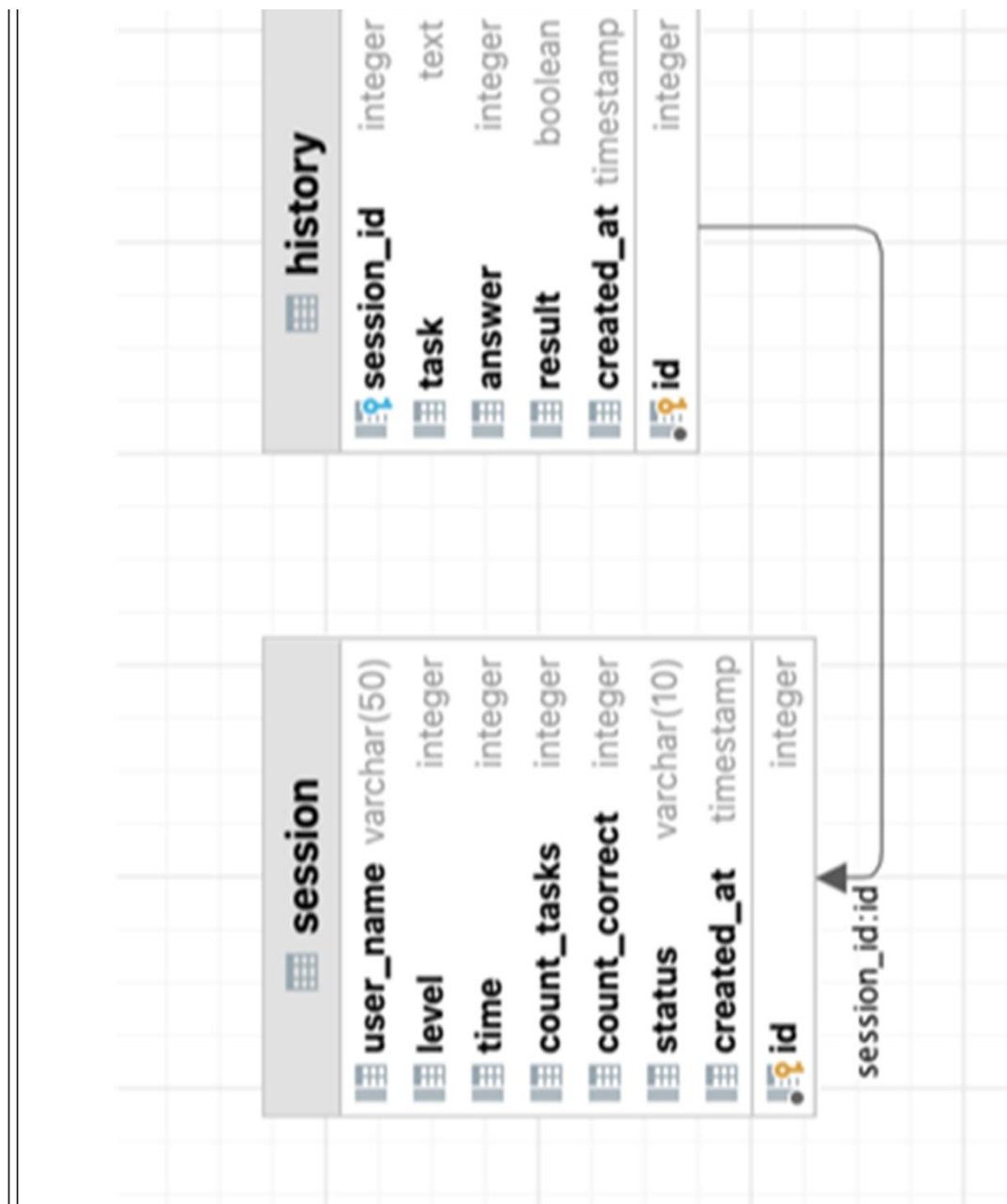
(обязательное)

Тренажер устного счета

Физическая модель базы данных

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		50

Приложение Б.1 – физическая модель базы данных



Приложение В

(обязательное)

Тренажер устного счета

Листинг программы

Файл *Database.py*

```
from sqlalchemy import create_engine, Column, Integer,
String, Boolean, ForeignKey, Text, TIMESTAMP
from sqlalchemy.orm import scoped_session, sessionmaker,
relationship
from sqlalchemy.ext.declarative import declarative_base
from config import DBNAME

engine = create_engine(f'sqlite:/// {DBNAME}.db')
db_session = scoped_session(sessionmaker(bind=engine))

Base = declarative_base()
Base.query = db_session.query_property()

class Session(Base):
    __tablename__ = 'session'
    id = Column(Integer, primary_key=True)
    user_name = Column(String(50))
    level = Column(Integer)
    time = Column(Integer)
    count_tasks = Column(Integer)
    count_correct = Column(Integer)
    status = Column(String(10))
    created_at = Column(TIMESTAMP)

    history = relationship('History', backref='session',
lazy='dynamic')

class History(Base):
    __tablename__ = 'history'
    id = Column(Integer, primary_key=True)
    session_id = Column(Integer, ForeignKey('session.id'))
    task = Column(Text)
    answer = Column(Integer)
    result = Column(Boolean)
    created_at = Column(TIMESTAMP)

def init_db():
    Base.metadata.create_all(bind=engine)
```

Файл *start_server.py*

```
import subprocess
```

```

import sys
import time
import psutil

import requests
from PyQt5.QtWidgets import QApplication, QWidget, QLabel,
QPushButton
from PyQt5.QtCore import Qt, QProcess
from config import HOST, PORT, APPNAME

BASE_URL = f'http://{HOST}:{PORT}'

from sessions_table import SessionsTable

class ServerWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.process = None
        self.initUI()
        #self.start_server()

    def initUI(self):
        self.setGeometry(100, 100, 850, 770)
        self.setWindowTitle('Server Control Panel')

        self.status_label = QLabel('Сервер отключен', self)
        self.status_label.setGeometry(20, 20, 150, 50)

        self.start_button = QPushButton('Запустить сервер', self)
        self.start_button.setGeometry(20, 60, 150, 50)
        self.start_button.clicked.connect(self.start_server)

        self.stop_button = QPushButton('Отключить сервер', self)
        self.stop_button.setGeometry(160, 60, 150, 50)
        self.stop_button.clicked.connect(self.stop_server)
        self.stop_button.setEnabled(False)

        self.refresh_button = QPushButton('Обновить таблицу',
self)
        self.refresh_button.setGeometry(650, 60, 150, 50)
        self.refresh_button.clicked.connect(self.refresh_table)

        self.sessions_table = SessionsTable()
        self.sessions_table.setGeometry(20, 150, 800, 400)
        self.sessions_table.load_data()

        self.sessions_table.setParent(self)
    def start_server(self):
        if self.process is None or self.process.poll() is not

```

```

None:
    # self.process =
    subprocess.Popen(['venv\\Scripts\\python.exe', 'server.py'])
    self.process = subprocess.Popen(['server.exe'])
    self.status_label.setText('Сервер запущен')

    self.start_button.setEnabled(False)
    self.stop_button.setEnabled(True)

def stop_server(self):
    for proc in psutil.process_iter():
        if proc.name() == "server.exe":
            proc.kill()
    self.status_label.setText('Сервер отключен')
    self.start_button.setEnabled(True)
    self.stop_button.setEnabled(False)

def refresh_table(self):
    self.sessions_table.load_data()

def closeEvent(self, event):
    if self.process is not None and self.process.poll() is
None:
        self.process.terminate()
        self.process.wait()
    event.accept()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = ServerWindow()
    window.show()
    sys.exit(app.exec_())

```

Файл *server.py*

```

from flask import Flask, request, jsonify
from datetime import datetime
from database import db_session, init_db, Session, History
from config import HOST, PORT
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret_key'

# инициализация базы данных

init_db()
# страница запуска тренажера

```

```

        @app.route('/start', methods=['POST'])
def start_session():
    data = request.json
    user_name = data.get('user_name')
    level = data.get('level')
    time = data.get('time')
    count_tasks = data.get('count_tasks')

    if not user_name or not level or not time or not count_tasks:
        return jsonify({'error': 'Заполните все поля'}), 400

    if not time.isdigit() or not count_tasks.isdigit():
        return jsonify({'error': 'Время и количество задач должны
        быть числами'}), 400

    session = Session(user_name=user_name, level=int(level),
    time=int(time), count_correct=0, count_tasks=int(count_tasks),
    status='Выполняется', created_at=datetime.now())
    db_session.add(session)
    db_session.commit()

    return jsonify({'session_id': session.id})

# Сохраняет задание и ответ
@app.route('/save_task', methods=['POST'])
def save_task():
    data = request.json
    session_id = data.get('session_id')
    task = data.get('task')
    answer = data.get('answer')
    result = data.get('result')

    if not session_id or not task or not answer:
        return jsonify({'error': 'Заполните все поля'}), 400

    history = History(session_id=session_id, task=task,
    answer=int(answer), result=bool(result),
    created_at=datetime.now())
    db_session.add(history)
    db_session.commit()

    return jsonify({'success': True}), 200

@app.route('/finish_session', methods=['POST'])
def finish_session():
    data = request.json
    session_id = data.get('session_id')

    count_correct = data.get('count_correct')

```

```

        session = Session.query.filter_by(id=session_id).first()
    if not session:
        return jsonify({'error': 'Сессия не найдена'}), 404

    session.status = data.get('status')
    session.count_correct = int(count_correct)
    db_session.commit()

    return jsonify({'success': True}), 200

@app.route('/stats')
def get_stats():

    sessions = Session.query.all()
    stats = []
    for session in sessions:
        stats.append({
            'id': session.id,
            'user_name': session.user_name,
            'level': session.level,
            'time': session.time,
            'count_tasks': session.count_tasks,
            'count_correct': session.count_correct,
            'status': session.status,
            'created_at': session.created_at.strftime('%Y-%m-%d
%H:%M:%S')
        })
    return jsonify(stats)

@app.route('/status_server')
def status_server():
    return jsonify({'success': True}), 200

@app.route('/shutdown', methods=['POST'])
def shutdown_server():
    os._exit(0)

if __name__ == '__main__':
    app.run(debug=False, host=HOST, port=PORT)
    Файл start_client.py

    import sys
    import requests
    from PyQt5.QtWidgets import QApplication, QWidget, QLabel,
    QLineEdit, QComboBox, QPushButton, QMessageBox, QVBoxLayout
    from trainer_page import Trainer
    from config import HOST, PORT, APPNAME

    BASE_URL = f'http://{HOST}:{PORT}'

```

```

class StartPage(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle(APPNAME)
        self.setGeometry(300, 400, 400, 300)

        self.user_name_label = QLabel('Введите имя:')
        self.user_name_input = QLineEdit()

        self.level_label = QLabel('Выберите уровень сложности:')
        self.level_combo = QComboBox()
        self.level_combo.addItem('1')
        self.level_combo.addItem('2')
        self.level_combo.addItem('3')
        self.level_combo.addItem('4')
        self.level_combo.addItem('5')

        self.time_label = QLabel('Введите время выполнения задач (в минутах):')
        self.time_combo = QComboBox()
        self.time_combo.addItem('1')
        self.time_combo.addItem('5')
        self.time_combo.addItem('10')
        self.time_combo.addItem('15')
        self.time_combo.addItem('20')
        self.time_combo.addItem('30')
        self.time_combo.addItem('45')
        self.time_combo.addItem('60')

        self.count_label = QLabel('Введите количество задач:')
        self.count_combo = QComboBox()
        self.count_combo.addItem('5')
        self.count_combo.addItem('10')
        self.count_combo.addItem('15')
        self.count_combo.addItem('20')
        self.count_combo.addItem('25')
        self.count_combo.addItem('30')
        self.count_combo.addItem('50')
        self.count_combo.addItem('100')

        self.start_button = QPushButton('Запустить тренажер',
self)
        self.start_button.clicked.connect(self.start_trainer)

        vbox = QVBoxLayout()
        vbox.addWidget(self.user_name_label)
        vbox.addWidget(self.user_name_input)
        vbox.addWidget(self.level_label)
        vbox.addWidget(self.level_combo)
        vbox.addWidget(self.time_label)
        vbox.addWidget(self.time_combo)
        vbox.addWidget(self.count_label)
        vbox.addWidget(self.count_combo)
        vbox.addWidget(self.start_button)

        self.setLayout(vbox)

        # Запускает тренажера
    def start_trainer(self):
        user_name = self.user_name_input.text()

        level = self.level_combo.currentText()
        time = self.time_combo.currentText()
        count = self.count_combo.currentText()

```

```

        if not user_name or not time or not count:
            QMessageBox.warning(self, 'Ошибка', 'Заполните все
поля')
            return

        if not time.isdigit() or not count.isdigit():
            QMessageBox.warning(self, 'Ошибка', 'Время и
количество задач должны быть числами')
            return

        try:
            response = requests.get(f'{BASE_URL}/status_server')
            if response.status_code != 200:
                raise Exception('Сервер не запущен')
        except:
            QMessageBox.warning(self, 'Ошибка', 'Не удалось
подключиться к серверу')
            return

        session_data = {
            'user_name': user_name,

            'level': level,
            'time': time,
            'count_tasks': count
        }
        response = requests.post(f'{BASE_URL}/start',
json=session_data)

        if response.status_code == 200:
            self.trainer = Trainer(response.json()['session_id'],
int(time), int(count), level)
            self.trainer.show()
            self.close()
        else:
            QMessageBox.warning(self, 'Ошибка', 'Не удалось
запустить тренажер')

if __name__ == '__main__':
    app = QApplication(sys.argv)
    start = StartPage()
    start.show()
    sys.exit(app.exec_())
    Файл trainer_page.py

    import sys
    import requests
    from PyQt5.QtWidgets import QApplication, QWidget, QLabel,
QTextEdit, QLineEdit, QPushButton, QMessageBox, QVBoxLayout,

```

```

        QHBoxLayout, QLCDNumber
from PyQt5.QtCore import Qt, QTimer
import random
from config import HOST, PORT, APPNAME

BASE_URL = f'http://{HOST}:{PORT}'

from result_page import ResultPage

class Trainer(QWidget):
    def __init__(self, session_id, time, count, level):
        super().__init__()
        self.session_id = session_id
        self.time = time
        self.count = count
        self.level = level
        self.current_task = 1
        self.correct_answers = 0
        self.initUI()

    def initUI(self):
        self.setWindowTitle(APPNAME)
        self.setGeometry(300, 300, 400, 300)

        self.timer_label = QLabel('Оставшееся время:')
        self.timer_lcd = QLCDNumber()

        self.timer_lcd.setDigitCount(5)
        self.timer_lcd.display(self.time * 60)

        self.task_number_label = QLabel(f'к')
        self.task_text = QTextEdit()
        self.task_text.setReadOnly(True)

        self.answer_label = QLabel('Ответ:')
        self.answer_input = QLineEdit()

        self.next_button = QPushButton('Далее')
        self.next_button.clicked.connect(self.next_task)

        hbox1 = QHBoxLayout()
        hbox1.addWidget(self.timer_label)
        hbox1.addWidget(self.timer_lcd)
        vbox = QVBoxLayout()
        vbox.addLayout(hbox1)
        vbox.addWidget(self.task_number_label)
        vbox.addWidget(self.task_text)
        vbox.addWidget(self.answer_label)
        vbox.addWidget(self.answer_input)
        vbox.addWidget(self.next_button)

        self.setLayout(vbox)

```

```

        self.generate_task()

self.timer = QTimer()

        self.timer.timeout.connect(self.update_timer)
self.timer.start(1000)

# Закрывает окно тренажера
def closeEvent(self, event):
    self.update_status()
    event.accept()

# Генерирует новую задачу
def generate_task(self):
    if self.level == '1':
        a = random.randint(1, 10)
        b = random.randint(1, 10)
    elif self.level == '2':
        a = random.randint(10, 100)
        b = random.randint(1, 100)
    elif self.level == '3':
        a = random.randint(100, 1000)
        b = random.randint(1, 1000)
    elif self.level == '4':
        a = random.randint(1000, 10000)
        b = random.randint(1, 10000)

    else:
        a = random.randint(10000, 100000)
        b = random.randint(1, 100000)

    operation = random.choice(['+', '-', '*', '/'])
    if operation == '+':
        self.task_text.setText(f'{a} + {b} = ?')
        self.correct_answer = a + b
    elif operation == '-':
        self.task_text.setText(f'{a} - {b} = ?')
        self.correct_answer = a - b
    elif operation == '*':
        self.task_text.setText(f'{a} * {b} = ?')
        self.correct_answer = a * b
    else:
        while True:
            b = random.randint(1, 100)
            if a % b == 0:
                self.task_text.setText(f'{a} / {b} = ?')
                self.correct_answer = a // b
                break

        self.task_number_label.setText(f'Задача
{self.current_task}/{self.count}')

```

```

        # Следующая задача
def next_task(self):
    answer = self.answer_input.text()

    try:
        response = requests.get(f'{BASE_URL}/status_server')
        if response.status_code != 200:
            raise Exception('Сервер не запущен')
    except:
        QMessageBox.warning(self, 'Ошибка', 'Не удалось
подключиться к серверу')
        return

    if not answer:
        QMessageBox.warning(self, 'Ошибка', 'Введите ответ')
        return

    if not all(char.isdigit() or char == '-' for char in
answer):
        QMessageBox.warning(self, 'Ошибка', 'Ответ должен
быть числом')
        return

    if int(answer) == self.correct_answer:

        result = 1
        self.correct_answers += 1
    else:
        result = 0

    data = {
        'session_id': self.session_id,

        'task': self.task_text.toPlainText(),
        'answer': answer,
        'result': result
    }

    requests.post(f'{BASE_URL}/save_task', json=data)
    if self.current_task == self.count:
        self.finish()
    else:
        self.current_task += 1
        self.answer_input.clear()
        self.generate_task()

# Обновляет время выполнения
def update_timer(self):
    remaining_time = self.timer_lcd.value()
    if remaining_time == 0:
        self.finish()
    else:

```

```

        self.timer_lcd.display(remaining_time - 1)
# Завершает сессию
def update_status(self):
    data = {
        'session_id': self.session_id,
        'count_correct': self.correct_answers,
        'status': 'Завершен'
    }
    requests.post(f'{BASE_URL}/finish_session', json=data)

# Завершает тренажер
def finish(self):
    self.timer.stop()
    self.update_status()
    self.result = ResultPage(self.count,
self.correct_answers)
    self.result.show()
    self.close()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    trainer = Trainer()
    trainer.show()
    sys.exit(app.exec_())
    Файл result_page.py

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel,
QPushButton, QVBoxLayout

class ResultPage(QWidget):
    def __init__(self, count, correct):
        super().__init__()
        self.count = count
        self.correct = correct
        self.initUI()
    def initUI(self):
        self.setWindowTitle('Результат')
        self.setGeometry(300, 300, 400, 300)

        self.result_label = QLabel(f'Вы решили {self.correct} из
{self.count} задач')
        self.restart_button = QPushButton('Начать заново')
        self.restart_button.clicked.connect(self.restart)

        vbox = QVBoxLayout()
        vbox.addWidget(self.result_label)
        vbox.addWidget(self.restart_button)

        self.setLayout(vbox)

```

```

        # Открывает главное окно
def restart(self):
    self.close()
    from start_client import StartPage
    self.start = StartPage()
    self.start.show()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    result = ResultPage()
    result.show()
    sys.exit(app.exec_())

```

Файл *Config.py*

```

APPNAME = 'Тренажер устного счета'
HOST = 'localhost'
PORT = 5002
DBNAME = 'statistics'

```

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		63

Приложение Г

(обязательное)

Тренажер устного счета

Презентация

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		64

Тема работы: Рекламер устного счет

Выполнил: студент группы 19ВВПП1 Урядов В.Д.

Научный руководитель: к.т.н., доцент каф. ВТ ПГУ Дубравин А.

ьность работы

нном мире, где математические знания являю
мой частью жизни, умение быстро и точно счи
ажным навыком. Однако, не все люди облада
и для того чтобы его развивать, необходимо пр
и.

ет и объект работы

ка тренажера устного счета на языке Python

ка образовательного приложения

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		67

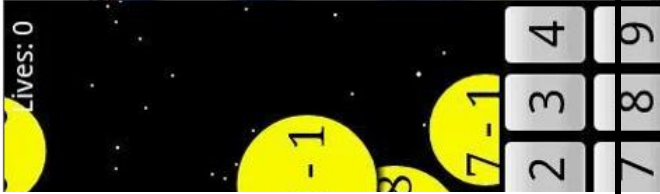
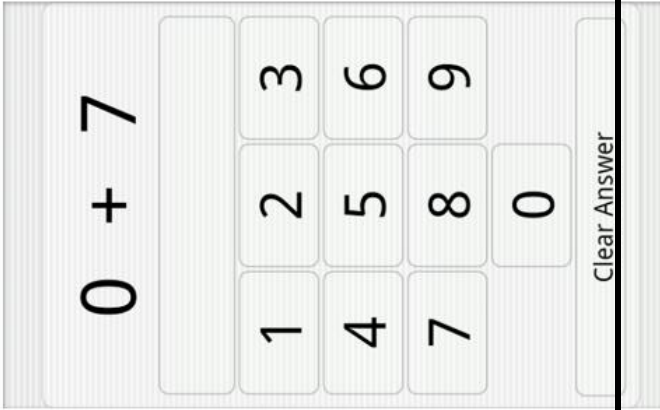
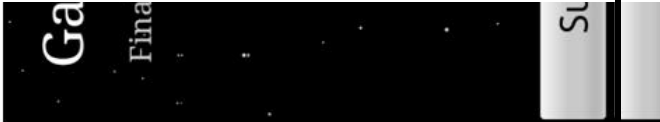
работы

работы заключается в разработке клиент-серверно
ния "Тренажер устного счета", которое позволит
ителям улучшить свои навыки устного счета и со
«у результатов для дальнейшего анализа

Овка задачи

- часть приложения должна иметь графически пользователя, позволяющий выбирать сложных заданий
- часть должна генерировать задания для полноты ответов на сервер для сохранения статистики.
- часть должна собирать статистику результатов ее в базу данных SQLite.

И Vorkout (Android)



GAME TYPE	MATHEMATICAL PUZZLE GAME
<div> <div></div> <div>subtraction</div> </div>	<div> <div>$X \div Y = Z$</div> <div>division</div> </div>
<div> <div></div> <div>addition</div> </div>	<div> <div>$(X \div Y) \div Z$</div> <div>division</div> </div>
<div> <div></div> <div>equality</div> </div>	<div> <div></div> <div>equality</div> </div>
<div> <div></div> <div>back</div> </div>	<div> <div></div> <div>back</div> </div>

LEVEL TYPE SCORE
7 X-Y-Z **500**

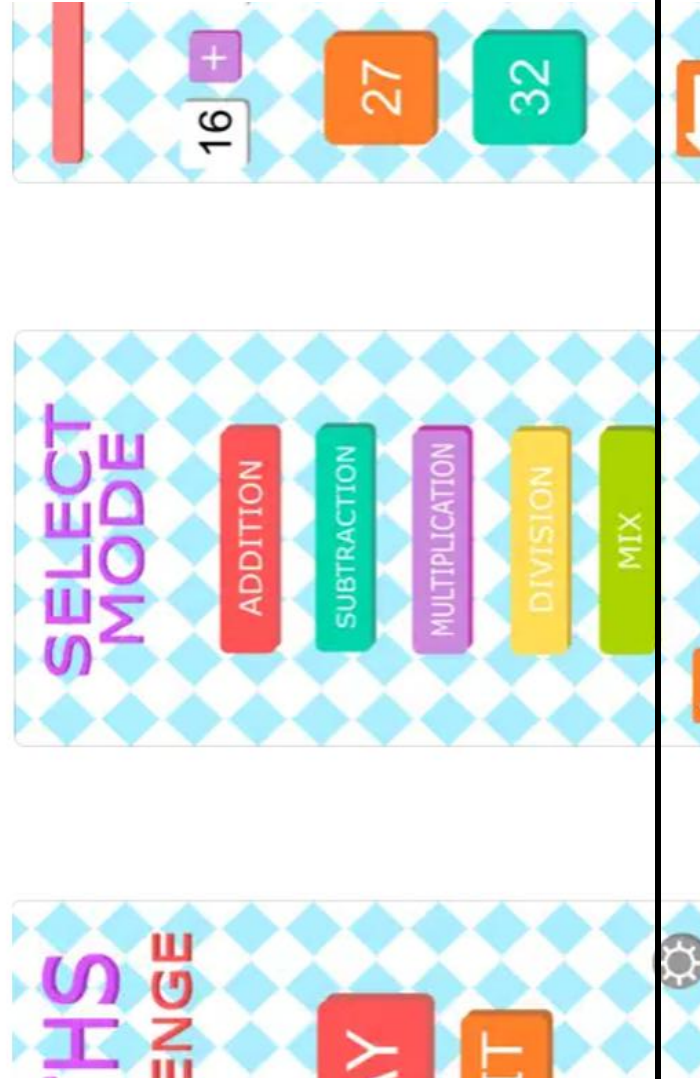
PROGRESS ■■■■■■■■■■■■■■■■■■■■■■

376 + 177 = (-217)

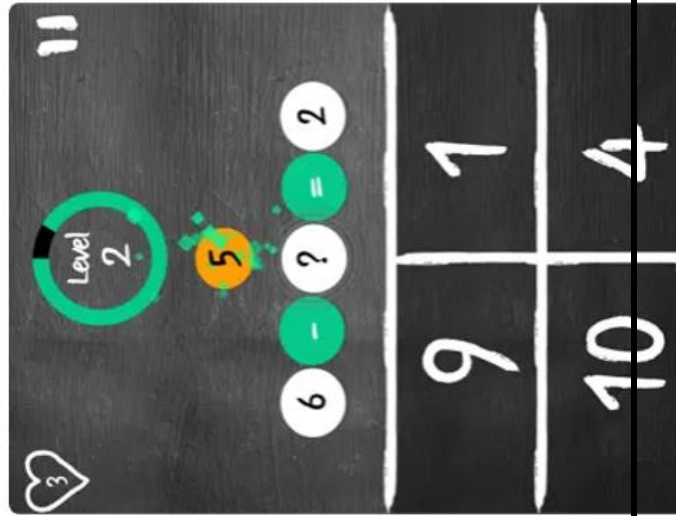
780 777 770

help me (3)

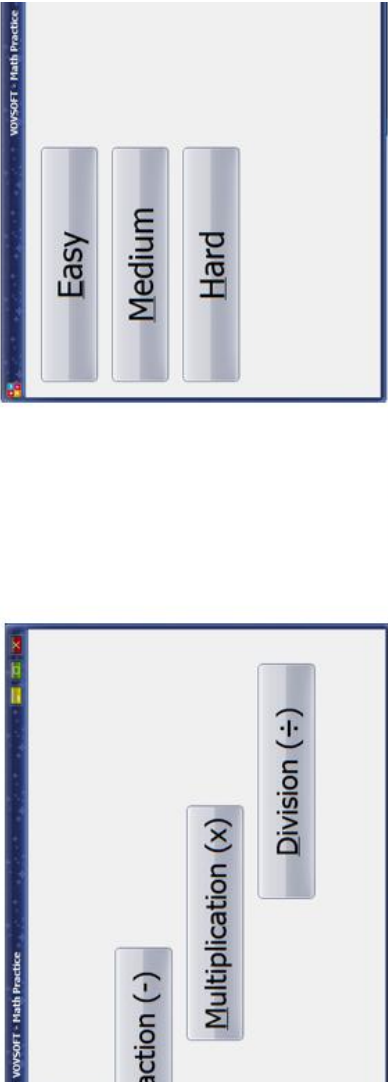
И Challenge (iOS)



И Vorkout (iOS)

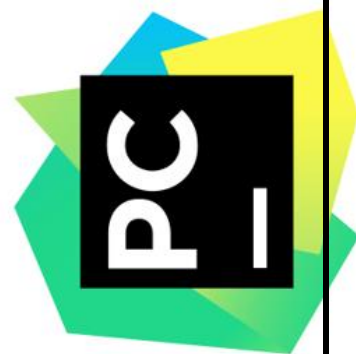


И Practice (Windows)



ТЕХНОЛОГИЙ

python



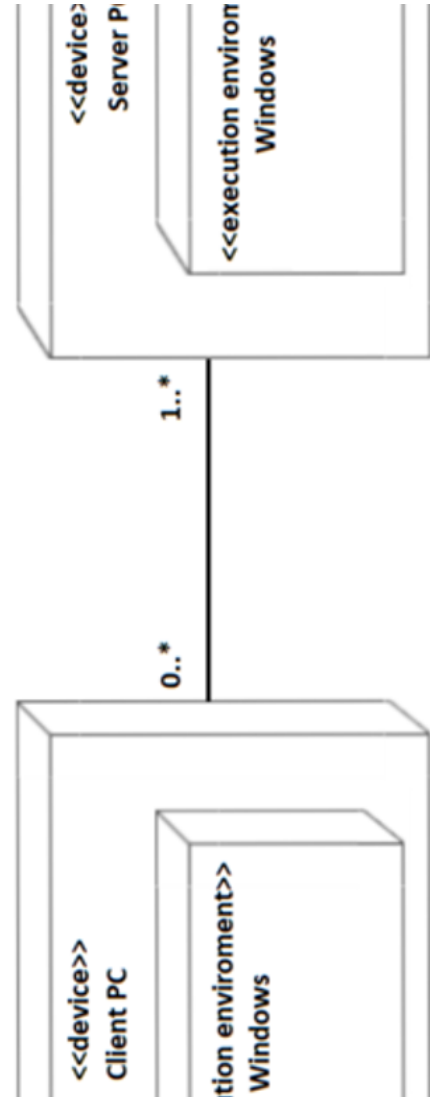
ІММА КЛАССОВ

y

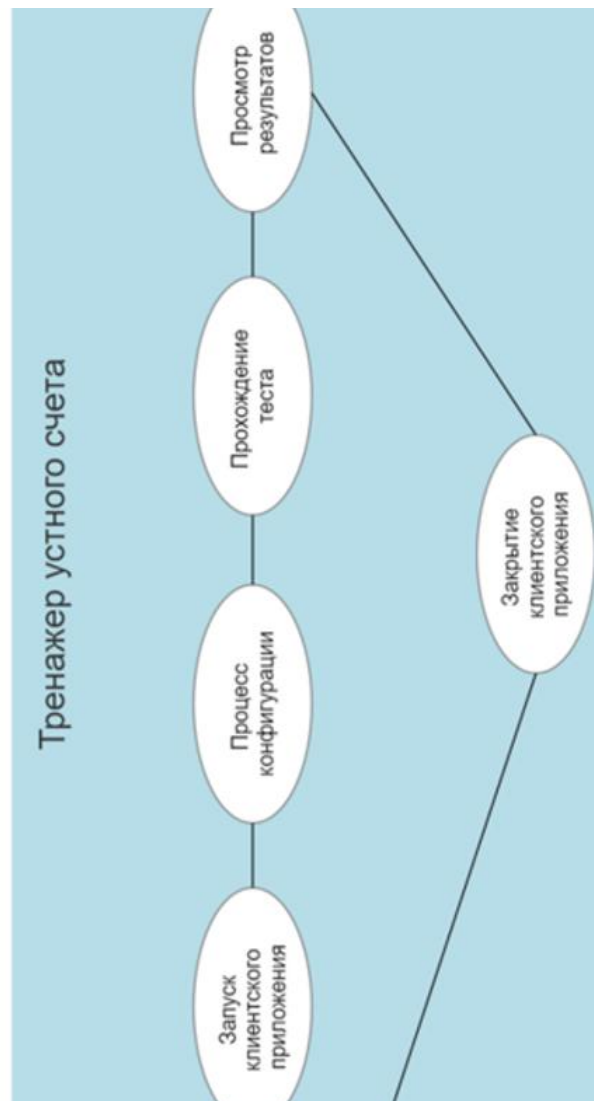
start_client.py
__init__(self)
initUI(self)
start_trainer(self)

start_server.py
__init__(self)
initUI(self)
start_server(self)
stop_server(self)
refresh_table(self)

МММ развертывания



Модель прецедентов



Server Control Panel

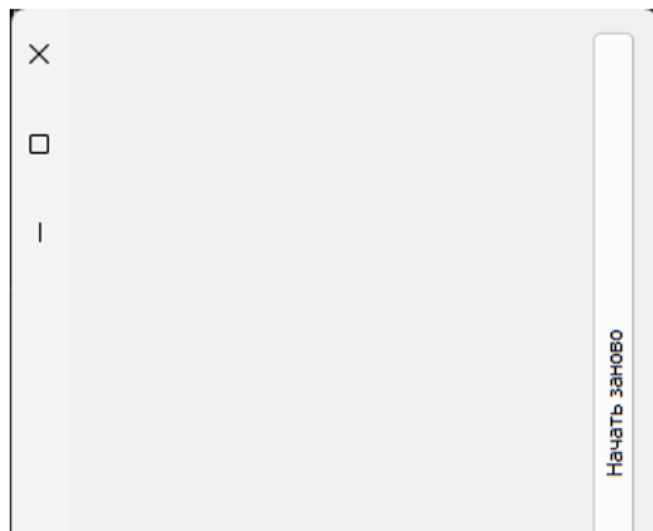
Сервер отключен

Запустить сервер

Отключить сервер

	Date	User	Timer	Tasks	Status
1	2023-06-07 ...	тест	1	5/5	Завершен
2	2023-06-16 ...	Андрей	1	0/5	Завершен
3	2023-06-16 ...	Ирина	1	5/5	Завершен
4	2023-06-16 ...	Владимир	1	2/5	Завершен
5	2023-06-23 ...	Олег	1	1/5	Завершен
6	2023-06-23 ...	Анубек	1	1/5	Завершен

страция работы



					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		81

Вывод

В ходе работы было разработано клиент-серверное приложение "Тренажер устного счета". Для клиентской части использовался графический интерфейс пользователя на языке Python и PyQt 5, а для серверной части - фреймворк Flask. В результате получилось полноценное приложение, которое можно использовать для тренировки устного счета и контроля успехов пользователей.

Ісibo за вниmани

					ПГУ1.090301.24.001 ПЗ	
Из	Лист	№ докум.	Подпись	Дата		83