**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**
**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**
**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**
Факультет компьютерных наук
Образовательная программа «Программная инженерия»

СОГЛАСОВАНО

Научный руководитель, старший преподаватель департамента больших данных и информационного поиска

_____ В. В. Куренков

«___» _____ 2025 г.

УТВЕРЖДЕНО

Академический руководитель образовательной программы «Программная инженерия», старший преподаватель департамента программной инженерии

_____ Н. А. Павлочев

«___» _____ 2025 г.

| Подп. и дата | |
|---|---|
| Инв.№ дубл. | |
| Взам. инв.№ | |
| Подп. и дата | |
| Инв.№ подл | |

# СИСТЕМА ПОСТРОЕНИЕ ГЕОМЕТРИЧЕСКИХ ЧЕРТЕЖЕЙ СО ВСТРОЕННЫМ ЯЗЫКОМ ПРОГРАММИРОВАНИЯ И ВОЗМОЖНОСТЬЮ УДАЛЕННОГО ПРОГРАММНОГО УПРАВЛЕНИЯ

**Пояснительная записка**

## ЛИСТ УТВЕРЖДЕНИЯ

**RU.17701729.12.17-01 81 01-1-ЛУ**

Исполнитель:

Студент группы БПИ233

_____ / С. А. Чубий /

«___» _____ 2025 г.

**Москва 2025**

УТВЕРЖДЕН

RU.17701729.12.17-01 81 01-1-ЛУ

# СИСТЕМА ПОСТРОЕНИЕ ГЕОМЕТРИЧЕСКИХ ЧЕРТЕЖЕЙ СО ВСТРОЕННЫМ ЯЗЫКОМ ПРОГРАММИРОВАНИЯ И ВОЗМОЖНОСТЬЮ УДАЛЕННОГО ПРОГРАММНОГО УПРАВЛЕНИЯ

**Пояснительная записка**

**RU.17701729.12.17-01 81 01-1**

**Листов 11**

**Москва 2025**

| Инв.№ подл | Подп. и дата | Взам. инв.№ | Инв.№ дубл. | Подп. и дата |
|---|---|---|---|---|
| | | | | |

## СОДЕРЖАНИЕ

# 1. ВВЕДЕНИЕ

## 1.1. Наименование программы

Наименование программы — **«Система построение геометрических чертежей со встроенным языком программирования и возможностью удаленного программного управления»**

Наименование программы на английском языке — **«Geometric Drawing System with a Built-in Programming Language and a Remote Program Control Capability»**

Краткое наименование программы — **«Geometrica»**

## 1.2. Краткая характеристика области применения программы

«Geometrica» — это десктоп-приложение, которое позволяет пользователю строить и изменять геометрические чертежи, используя графический интерфейс (GUI), интерфейс командной строки (CLI) или библиотеку для языка программирования Rust.

## 1.3. Документ(ы), на основании которых ведется разработка

Разработка ведётся на основании учебного плана подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденной академическим руководителем программы темы курсового проекта.

## 1.4. Наименование темы разработки

Наименование темы разработки: «Система построение геометрических чертежей со встроенным языком программирования и возможностью удаленного программного управления».

Условное обозначение темы разработки – «Geometrica».

## 2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

### 2.1. Функциональное назначение

Программный продукт позволяет пользователю строить и автоматически перестраивать геометрические чертежи. Производить расчеты на основе построенного чертежа.

### 2.2. Эксплуатационное назначение

Продукт состоит из трех исполняемых файлов для ОС Linux и одной библиотеки для языка программирования Rust:

- Сервера;
- Графического (GUI) клиента;
- Клиента командной строки (CLI);
- Клиента-библиотеки (lib).

Целевой аудиторией являются:

- Школьники, изучающие геометрию (5–11 классы);
- Школьные учителя, преподающие геометрию (5–11 класс);
- Студенты ВУЗов, изучающие вычислительную геометрию;
- Преподаватели ВУЗов, преподающие вычислительную геометрию.

Программный продукт может быть использован на обычных занятиях (например, для демонстрации тех или иных теорем, совместного решения задач), для проведения проверочных работ, выполнения домашних заданий, отладки геометрических программ, самостоятельного решения геометрических задач.

CLI- и lib-клиенты в первую очередь нацелены на студентов и преподавателей ВУЗов. GUI клиент будет интересен всем представителям целевой аудитории.

# 3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

## 3.1. Постановка задачи на разработку программы

Цель работы — реализовать все компоненты описанной выше системы, а именно:

- Сервер;
- Графический (GUI) клиент;
- Клиент командной строки (CLI);
- Клиент-библиотеку (lib).

## 3.2. Описание алгоритма и функционирования программы

### 3.2.1. Краткое описание крейтов[1] и взаимодействия между ними

Программный продукт разделен на несколько крейтов:

- Клиентская сторона:
  - **GUI** — графический клиент. Поставляется пользователю, как часть программного продукта.
  - **CLI** — клиент командной строки. Поставляется пользователю, как часть программного продукта.
  - **Client** — клиент-библиотека. Поставляется пользователю, как часть программного продукта, а также используется в реализации крейтов **CLI** и **GUI**.
  - **Parser** отвечает за парсинг встроенного языка программирования. Является частью внутренней реализации, пользователю **не** поставляется.
- Серверная сторона:
  - **Server** — сервер. Поставляется пользователю, как часть программного продукта.
  - **Executor** выполняет основную часть вычислений. Является частью внутренней реализации, пользователю **не** поставляется.
- Общее:
  - **Types** содержит объявления структур, используемых как на серверной, так и на клиентской стороне. Является частью внутренней реализации, пользователю **не** поставляется.

Более подробное описание некоторых из крейтов содержится в последующих пунктах.

Отношения между крейтами можно увидеть на Рис. 1.

### 3.2.2. Крейт Types

[1] **Крейт (crate)** единица компиляции в Rust. Ближайший аналог в других языках программирования — пакет.

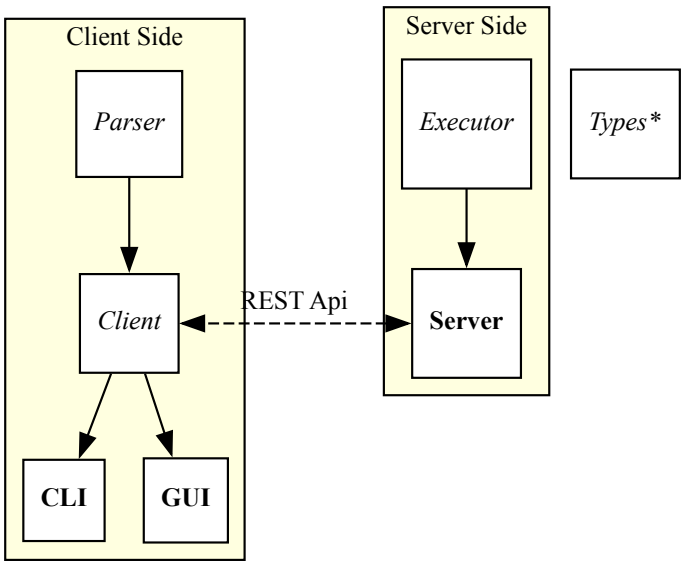


Рис. 1. Взаимодействие между крейтами.

Все крейты также зависят от крейта Types. Эти связи не изображены в целях упрощения рисунка.

Бинарные крейты (binary crates) выделены **жирным**, крейты-библиотеки (library crates) выделены *курсивом*.

Обычными стрелками показаны библиотечные зависимости, пунктирными — зависимости других типов.

Крейт Types содержит в себе объявления некоторых струкур (и связанных с ними функций и методов), используемых как серверной, так и клиентской сторонами. Среди них:

- структуры, описывающие типы встроенного языка программирования: `Value`, `ValueType`, `Pt`, `Line` и д.р.;
- структуры, описывающие синтаксис встроенного языка программирования: `Statement`, `Definition`, `Expr` и д.р.;
- структуры, описывающие REST API: `items::get_all::Request`, `items::get_all::Response`, `set::Request`, `set::Response` и д.р.

Предполагается, что этот крейт должен быть настолько компактным, насколько это возможно. С этой целью в нем применяется техника условной компиляции (conditional compilation), благодаря которой часть функционала крейта можно не компилировать, если в ней нет необходимости, что

позволяет уменьшить размер результирующей программы, путем отключения ненужного кода и ненужных зависимостей.

### 3.2.3. Крейт Parser

Крейт Parser реализует логику, связанную с парсингом встроенного языка программирования. Он предоставляет пользователю несколько методов для парсинга тех или иных языковых конструкций: `script`, `expr`, `statement` и д.р., а также трейт[2] `ParseInto<T>` с аналогичным назначением.

Для парсинга используется библиотека `peg`.

### 3.2.4. Крейт Client

Крейт Client реализует логику клиентской стороны приложения.

Основной структурой данного крейта является `Client`. Он предоставляется набор методов (`command`, `eval`, `get_all_items` и д.р.) для выполнения действий над чертежом. Некоторые из этих методов являются легкими обертками над соответствующими методами REST API, описанными в крейте Types. Некоторые имеют более сложную логику; в частности, интересно рассмотреть, как происходит исполнение кода на встроенном языке программирования (методы `exec` и `exec_one`; смотри также Рис. 2):

- Из строкового представления код парсится в `Vec<Statement>` в случае `exec` и в `Statement` в случае `exec_one`. Дальнейшие шаги описаны для каждого `Statement` отдельно.
- Если очередное выражение (`Statement`) является объявлением (`Definition`), то запрос на обработку сразу отправляется на сервер (смотри Раздел 3.2.7.2).
- Иначе, если очередное выражение (`Statement`) является командой (`Command`), то оно обрабатывается особым образом. Например, команда `set!` изменяет значение вершины, `eval!` вычисляет значение выражения и т.д.

### 3.2.5. Крейт GUI

Крейт GUI содержит реализацию графического клиента.

Он использует библиотеку `iced` для отрисовки интерфейса и крейт Client для взаимодействия с сервером.

### 3.2.6. Крейт CLI

Крейт CLI содержит реализацию клиента командной строки.

Он может работать в нескольких режимах:

[2]Ближайшим аналогом **трейта (trait)** из Rust в других языка программирования является интерфейс.

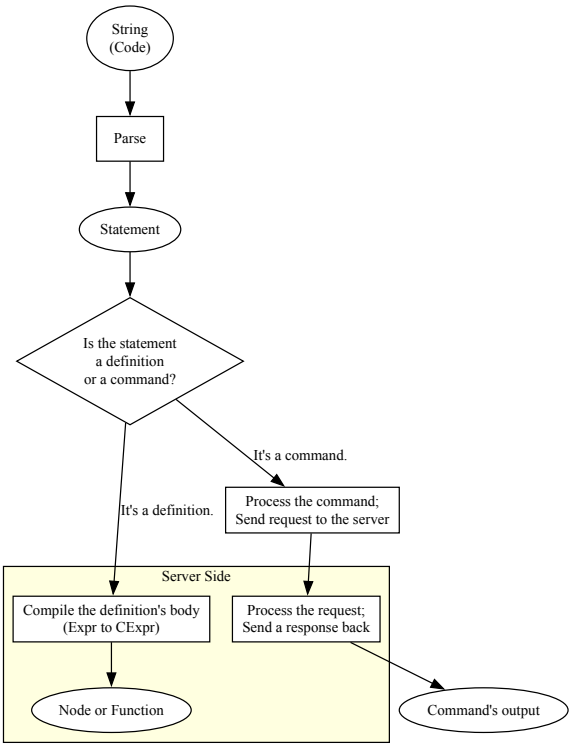


Рис. 2. Процесс исполнения кода.

В кругах обозначены состояния, в прямоугольниках — действия, в ромбах — условия.

Действия в желтом прямоугольнике происходят на стороне сервера, остальные — на стороне клиента.

- **Скриптовый режим** будет запущен, если передать имя файла, в качестве аргумента командной строки. Тогда этот файл будет обработан, как скрипт на встроенном языке программирования. Результат выполнения скрипта будет напечатан на стандартный вывод.
- **Режим стандартного ввода** будет запущен, если передавать данные на стандартный ввод через `pipe`. Тогда входные данные будут обработаны, как код на встроенном языке программирования. Результат выполнения скрипта будет напечатан на стандартный вывод.
- **Интерактивный режим** будет запущен в остальных случаях (то есть, если не передавать аргументов командной строки и не использовать `pipe`). В этом режиме пользователь может интерактивно вводить код на встроенном языке программирования и сразу получать результат его выполнения.

### 3.2.7. Крейт Executor

Крейт Executor выполняет основную часть вычислений, а именно, работает с выражениями на встроенном языке программирования и поддерживает нынешнее состояние чертежа.

#### 3.2.7.1. Поддержка состояния

Для хранения состояния чертежа служит структура `Node` (вершина). Вершина может содержать либо некоторое фиксированное значение; либо функцию и набор её аргументов (список других вершин), на основе которых это значение можно вычислить. Вершина также хранит список вершин, которые зависят от неё.

Если при выполнении команды значение некоторой вершины было изменено, то значения всех вершин из её списка зависимых, также пересчитываются; процесс пересчета продолжается рекурсивно.

#### 3.2.7.2. Работа с выражениями

Если команды (`Command`) во многом обрабатываются клиентом, то объявления полностью обрабатываются сервером. Так, если некоторое выражение (`Statement`) является объявлением функции (`FunctionDefinition`) или значения (`ValueDefinition`), то в списке функций или вершин соответственно, создается новый элемент. Тело функции или значение выражения, представленное в виде `Expr`, обрабатывается описанным ниже образом.

`Expr` компилируется в `CExpr` (от Compiled `Expr`). На этом этапе происходит разрешение имен переменных и функций, происходят упрощение структуры выражения. Так, если структура `Expr` очень похожа на синтаксис встроенного языка программирования и удобна для ввода/ вывода (превращения `Expr` в строку и обратно), то структура `CExpr` оптимизирована для простоты вычисления.

Вычисление значения выражений происходит при пересчете дерева вершин, а также при явном вызове команды `eval!`.

### 3.2.8. Крейт Server

Крейт Server является фасадом для крейта Executor.

Он реализует API, описанное в крейте Types. Server принимает запросы от крейта Client, использует Executor для их обработки, после чего отправляет результат обратно крейту Client.

Server может быть запущен либо вручную, либо одним из клиентов.

Для реализации HTTP сервера используется библиотека `axum`, для [де]серилизации запросов и ответов (в формате JSON) используется библиотека `serde`.

## 3.3. Описание организации входных данных

- Взаимодействие с GUI клиентом происходит с использованием клавиатуры, мыши (ввод) и монитора (вывод).
- Взаимодействие с CLI клиентом происходит с использованием клавиатуры (ввод) и монитора (вывод).
- Взаимодействие с сервером происходит через один из клиентов по REST API. API использует протокол HTTP для передачи данных в формате JSON.
- Взаимодействие с lib-клиентом происходит посредством подключения его в качестве библиотеки к программе на ЯП Rust, что можно сделать с помощью пакетного менеджера `Cargo`.

### 3.4. Описание состава технических и программных средств

Для максимально качественной работы системы установлены следующие требования:

- Операционная система Linux[3]
- 4Гб оперативной памяти
- 128Гб памяти на HDD или SSD
- Мышь, клавиатура, монитор

[3]Разработка и тестирование проводились на `NixOS Unstable (rev: 42a1c96)`.

## 4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

### 4.1. Предполагаемая потребность

Система будет полезна школьникам (5–11 классы), студентам и преподавателям на уроках геометрии и вычислительной геометрии. Программный продукт может быть использован на обычных занятиях (например, для демонстрации тех или иных теорем, совместного решения задач), для проведения проверочных работ, выполнения домашних заданий, отладки геометрических программ, самостоятельного решения геометрических задач.

### 4.2. Сравнение с аналогичными решениями

| | **Geometrica**[4] | GeoGebra[5] | Desmos Geometry[6] | Живая Математика[7] | MathKit[8] |
|---|---|---|---|---|---|
| Программа бесплатна | + | + | + | - | + |
| Есть оффлайн версия | + | + | - | + | + |
| Возможно создание макросов | ? | - | - | + | + |
| Есть встроенный ЯП | + | ? | ? | - | - |
| Есть библиотека для существующего ЯП | + | + | + | - | - |
| Есть REST API | + | - | - | - | - |
| Возможная работа из командной строки | + | - | - | - | - |
| Возможны косметические преобразования[9] | - | + | + | + | + |

Таблица 1. Сравнение функциональных характеристик с аналогами

«+» — функция имеется, «-» — функция отсутствует, «?» — функция частично присутствует/ имеются значительные ограничения.

Таблица 1 показывает сравнение разрабатываемого продукта (**Geometrica**) с некоторыми аналогами. Разъяснения некоторых пунктов таблицы приведено ниже.

[4]Данная система.
[5]https://www.geogebra.org/geometry
[6]https://www.desmos.com/geometry
[7]https://www.int-edu.ru/content/rusticus-0
[8]https://obr.1c.ru/mathkit/
[9]То есть можно менять цвета различных объектов, ширину прямых и т.д.

Создание макросов в данной системе не предусмотрено, однако большую часть их функционала можно заменить, используя скрипты через CLI-клиент.

И GeoGebra, и Desmos имеют собственные встроенные языки программирования, однако в они достаточно ограничены. Во-первых, ограничен синтаксис: так в обоих языках нельзя комментировать код, создавать локальные переменные (отсутствует конструкция `let`), а в GeoGebra нельзя объявлять новые функции. Во-вторых, ограничены возможности по вводу скриптов, а именно: весь скрипт отображается в одну строку в небольшой ячейке, что усложняет работу с большими скриптами; один скрипт может создать только один объект или семейство объектов (то есть, например, объявить десять разных точек в одном скрипте нельзя). Наконец, в языках отсутствуют команды, позволяющие менять структуру чертежа, такие как: `delete!`, `set!` и д.р.

### 4.2.1. Результат сравнения

Из таблицы видно, что проект имеет преимущества перед существующими аналогами, особенно в области программирования (встроенный ЯП, REST API, работа из командной строки), что будет особенно важно при отладке геометрических программ. Это показывает, что в разрабатываемом программном продукте есть смысл.

При этом, недостатки также присутствуют. Это означает, что в будущем проект можно дорабатывать, в тех направлениях, в которых он сейчас проигрывает конкурентам.