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

СОГЛАСОВАНО	УТВЕРЖДЕНО
Научный руководитель, старший преподаватель департамента больших данных и информационного поиска	Академический руководитель образовательной программы «Программная инженерия», старший преподаватель департамента программной инженерии
_____ В. В. Куренков	_____ Н. А. Павлочев
«___» _____ 2025 г.	«___» _____ 2025 г.

Подп. и дата	
Инв.№ дубл.	
Взам. инв.№	
Подп. и дата	
Инв.№ подп	

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

Техническое задание

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

Техническое задание

RU.17701729.12.17-01 81 01-1

Листов 8

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

Москва 2025

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.

Продукт будет полезен как для проведения занятий, так и для индивидуальной работы.

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.

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.

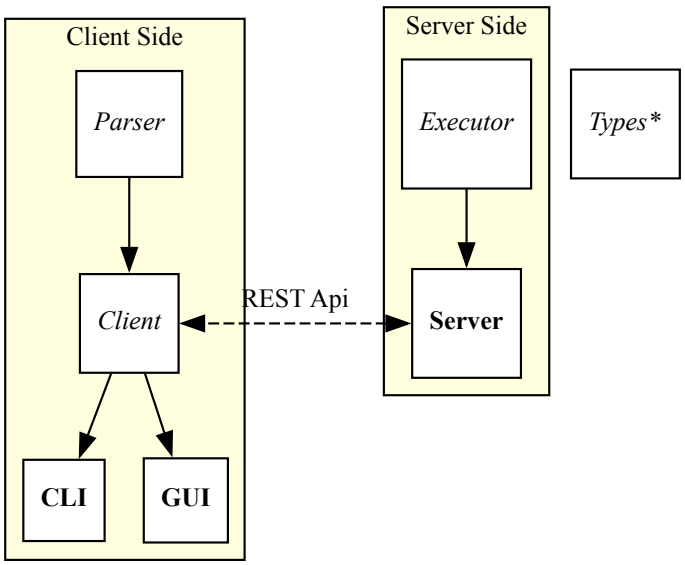

Рис. 1. Взаимодействие между крейтами.

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 `api`: `api::items::get_all::Request`, `api::items::get_all::Response`, `api::set::Request`, `api::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 позволит

уменьшить размер результирующего файла, 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 взаимодействия с сервером.

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:

- **Скриптовый режим** будет запущен, если передать имя некоторого 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, если в остальных случаях (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

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

Крейт Executor выполняет основную часть вычислений, а именно, исполняет код на встроенном языке программирования и поддерживает нынешнее состояние чертежа.

Для хранения 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, — это упрощает процесс пересчета.

Процесс выполнения кода происходит следующим образом:

- Из строкового представления код парсится в `Vec<Statement>`. На этом этапе структурные преобразования минимальны. Структура `Statement` очень похожа на синтаксис встроенного языка.
- Далее возможно два случая:
  - Если очередное выражение (`Statement`) является командой (`Command`; то есть оно может менять структуру дерева), то оно обрабатывается особым образом. Например, команда `set` изменяет значение вершины, `eval` вычисляет значение выражения и т.д.
  - Иначе, если очередное выражение (`Statement`) является объявлением (`FunctionDefinition` или `ValueDefinition`), то в списке функций или вершин соответственно, создается новый элемент. Тело функции или значение выражения, представленное в виде `Expr`, обрабатывается описанным ниже образом (парсинг уже был выполнен, поэтому первый пункт в списке пропускается).

Процесс обработки `Expr` (также смотри):

- Из строкового представления выражение парсится в `Expr`. Сказанное выше про `Statement` верно и для `Expr`: структура `Expr` максимально приближена к синтаксису встроенного языка программирования. Это полезно, при выводе `Expr` на экран (превращении `Expr` обратно в строку).
- `Expr` компилируется в `CExpr` (от Compiled `Expr`). На этом этапе происходит разрешение имен переменных и функций, происходят упрощение структуры выражения. Так, если структура `Expr` похожа на синтаксис языка и удобна для ввода/ вывода, то структура `CExpr` оптимизирована для простоты вычислений.

3.3. Описание и обоснование выбора метода организации входных данных

3.4. Описание и обоснование выбора состава технических и программных средств

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Предполагаемая потребность

4.2. Сравнение с аналогичными решениями