Московский авиационный институт

(Национальный исследовательский университет)

**Лабораторные работы 1 - 10**

**Архитектура программной среды**

**Лаборатория 1.** Интегрированные среды разработки и их базовые отличия.

**Лаборатория 2.** Потоковая диаграмма GitHub и Архитектура среды.

**Лабораторная 3.** Разработка потоковых диаграмм их реализация в среде VS Code.

**Лабораторная 4.** Создание профайла VS Code для работы в среде Python. Виды профайлов. Их использование при распределенной разработки. Структура профайла. GitHub Copilot. UML (Unified Model Language) - диаграмма взаимодействий.

**Лабораторная 5.** Создание ветвей в GitHub для распределенной разработки и их слияние из среды VS Code.

**Лабораторная 6.**

**Лабораторная 7.**

**Лабораторная 8.**

**Лабораторная 9.**

**Лабораторная 10.**

**\*Laboratory 11.**

**\*Laboratory 12.**

|  |  |  |
| --- | --- | --- |
| Глушатов И. С. | подпись: | \_\_\_\_\_\_ |
| Семенов А. С. | подпись: | \_\_\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_ | 2024 |

**Лабораторная 1.**  Интегрированные среды разработки и их базовые отличия

IDE – интегрированная среда разработки – комплекс программных средств, которые используется для редактирования кода, дебага, тестирования, структурирования проектов, использования дополнительных плагинов для ускорения и удобства разработки.

Основными компонентами являются:

1. Текстовый редактор
2. Компилятор/Интерпретатор
3. Отладчик
4. Система управления версиями

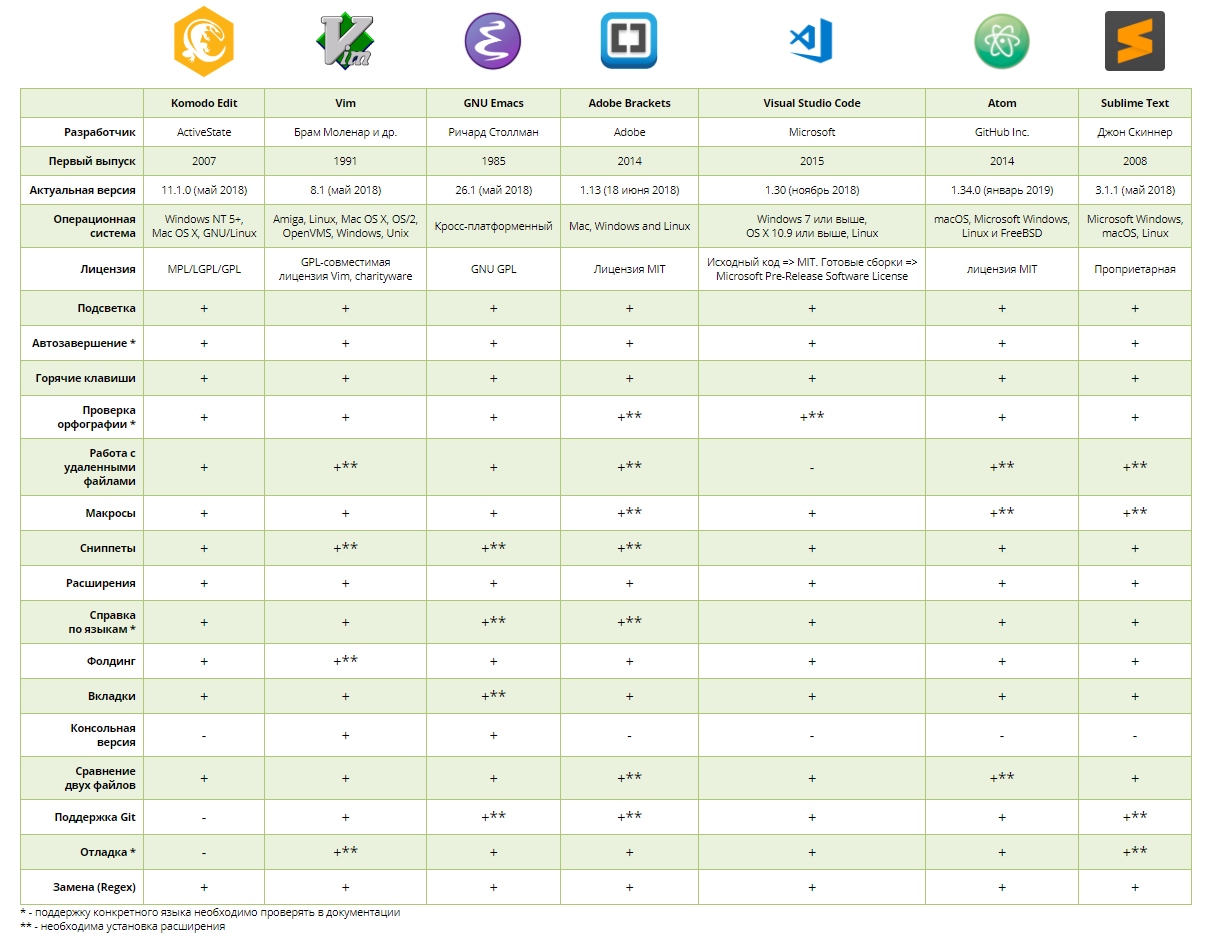
Практически все IDE поддерживают плагины.

Плагины – удобное решение, помогающее гибко подстраивать работу IDE под свои нужды. Если в редакторе отсутствует поддержка какого-либо фреймворка или языка, то это можно исправить установкой необходимых сниппетов и плагинов для поддержки синтаксиса. Если не нравится, как автоматически форматируется код и уже исчерпались возможности настроек IDE, то можно задействовать десятки различных плагинов, реализующих альтернативу встроенному форматированию.

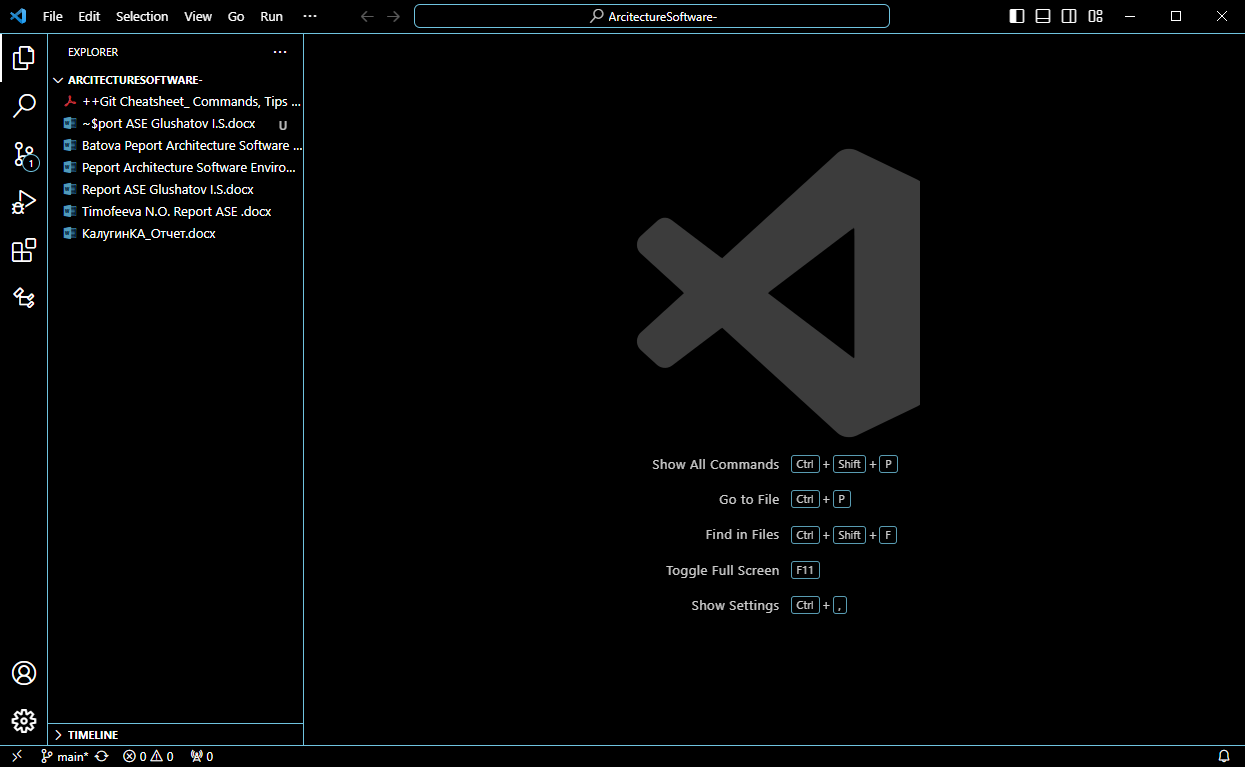
Преимущества использования IDE в разработке ПО:

1. Экономия времени – IDE помогает автоматически дописывать некоторые синтаксические конструкции; подсвечивает синтаксис, что позволяет быстрее ориентироваться в тексте кода; подсвечивает некоторые синтаксические ошибки, что позволяет не тратить время на компиляцию и многое другое.
2. Централизованность проекта – IDE позволяет собирать проект в цельный программный продукт; настраивать пайплайн разработки-сборки-развертывания проекта; система контроля версий позволяет контролировать поток разработки проекта от начала и до конца, предотвращая потерю данных и изменений в большом масштабе.

Ниже приведена таблица со сравнением возможностей различных IDE:



Visual Studio Code – среда разработки, которая построена на основе открытых источников, работает везде и имеет встроенные функции для запуска и отладки кода, а также расширения для установки новых языков, тем, отладчиков и подключения к дополнительным сервисам.



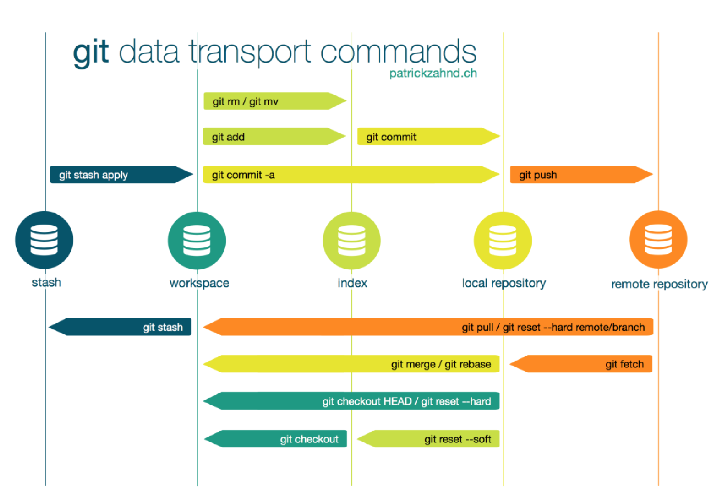
Данная IDE очень гибкая и позволяет полностью настроить под себя и свои задачи. Это можно делать путём установки расширений.

Расширения осуществляют подсветку кода, инструментами для взаимодействия с другими программами и сервисами через интернет.

Также в данной среде предусмотрен функционал быстрого поиска и настройки проекта с помощью строки.

**Лабораторная 2.** Потоковая диаграмма GitHub и Архитектура среды.

Ниже представлена общая потоковая диаграмма при работе над проектом как в локальном, так и в удаленном репозитории.



В потоковой диаграмме GitHub выделяются следующие элементы:

1. Workspace – рабочее пространство со всеми файлами и директориями над которыми мы работаем
2. Index – промежуточное хранилище измененных файлов перед фиксацией изменений, объединяет изменения. Не все файлы могут быть полезными(кэш, метаинформация, временные файлы, мусор), поэтому данная область позволяет зафиксировать только важные изменения.
3. Local repository – локальный репозиторий, куда коммитятся изменения, хранятся временные ветки и другая метаинформация.
4. Remote repository – централизованный удаленный репозиторий, где хранятся версии проекта, позволяет делиться изменениями между несколькими компьютерами.

Важнейшим компонентом потоковой диаграммы являются branches (ветки). Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов. В больших проектах часто используется более комплексная структура:

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

1. Master – основная ветка, которая отражает проект на стадии продакшена.
2. Develop – ветка разработки, которая указателем на HEAD отражает последние внесенные изменения, готовые к выкладке в продакшен
3. Release – поддерживает подготовку нового производственного выпуска, позволяя в последнюю минуту исправить незначительные ошибки и подготовить метаданные к выпуску.
4. Feature – ветки, в которых проводится непосредственно разработка проекта большую часть времени. Она отводится от ветки develop и сливается с ней после внесения изменений.
5. Hotfixes – ветка для исправления критических багов, попавших в продакшен. Она должна вливаться в master и develop, чтобы избежать ошибок в следующем выпуске.

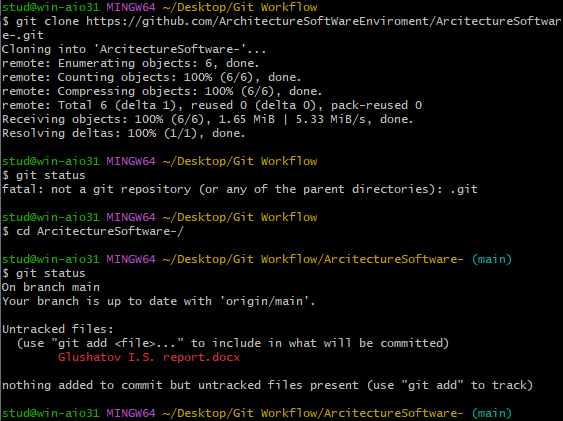
Потоковая диаграмма для типичного рабочего процесса разработки ПО:

Изображение выглядит как снимок экрана, текст, Параллельный, линия

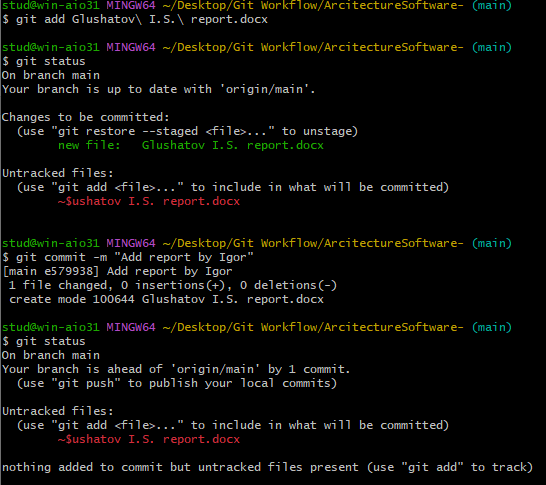
Автоматически созданное описание

Ниже будет показан пример прохождения вышеуказанных этапов.

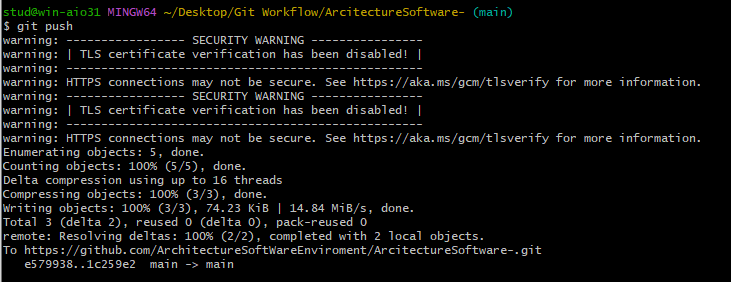
После клонирования репозитория и добавления нового файла:



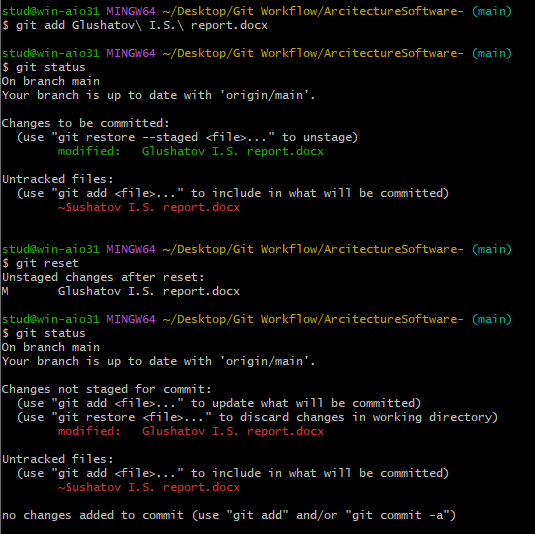
Состояние после команды add и commit:



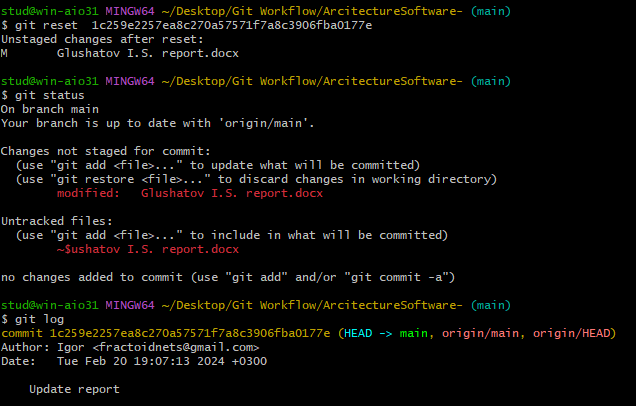
После команды push и отправления изменений на удаленный репозиторий:



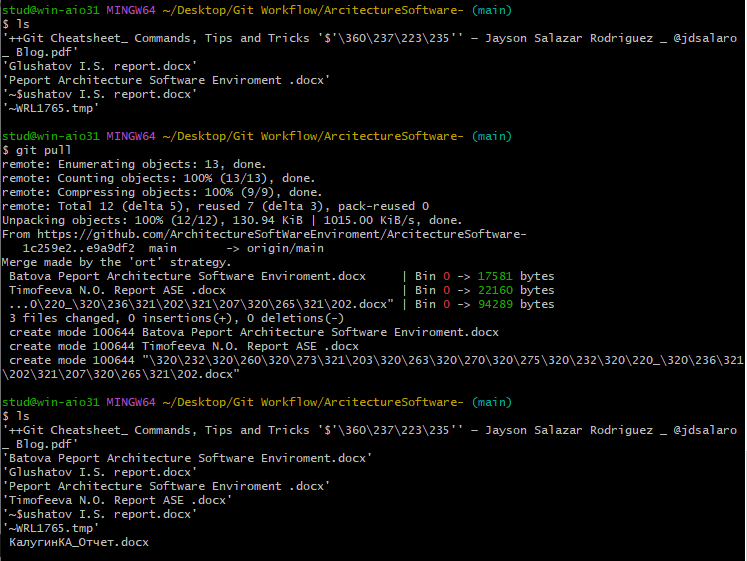
После команды add выполняем команду reset и отменяем добавление в индекс:



После команды git reset [коммит] состояние восстанавливается до указанного коммита:



После изменений на удаленном репозитории делаем команду pull:

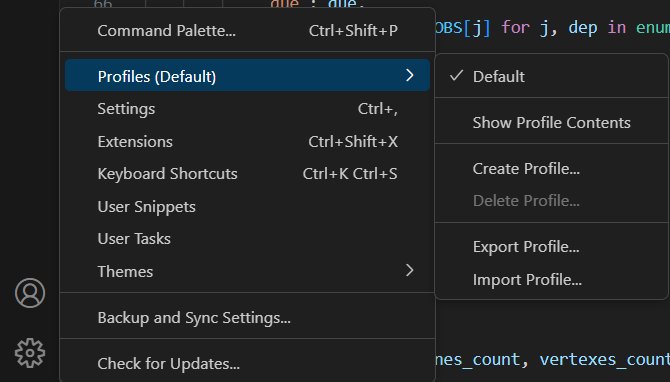


**Лабораторная 3.** Создание профайла VS Code для работы в среде Python. Виды профайлов. Их использование при распределенной разработки. Структура профайла.

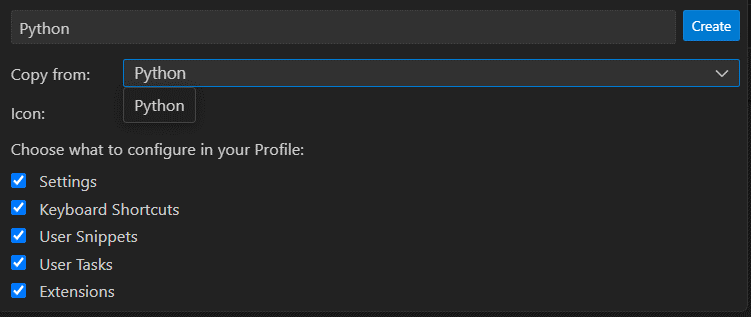
Профайл в VS Code – это набор множества настроек VS Code, плагинов и UI, который позволяет организовывать процесс разработки и делиться настройками с другими людьми.

Также бывают частичные профайлы, позволяющие менять только определенные настройки и наследовать остальные от базового профиля.

Создание профайла:

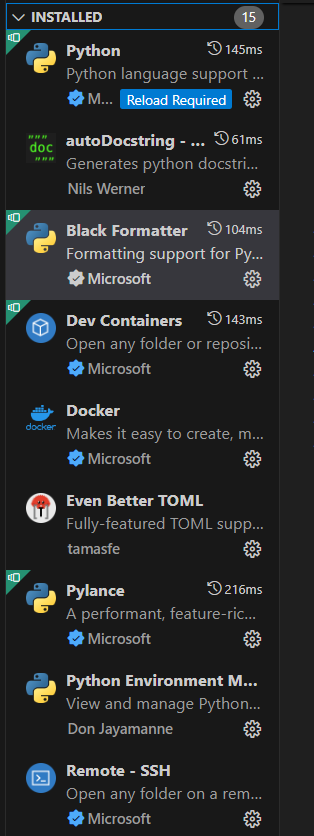


Задаем название для создаваемого профиля и выбираем корректируемые параметры. Надо ввести имя для профайла, всё остальное можно оставить по-умолчанию.



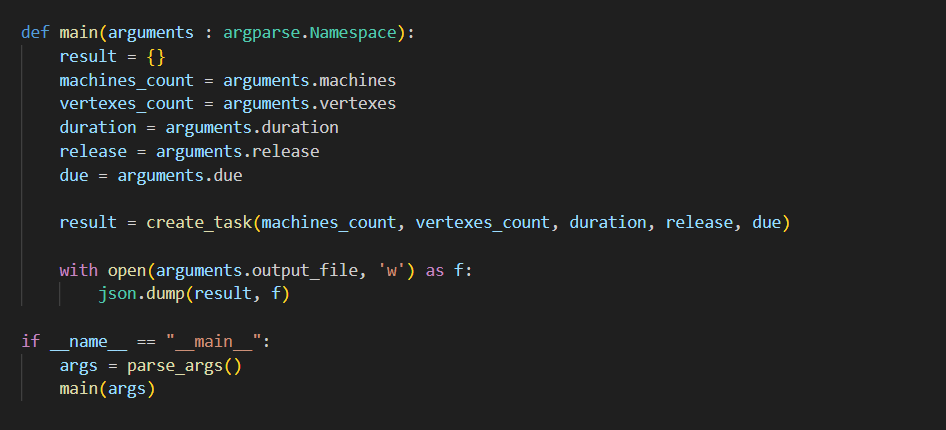
После этого видим, что в созданном профиле есть базовая часть и настройка расширений

Устанавливаем расширения для работы с Python проектом

 Изображение выглядит как текст, снимок экрана, программное обеспечение

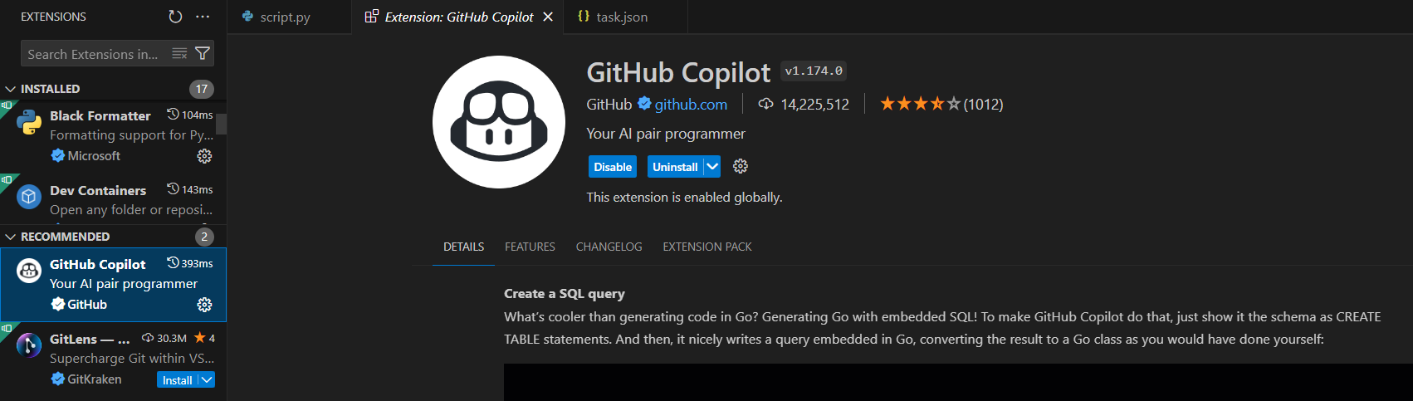
Автоматически созданное описание

На написанной программе можно проверить, что всё работает.



Как видим расширение работает, подсветка синтаксиса есть. Таким образом, можно сделать вывод, что создание профайлов это, по сути дела, гибкая настройка IDE для конкретного проекта или направления. Т. е. можно создать проект для веб разработки на C++, либо профиль для фронтенд разработки. При переходе на свой профиль разработчик будет чётко понимать, что у него есть всё необходимое для выполнения задачи.

Теперь установим Github Copilot.



GitHub Copilot — ваш искусственный напарник-программист. Система в реальном времени анализирует код, который пишет пользователь, а затем предлагает варианты его продолжения в виде отдельных фрагментов или целых функций. Пример работы GitHub Copilot представлен ниже.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Лабораторная 4.** Разработка потоковых диаграмм. UML (Unified Model Language) - диаграмма взаимодействий.

Потоковая диаграмма – это диаграмма, наглядно отображающая течение информации в пределах процесса или системы. Основные её компоненты: **process, data flow, warehouse, terminator**.

UML – это унифицированный язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

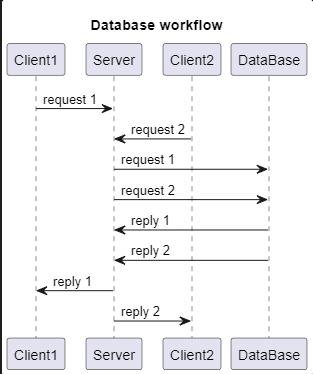
Диаграммы взаимодействия в UML представляют собой модели, которые необходимы для описания поведения взаимодействующих групп объектов. Разделяются на: коммуникации, обзора взаимодействия, последовательности, синхронизации.

Диаграммы взаимодействия используются, когда мы хотим понять поток сообщений и структурную организацию. Поток сообщений означает последовательность передачи управления от одного объекта к другому. Структурная организация означает визуальную организацию элементов в системе.

Можно использовать диаграммы взаимодействия:

1. Моделировать поток управления по временной последовательности.
2. Для моделирования потока управления структурными организациями.
3. Для перспективного проектирования.
4. Для обратного проектирования.

Теперь попробуем нарисовать UML диаграмму клиент-серверной архитектуры программы на python с базой данных с помощью плагина PlantUML в VS Code.



Код диаграммы:

@startuml

title Database workflow

Client1 -> Server : request 1

Client2 -> Server : request 2

Server -> DataBase : request 1

Server -> DataBase : request 2

DataBase -> Server : reply 1

DataBase -> Server : reply 2

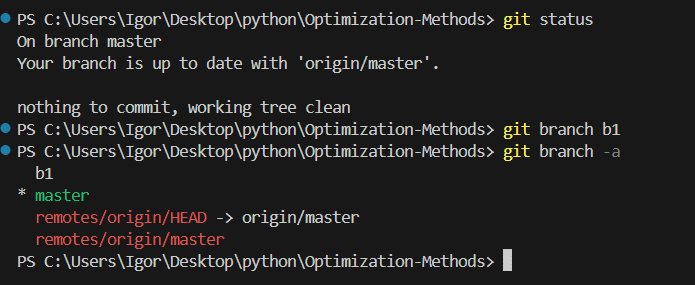
Server -> Client1 : reply 1

Server -> Client2 : reply 2

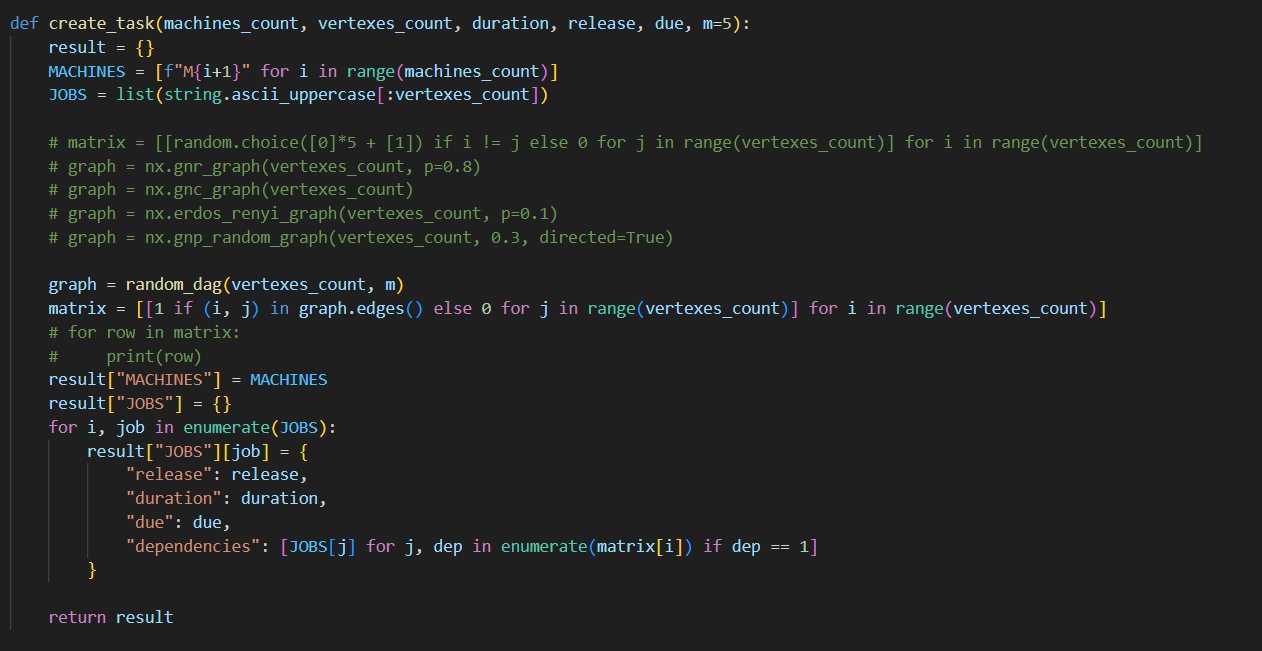
@enduml

**Лабораторная 5**. Создание ветвей в GitHub для распределенной разработки и их слияние из среды VS Code.

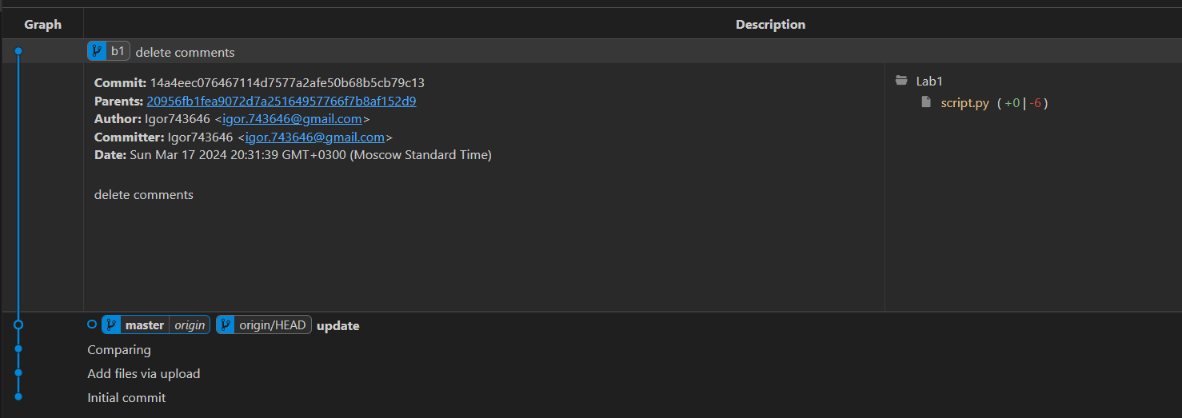
В прошлой лабораторной работе мы написали программу, которая создаем базу данный. Теперь допишем код так, чтобы была возможность добавлять данные в таблицу. Для этого создаем новую ветку и переходим в неё.



Исходный код в ветке **b1** функции **create\_task**:

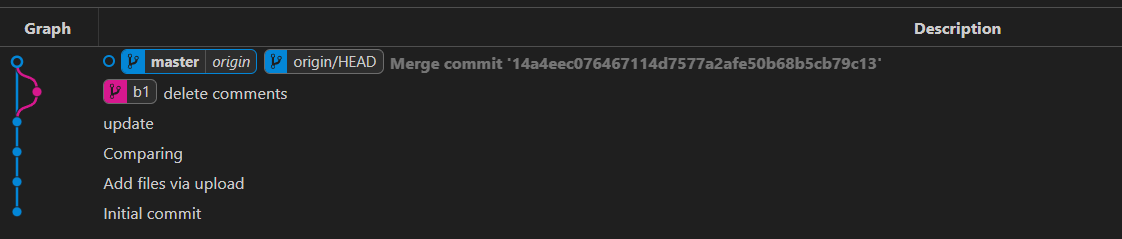


В VS Code увидим изменения после перемещения в мастер ветку

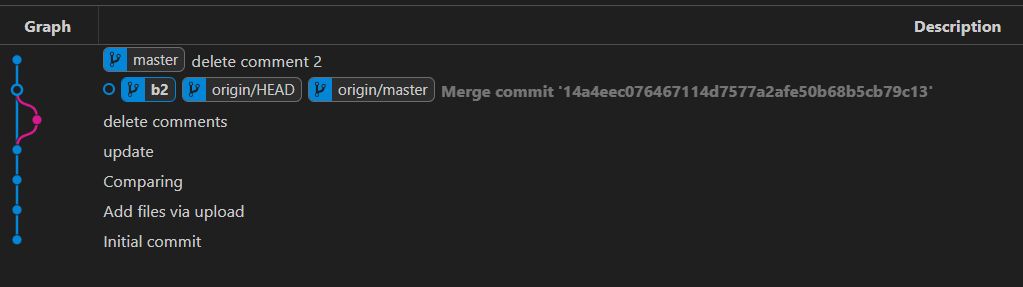


Далее можем выполнить слияние созданной ветки с мастер.

Результат слияния веток



Теперь выполним смещения статуса мастер ветки на новую ветку. Для этого создаем новую ветку и изменяем немного исходный код.



И выполняем функцию rebase branch. Результат выполнения операции rebase:

