Московский авиационный институт

(Национальный исследовательский университет)

**Лабораторные работы 1 - 10**

**Архитектура программной среды**

.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Алавиев О.А./

(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_/Семенов АС./

(Подпись)

Дата : \_\_ 2024

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

**1.**Интегри́рованная среда́ разрабо́тки, ИСP ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) integrated development environment — IDE)- комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

**2.** Среда разработки включает в себя:

текстовый редактор,

Транслятор (компилятор и/или интерпретатор),

средства автоматизации сборки

дебаггер

загрузчик программы

**3.** Отличие IDE от редактора кода

Редактор кода — это простая легковесная программа, которая похожа на текстовый редактор, только больше адаптированная под написание кода. Она сохраняет проекты в нужном расширении, подсвечивает синтаксис и автоматически проверяет отступы в коде.

IDE — более масштабный инструмент, программный комплекс. Помимо редактора в нем есть собственная консоль, инструменты для запуска, тестирования и отладки кода. Через IDE можно создавать крупные проекты, подключать систему контроля версий (Git). Некоторые IDE позволяют устанавливать внутри себя новые фреймворки и библиотеки, не заходя в терминал системы.

**4.** Плагины позволяют под конкретный проект или задачу настроить функционал IDE

**5. Экономия времени и усилий.** Основная цель использования IDE — ускорить и упростить разработку. Они предоставляют множество полезных ресурсов, ярлыки, функции распознавания ошибок и многое другое.

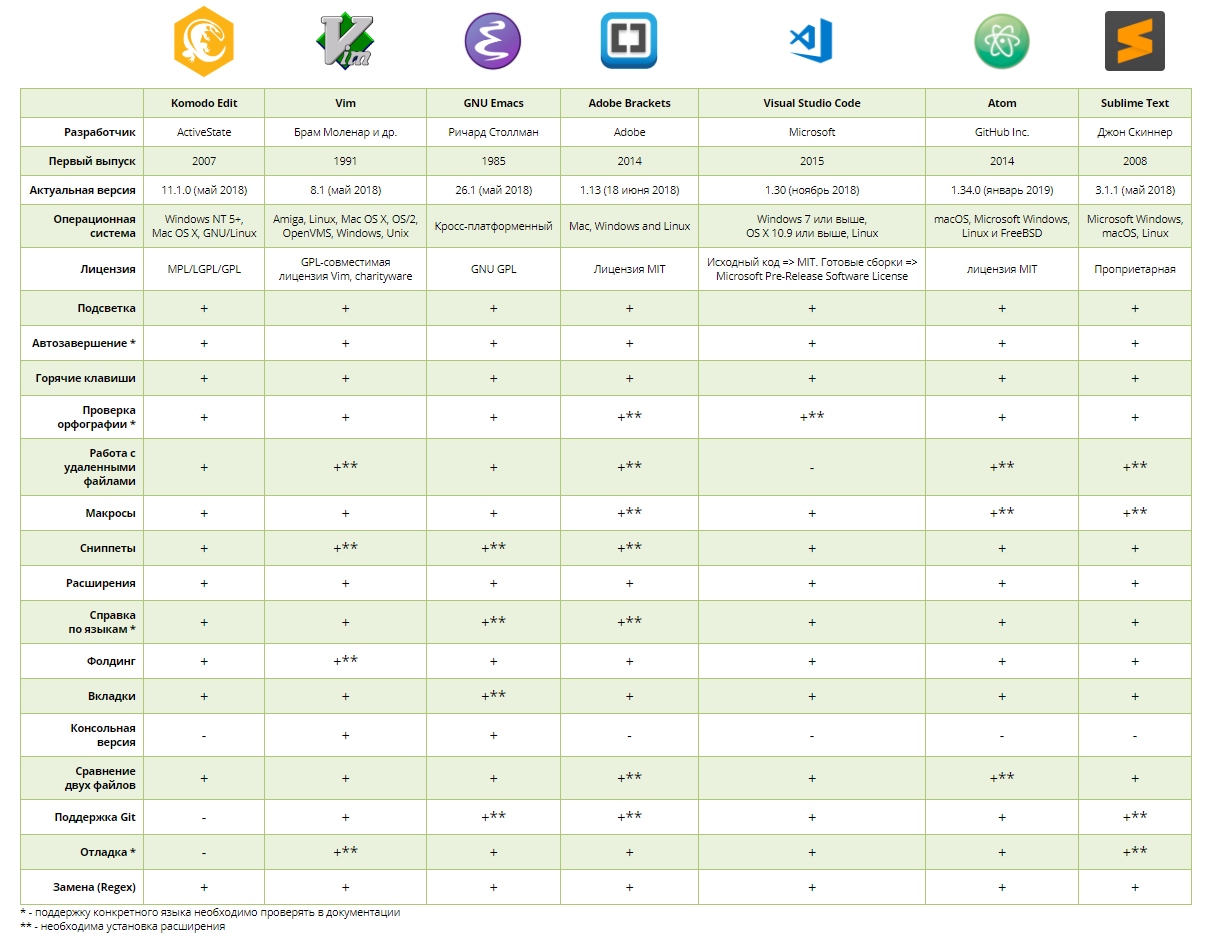
**Лёгкая настройка.** IDE объединяет различные возможности в одном месте и, следовательно, уменьшает трудности при постоянном переключении между инструментами.

**Поддержка совместной работы.** Большинство IDE поддерживают совместную работу групп и, таким образом, могут помочь ускорить производство.

**Возможность исправлять синтаксис, выдавать предупреждения и помогать писать качественный код.**

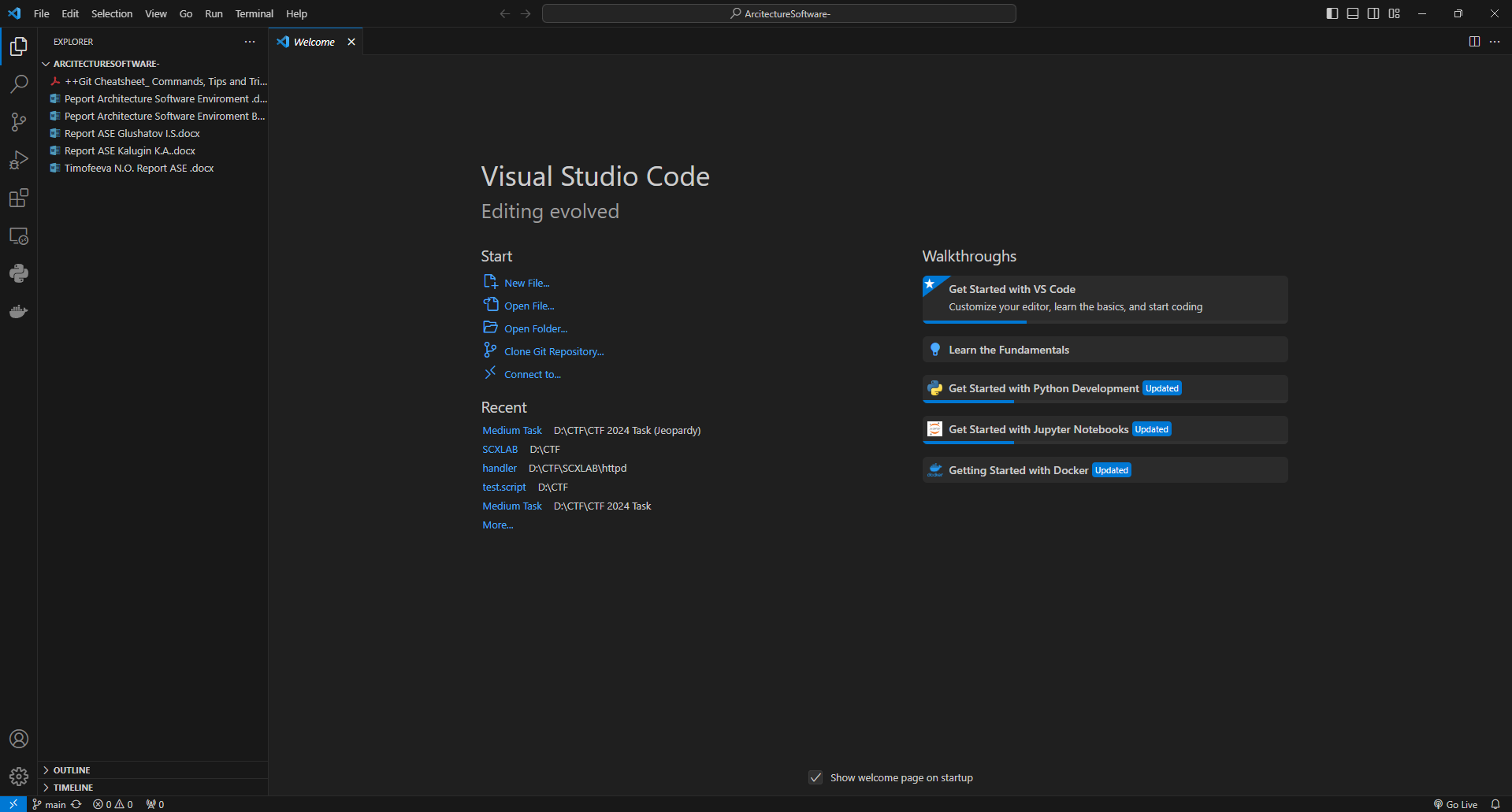
**Возможность создавать драйверы и утилиты.**

**Задача 1.**

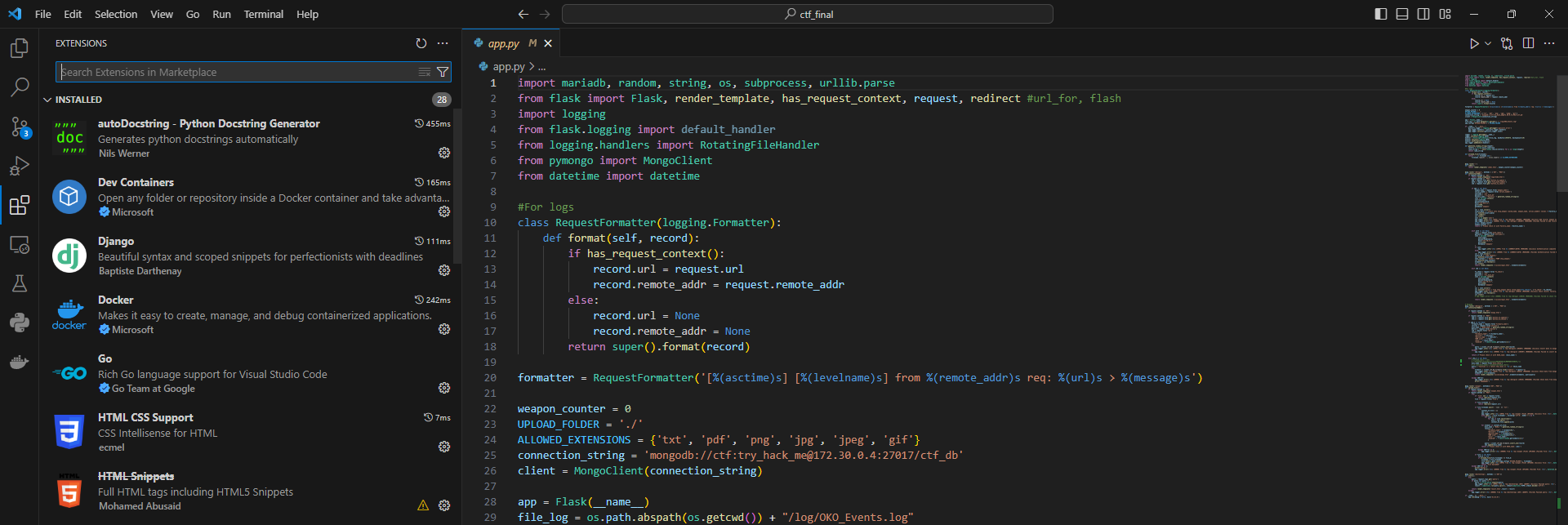


**Задача 2.**

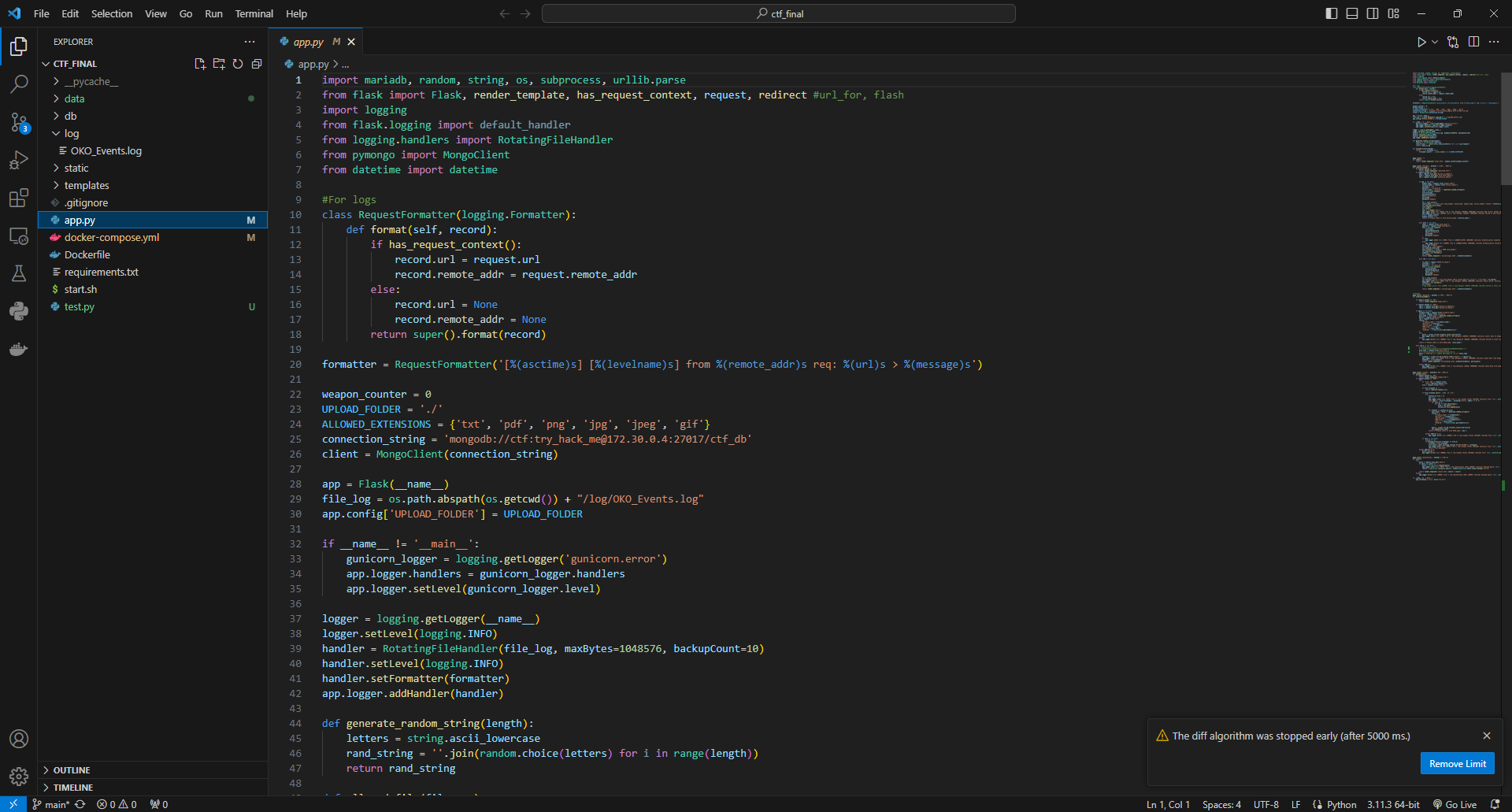
Visual Studio Code – среда разработки, которая построена на основе открытых источников, работает везде и имеет встроенные функции для запуска и отладки кода, а также расширения для установки новых языков, тем, отладчиков и подключения к дополнительным сервисам.

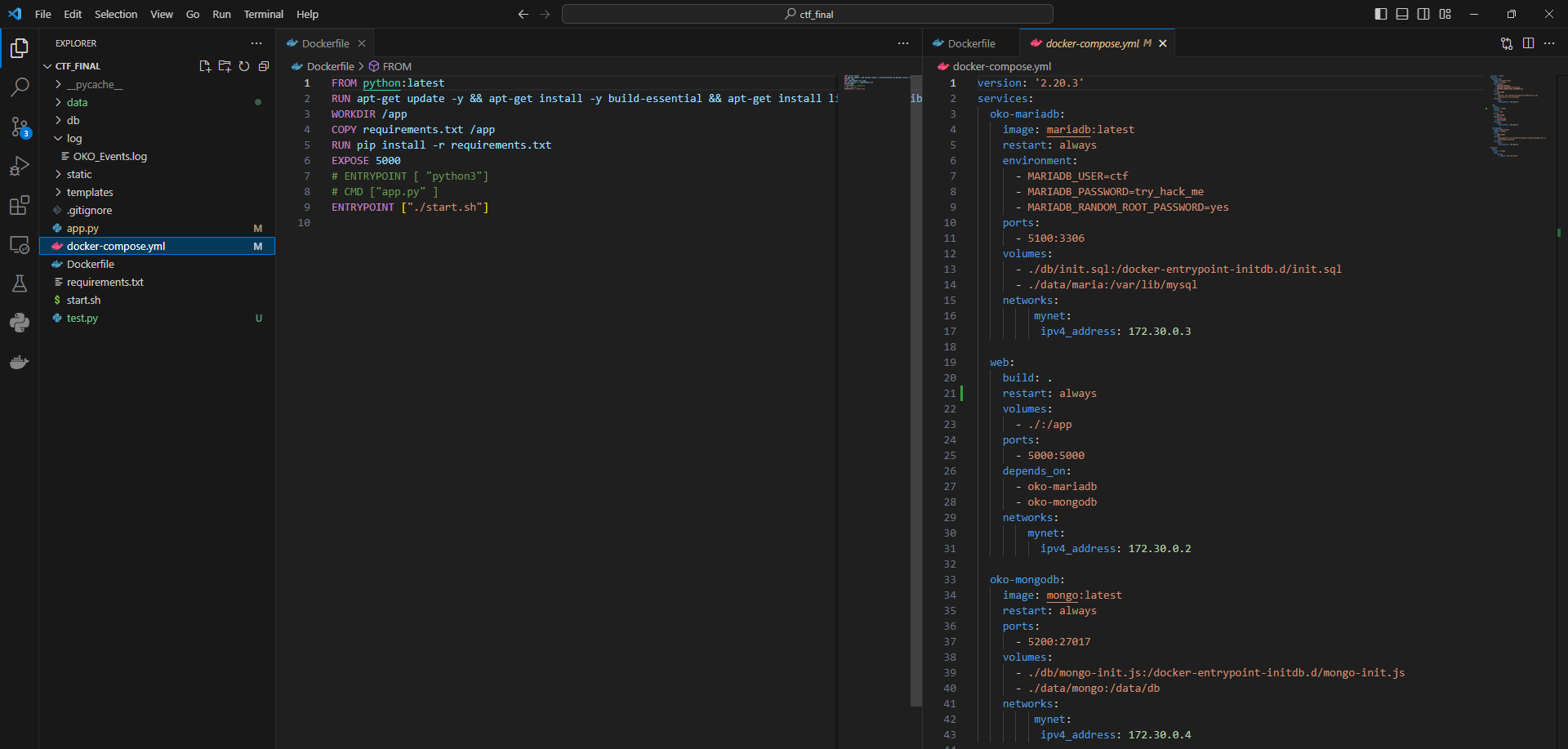


Основная особенность данной IDE заключается в том, что она полностью гибкая и позволяет её полностью настроить под себя и свои задачи. Это можно делать путём установки расширений.



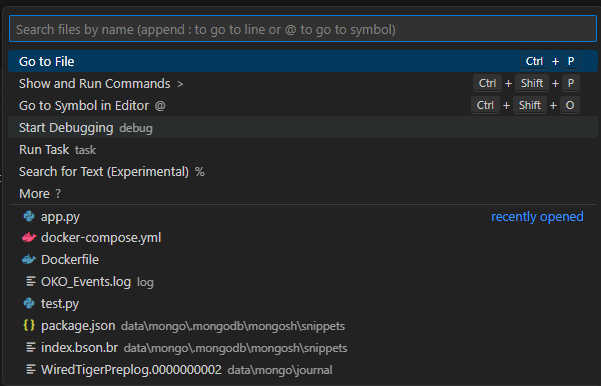
Ниже представлен пример кода на языке программирования Python с учётом предустановленных расширений (Python, Docker).



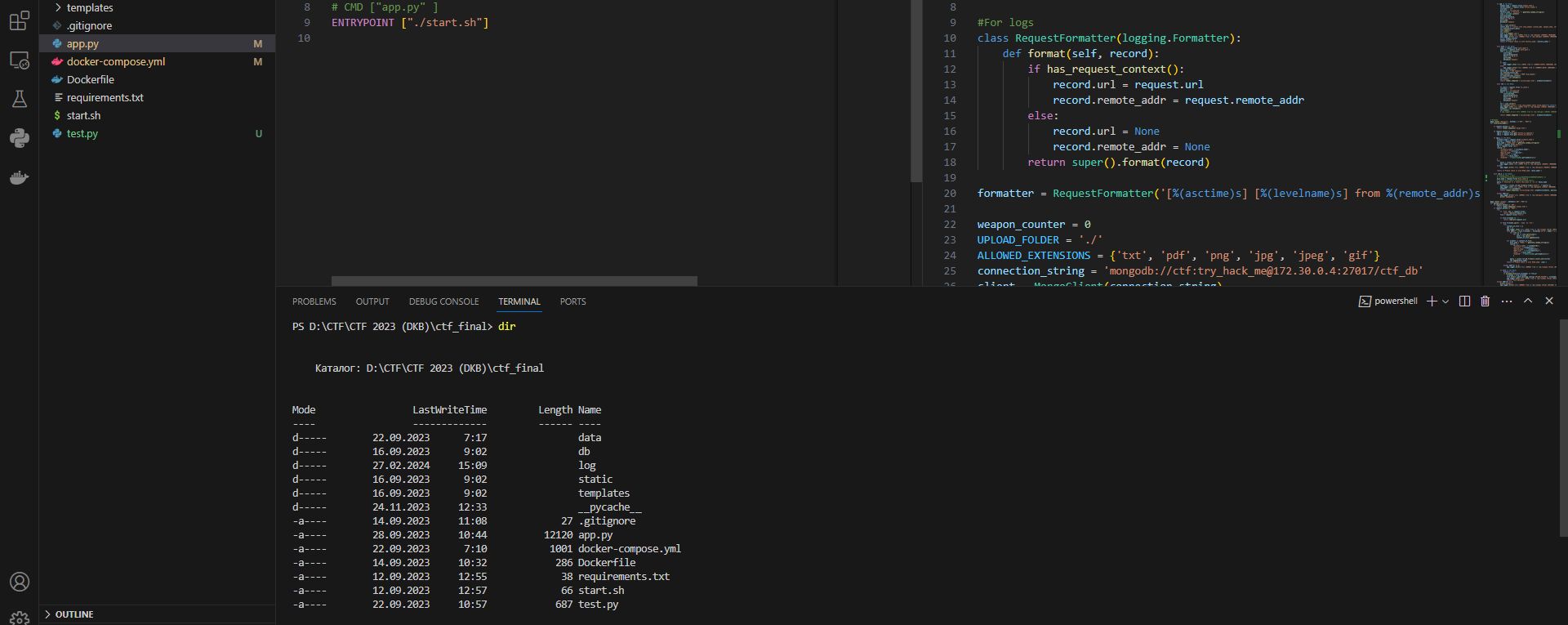
****

Расширения осуществляют подсветку кода, большинство из них поставляются с функционалом помощи разработчику в виде подсказок.

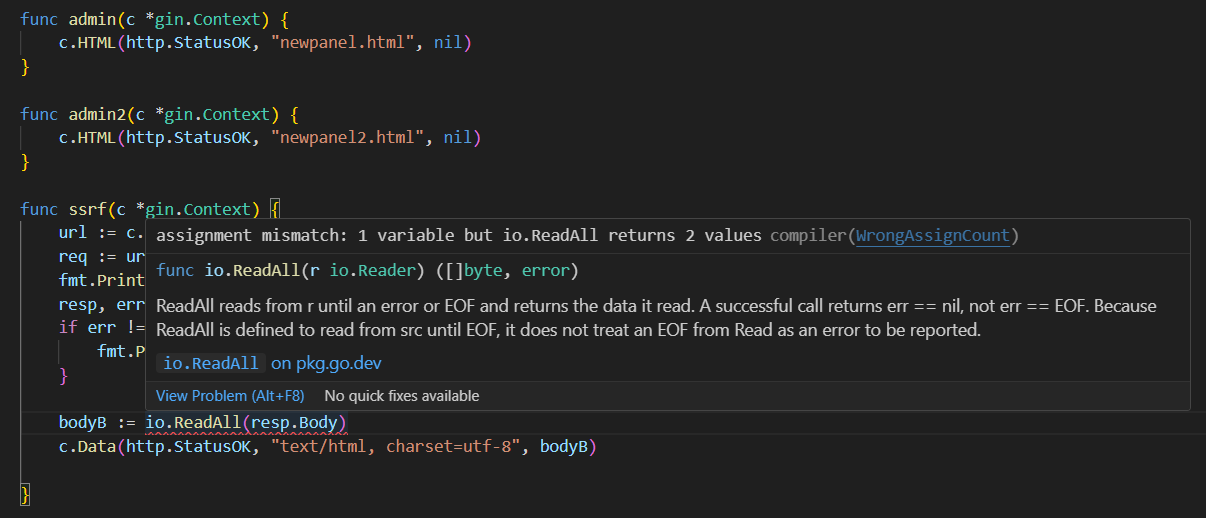
Также в данной среде предусмотрен функционал быстрого поиска и настройки проекта с помощью строки, показанной ниже.



В данной IDE существует встроенный терминал для пользователей, привыкших использовать данный инструмент в Linux или имеют хорошие навыки в использовании Power Shell. На рисунке ниже представлен результат работы команды **dir**, которая выводит список файлов в рабочей директории.



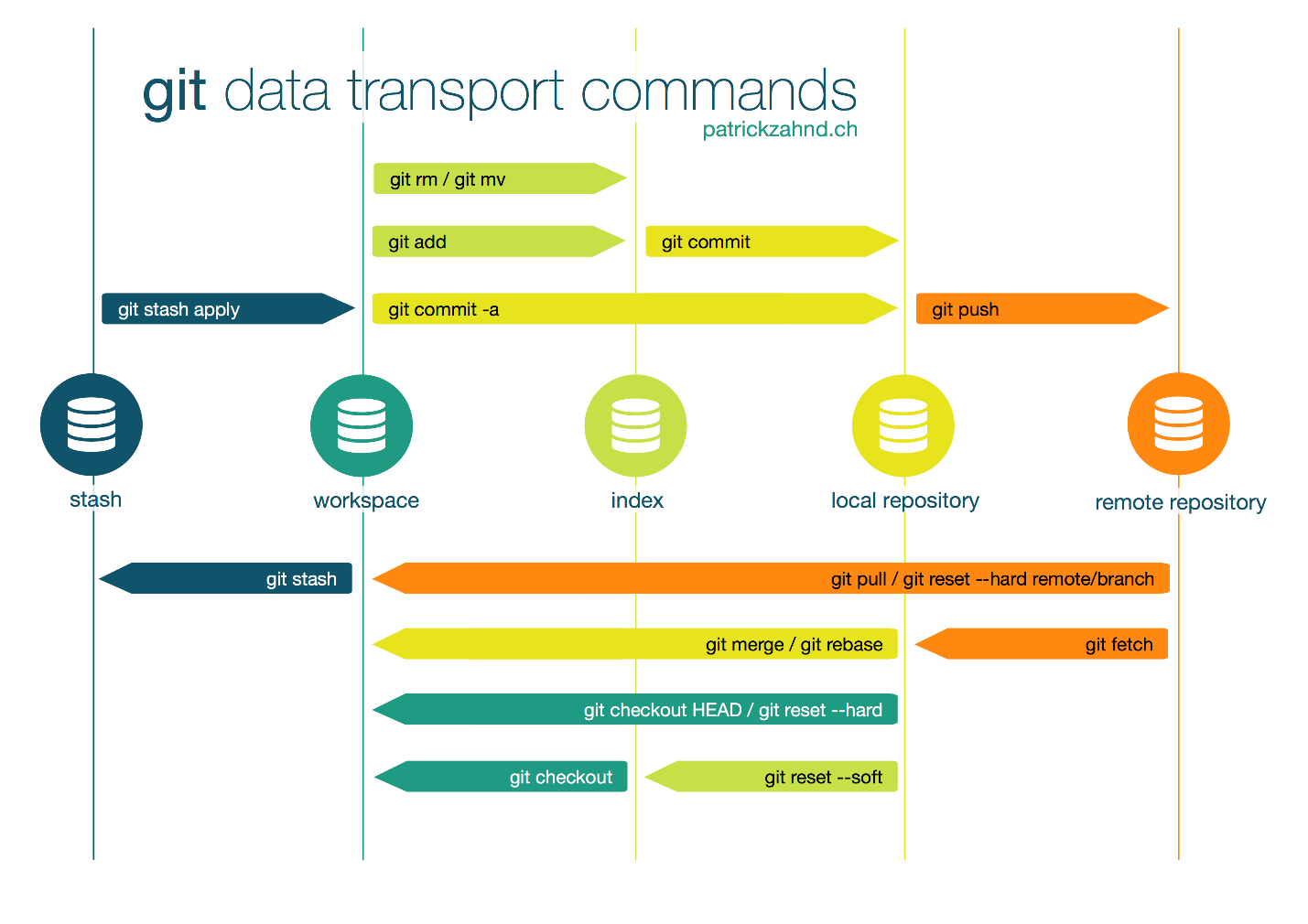
Далее было установлено расширение для языка Go, и будет показан пример работа режима отладки в режиме реального времени.



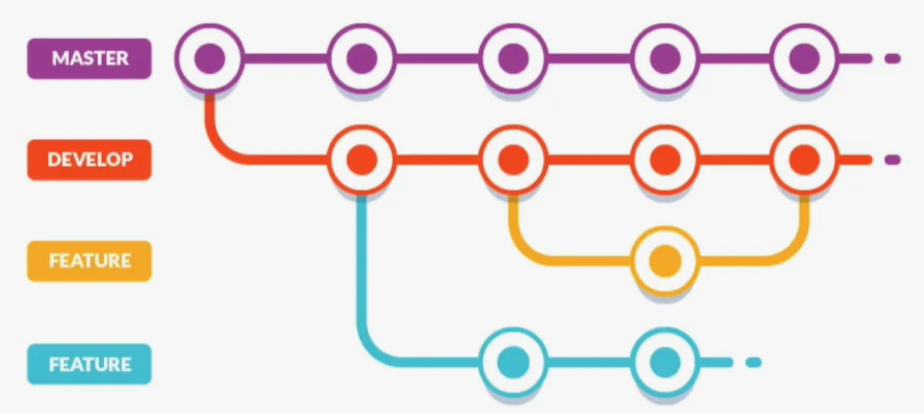
**Лабораторная 2.** Потоковая диаграмма GitHub и Архитектура среды.

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

**1.**

2/2/

**2.**Ветки помогают сделать разработки групповой, обезопасить от внезапных ошибок в коде, сделать процесс разработки прозрачным



**3. -** Основные принципы, лежащие в основе архитектуры GitHub, включают:

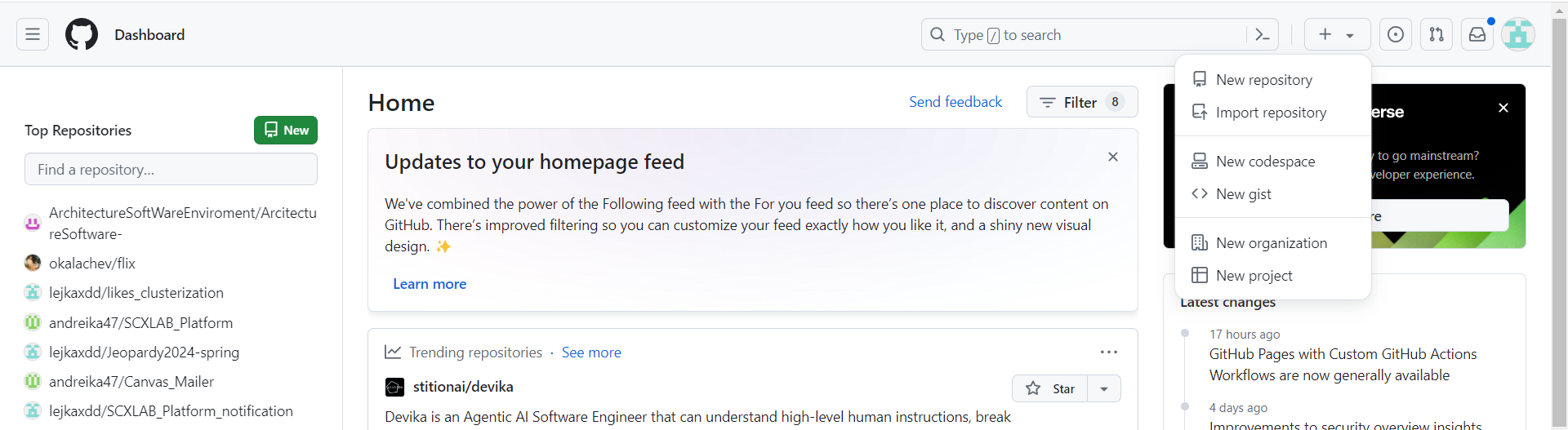
* **Асинхронность**: GitHub использует асинхронное программирование для обработки большого количества запросов одновременно. Это позволяет ему обрабатывать большое количество запросов без замедления работы.
* **Распределенность**: GitHub использует распределенную архитектуру, что означает, что он может работать на нескольких серверах одновременно. Это позволяет ему масштабироваться и обрабатывать большое количество запросов.
* **Горизонтальное масштабирование**: GitHub использует горизонтальное масштабирование для увеличения производительности. Это означает, что он может добавлять больше серверов для обработки большего количества запросов.
* **Вертикальное масштабирование**: Увеличение производительности в рамках одной сущности. Т.е мы выполняем модифакацию имеющихся ресурсов.
* **Кэширование:** GitHub использует кэширование для ускорения работы. Это означает, что он сохраняет данные в памяти или на диске, чтобы избежать повторного выполнения сложных операций**.**

**4.** Локальный репозиторий создается на основе существующего с помощью команд:

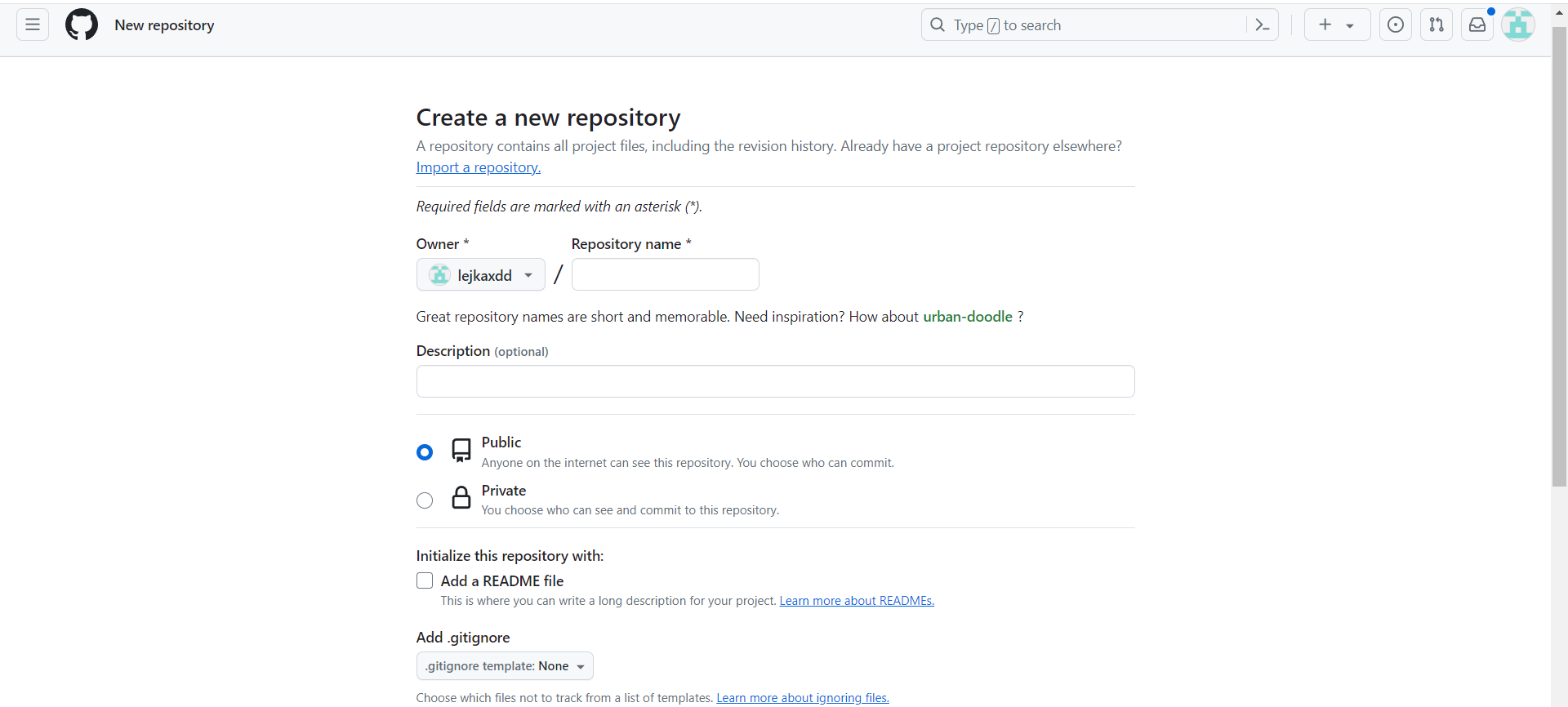
**git init –** создать локальный репозиторий

**git clone –** склонировать удаленный репозиторий

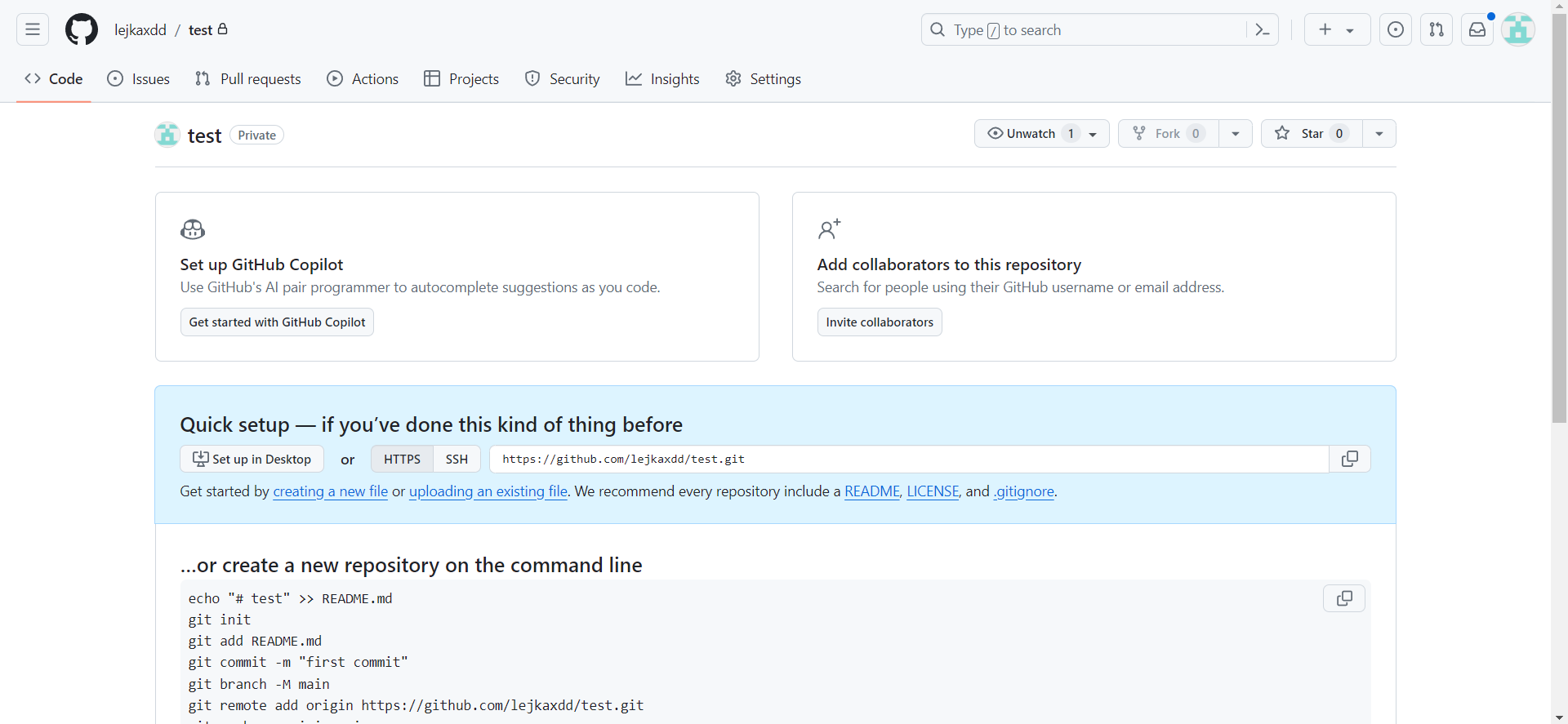
**5.** На главной странице нажать на кнопку создания репозитория.



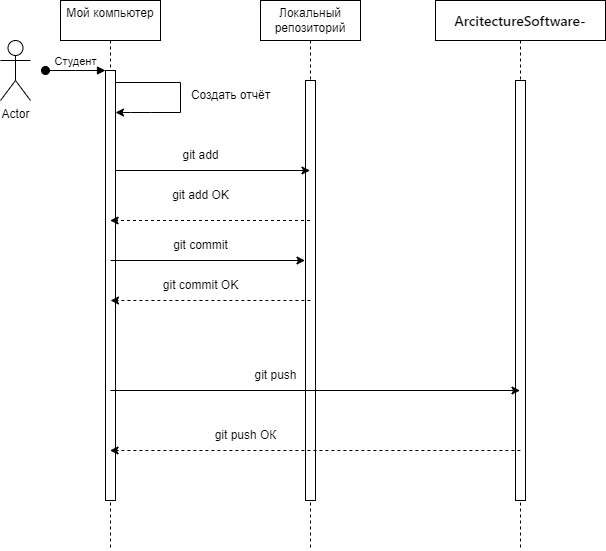
Задать нужные параметры



Репозиторий создан

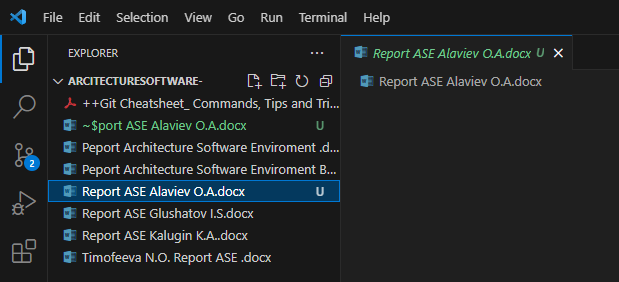


**Задание 1**

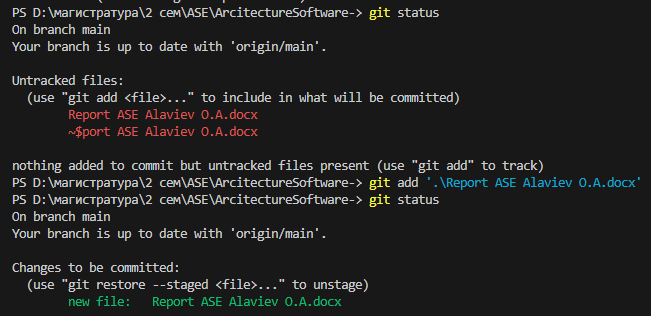


**Задание 2**

Ниже будет показан пример прохождения вышеуказанных этапов. Сначала был склонирован удаленный репозиторий. Создали текущий документ и внесли в него изменения.



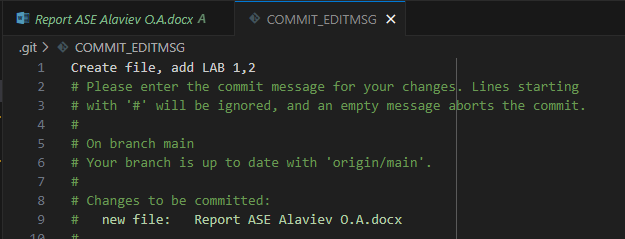
Мы видим, что наш файл находится в состоянии untracked. Сначала необходимо зафиксировать локальное состояние (commit)



Далее вводим команду git commit

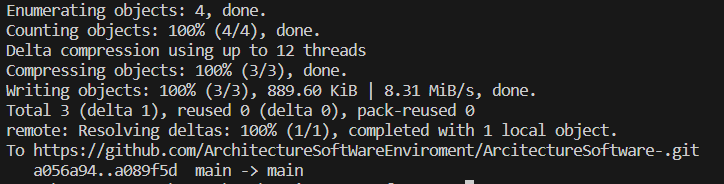


На каждый коммит лучше всего оставлять комментарии, чтобы в дальнейшем было понятно на каком этапе разработки были сделаны изменения.



После этого необходимо загрузить данное состояние в удаленный репозиторий с помощью команды git push.





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

**1.** Профили в VS Code — это функция, которая помогает пользователю подстраивать редактор индивидуально для каждого проекта.

Профиль отвечает за практически все настройки редактора:

* набор расширений,
* горячие клавиши,
* задачи по отладке,
* настройки не только редактора, но и его расширений.

Профиль можно применить на несколько проектов.

В Python профиле заранее будут установлены все необходимые компоненты для быстрого старта.

**2.** В VS Code есть два базовых профиля для работы с Python:

* Python Profile Template

Профиль Python является хорошей отправной точкой для разработки на Python. Он поставляется с фрагментами, специфичными для Python, и имеет следующие расширения:

autoDoc string

Black Formatter

Docker

Python

Python Environment Manager

Remote Development extension pack

* Data Science Profile Template

Профиль Data Science является хорошей отправной точкой для всей работы с данными и записной книжкой. Он поставляется с конкретными фрагментами и имеет следующие расширения:

Data Wrangler

Black Formatter

Jupyter

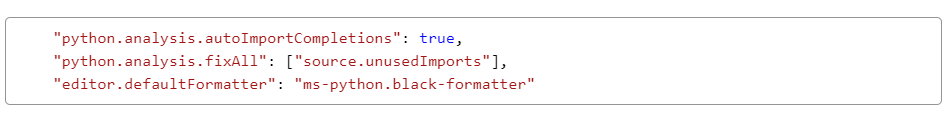
Python

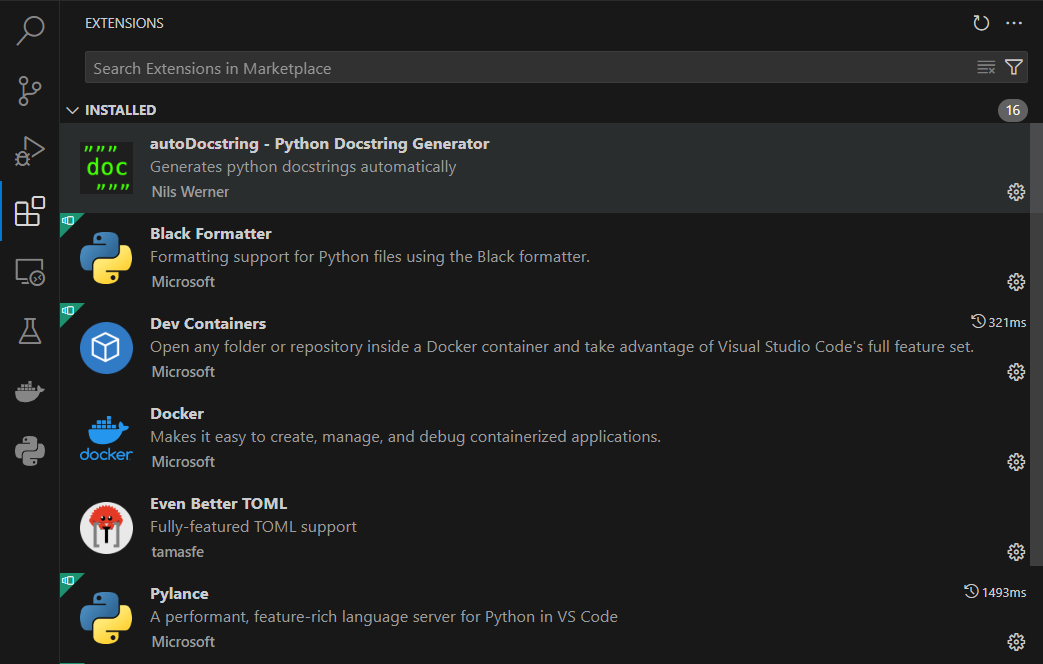
Dev Containers

GitHub Copilot - Your AI pair programmer.

**3.** Использование готовых профилей VS Code гарантирует, что при распределенной разработке у программистов будут одинаковые инструменты, это важный аспект при распределенной разработке.

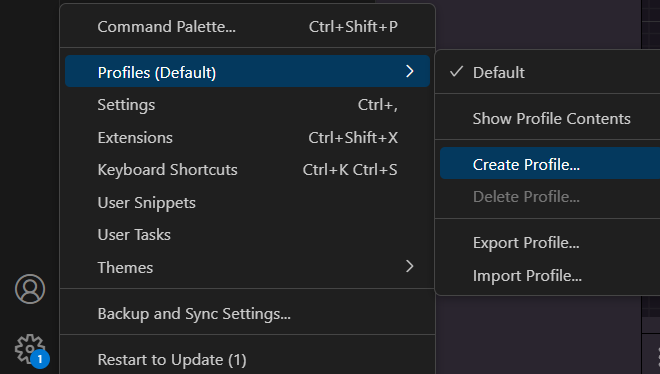
**4.** Типичная настройка.



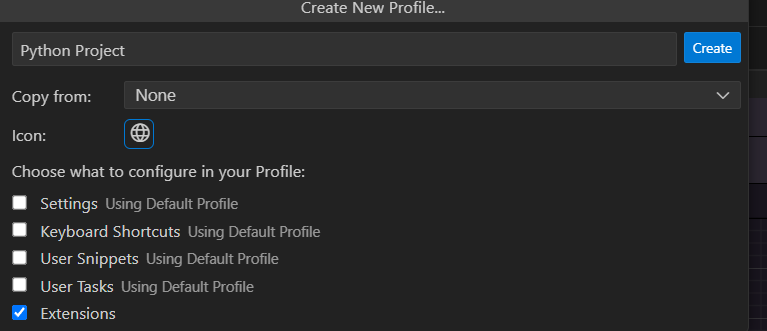


**5. -**

**Задание 1**



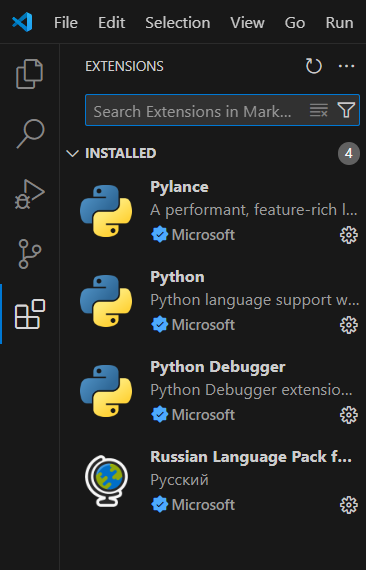
Задаем название для создаваемого профилю и выбираем корректируемые параметры.



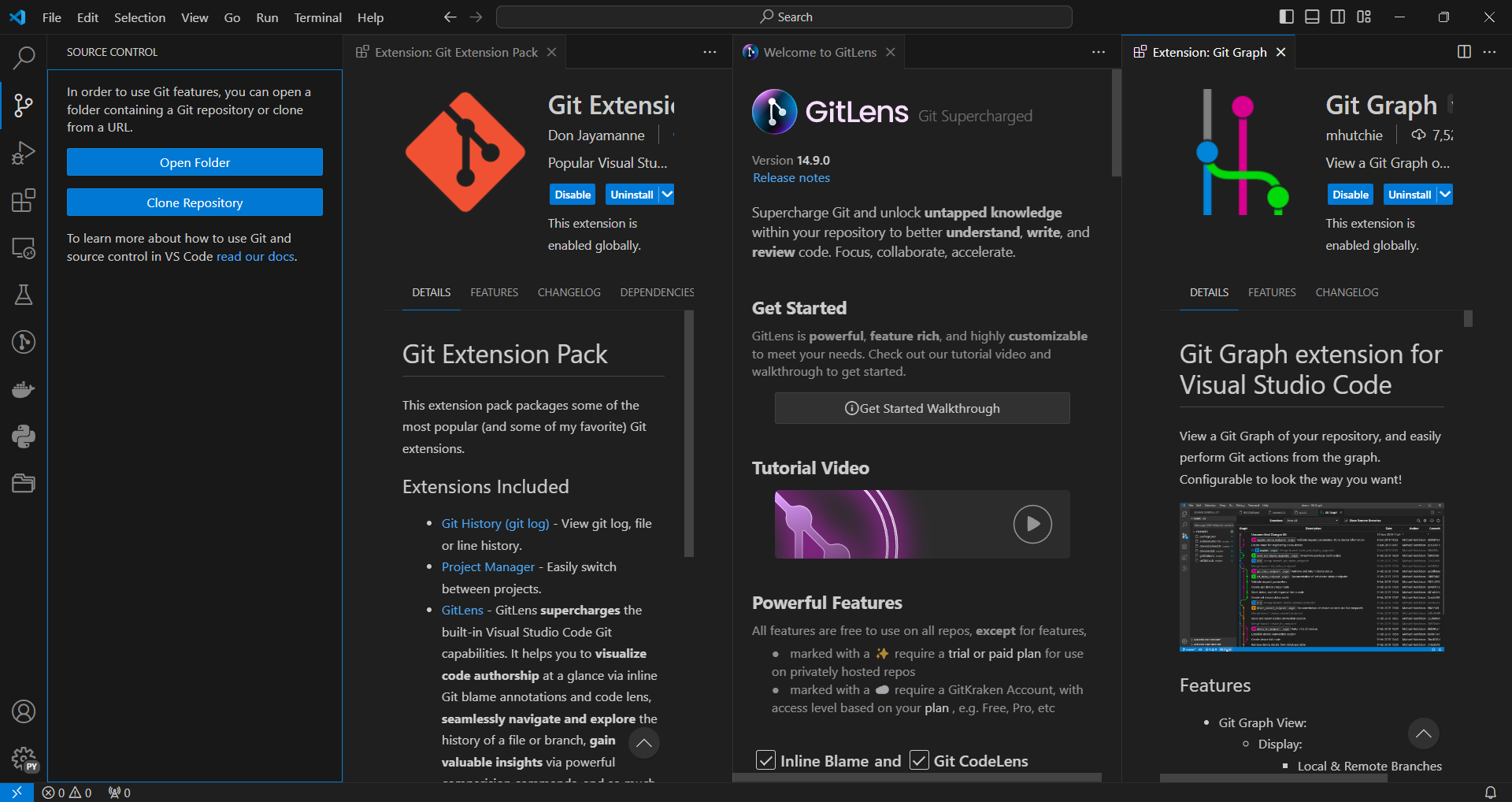
После этого видим, что в созданном профиле есть базовая часть и настройка расширений.



Устанавливаем расширения для работы с Python проектом



**Задание 2**



**Лабораторная работа 4** Потоковые диаграммы

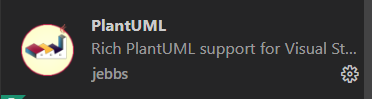
**1.** DFD (Data Flow Diagram) — это диаграмма, которая отображает потоки данных между системами и базами данных.

Основные элементы DFD диаграммы:

* Процесс. Процессы, при которых происходит изменение потока данных (обработка, трансформация и др.).
* Внешняя сущность. Сущность (объект), которая получает или отправляет данные при взаимодействии с описанным процессом.
* Хранилище данных. Все хранилища данных или отдельные файлы, которые хранят исходные или выходные данные, а также все промежуточные хранилища.
* Поток данных. Поток данных, который отображает направление и сами данные, перемещающиеся между внешними сущностями и хранилищами данных с помощью процессов.

**2.** Потоковая диаграмма помогает сделать процесс более прозрачным, видимым. Сразу можно увидеть архитектурные проблемы, все взаимодействия модулей. Всё это облегчает процесс разработки.

**3.** Для создания потоковой диаграммы в VS Code необходимо установить расширение



**4.** UML (Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

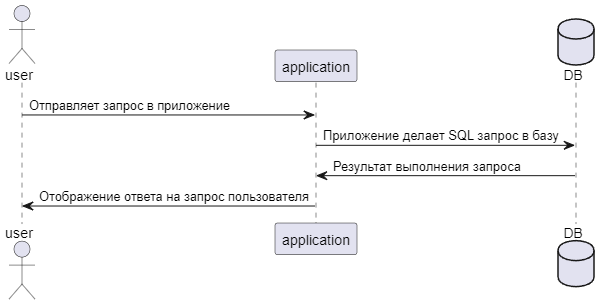
* **Диаграмма коммуникации** — взаимодействия между частями композитной структуры или ролями кооперации.
* **Диаграмма последовательности** — взаимодействия объектов, упорядоченные по времени их проявления.
* **Диаграмма сотрудничества** — позволяет описать взаимодействия объектов, абстрагируясь от последовательности передачи сообщений.
* **Диаграмма обзора взаимодействия** — разновидность диаграммы деятельности, включающая фрагменты диаграммы последовательности и конструкции потока управления.
* **Диаграмма синхронизации** — альтернативное представление диаграммы последовательности, явным образом показывающее изменения состояния на линии жизни с заданной шкалой времени.

**5.** Они могут быть полезны на различных этапах проектирования и анализа систем.

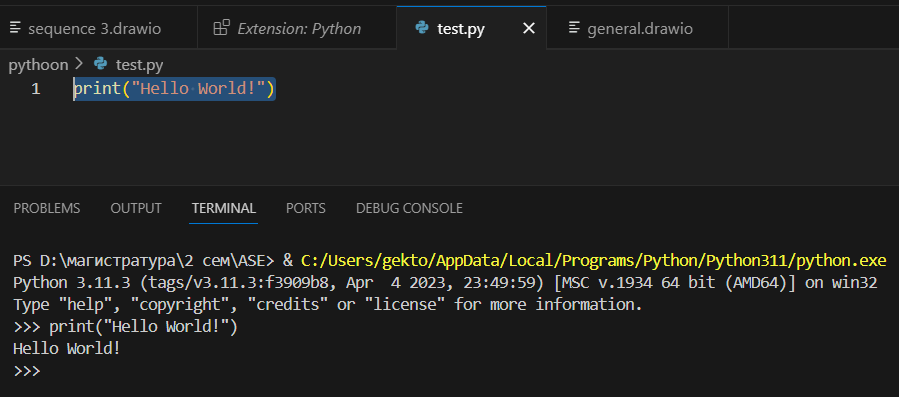
**Проектирование:** Диаграммы взаимодействий помогают разработчикам понять, как различные компоненты системы должны взаимодействовать друг с другом. Они позволяют визуализировать последовательность сообщений, которые передаются между объектами, и определить, какие объекты должны быть созданы и уничтожены в процессе работы системы.

**Анализ:** Диаграммы взаимодействий также могут быть полезны при анализе систем. Они позволяют аналитикам понять, как система работает и какие проблемы могут возникнуть при ее использовании. Аналитики могут использовать диаграммы взаимодействий для выявления потенциальных проблем и улучшения работы системы.

**Задание 1, 2**



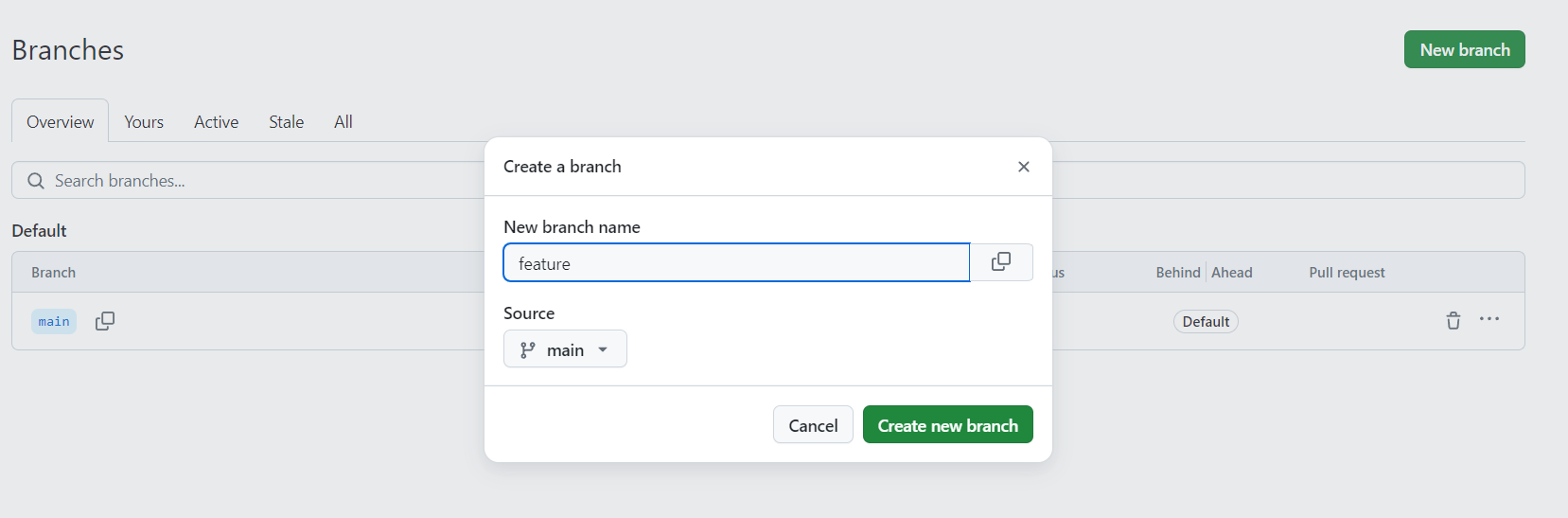
Напишем простую программу



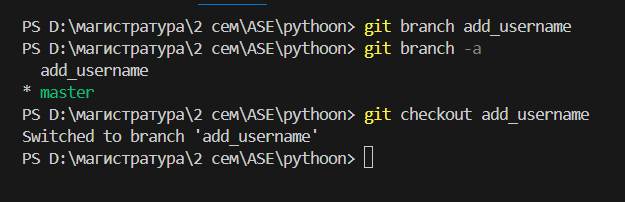
Как видим расширение работает, программа запустилась. Таким образом, можно сделать вывод, что создание профайлов это по сути дела гибкая настройка IDE для конкретного проекта или направления. Т.е можно создать проект для веб разработки на Go, либо профиль для разработки нативных приложений. При переходе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

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

**1.** Создание ветки через web-интерфейс.



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.



**2.** Принципы:

* **Использовать ветки для фиксации изменений**: Ветки Git позволяют фиксировать изменения в коде, которые еще не готовы к коммиту в основной ветке. Это позволяет разработчикам работать над разными задачами одновременно, не мешая друг другу.
* **Выполнять слияние веток**: Когда вы закончили работу над задачей, необходимо слить вашу ветку с основной веткой. Это позволит другим разработчикам увидеть ваши изменения и продолжить работу над проектом.
* **Использовать ветки для тестирования.** Ветки Git также могут быть использованы для тестирования новых функций или исправлений ошибок. Вы можете создать отдельную ветку для тестирования, а затем слить ее с основной веткой после успешного тестирования.

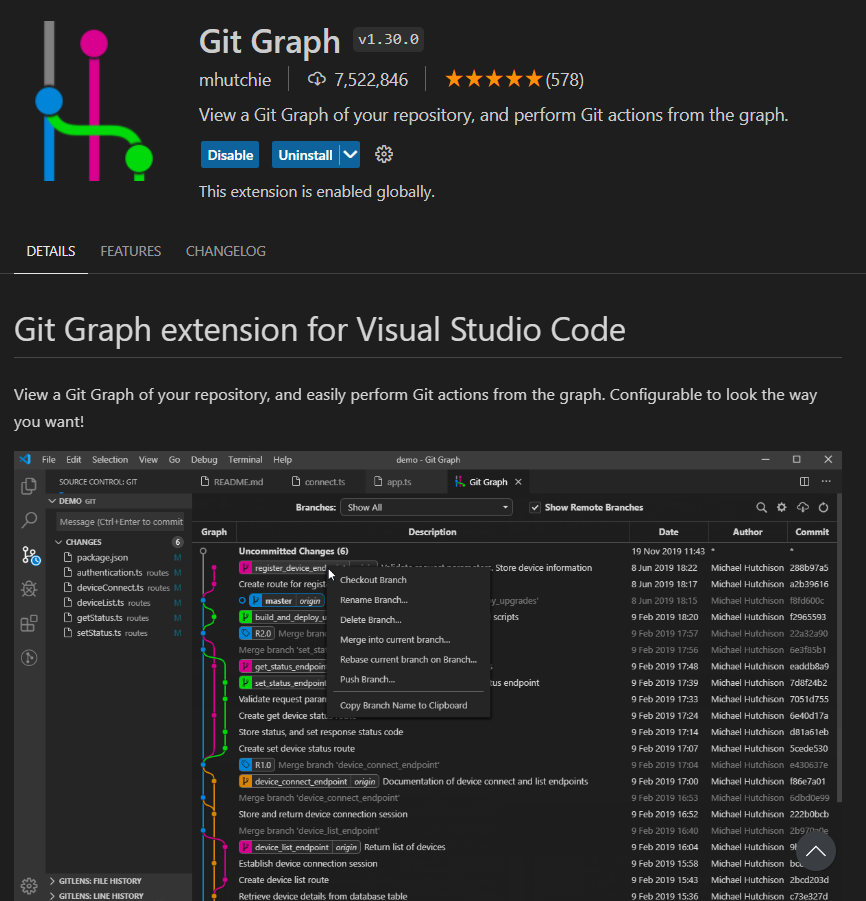
**3.** Использовать команду git merge

**4.** Конфликты слияния (merge conflicts) возникают при объединении двух или более веток в одну. Они происходят, когда в разных ветках были внесены изменения в одни и те же файлы или папки.

Для решения конфликтов слияния в VS Code необходимо открыть файл, в котором возник конфликт, и просмотреть его содержимое. В этом файле будут выделены строки, которые были изменены в разных ветках. Пользователь может выбрать, какую версию строки оставить в файле, или же внести изменения вручную.

После того, как все конфликты слияния были решены, пользователь может продолжить слияние веток. Если все конфликты были успешно решены, то VS Code автоматически объединит все изменения в одну ветку.

**5.** Расширение для удобного управления ветками.



**Задание 1, 2**

Исходный код приложения

import sqlite3

# Устанавливаем соединение с базой данных

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

# Создаем таблицу Users

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY,

username TEXT NOT NULL)

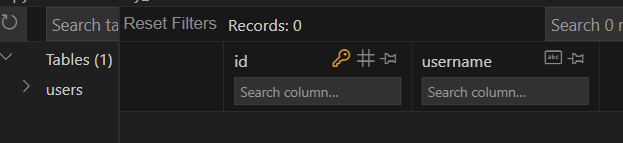
''')

# Сохраняем изменения

connection.commit()

connection.close()

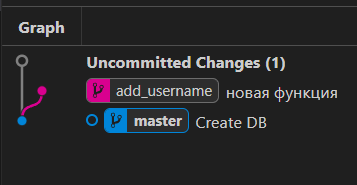
Результат работы



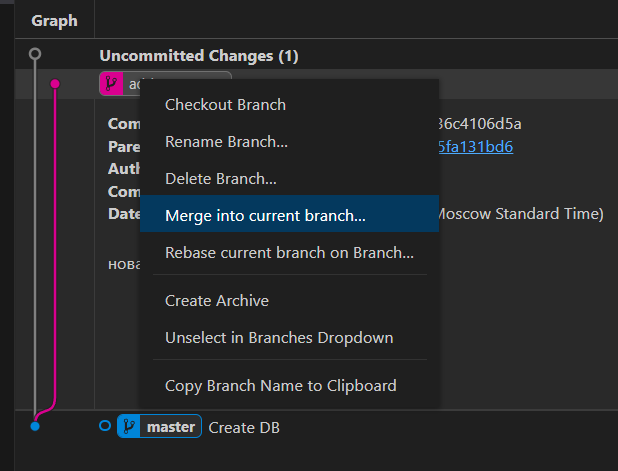
Исходный код в ветке **add\_username**



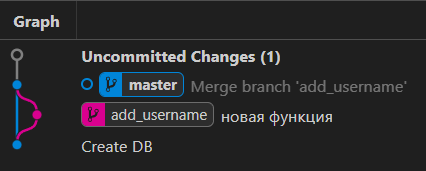
В VS Code увидим изменения после перемещения в мастер ветку



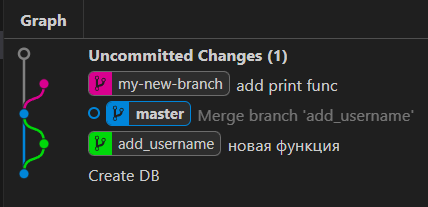
Далее можем выполнить слияние созданной ветки с мастер.



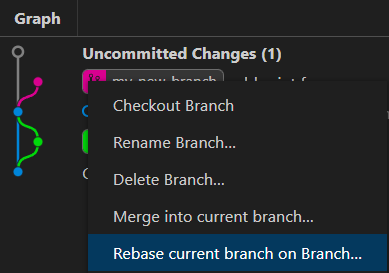
Результат слияния веток



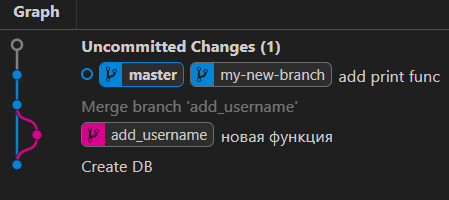
Теперь выполним смещения статуса мастер ветки на новую ветку. Для этого создаем новую ветку и изменяем немного исходный код.



И выполняем функцию rebase branch

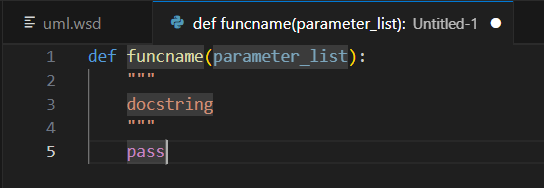


Результат выполнения операции rebase

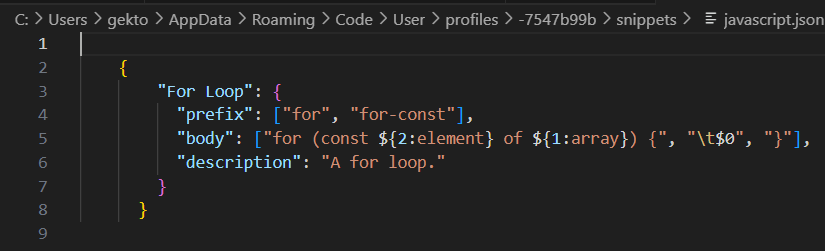


Лабораторная работа 6. User Snippets. User Task

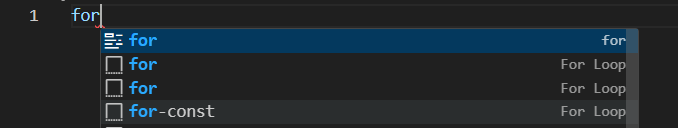
1. Сниппеты - это шаблоны, которые упрощают ввод повторяющихся шаблонов кода, таких как циклы или условные операторы. Пример представлен ниже. Данные шаблоны ускоряют процесс разработки ПО.



Пользовательский сниппет



Пример его вызова из кода



Результат вызова

