Московский авиационный институт

(Национальный исследовательский университет)

**Лабораторные работы 1 - 12**

**Архитектура программной среды**

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

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

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

**Лабораторная 4.** Разработка потоковых диаграмм их реализация в среде VS Code.

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

**Лабораторная 6.** Файлы конфигураций (settings), создание User Task, настройка редактора VS Code.

**Лабораторная 7.** Создание и работа в профиле для формирования документации.

**Лабораторная 8.** Создание и работа в профиле VS Code для Data Science.

**Лабораторная 9.** Создание и отладка веб-приложений с использованием расширения VS Code для браузера.

**Лабораторная 10.** Создание и работа в профиле VS Code для AI приложений: NumPy, TensorFlow и Python.

**Лабораторная 11.** Использование Docker для разработки и развертывания приложений.

**Лабораторная 12.** Использование ChatGPT для программирования прикладных задач.

|  |  |  |
| --- | --- | --- |
| Тимофеева Н. О. | подпись: | \_\_\_\_\_\_ |
| Семенов А. С. | подпись: | \_\_\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_ | 2024 |

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

1. **Что такое интегрированная среда разработки (IDE) и для чего она используется?**

IDE (интегрированная среда разработки) - это программное обеспечение, предназначенное для облегчения процесса разработки. Оно обычно включает в себя текстовый редактор, инструменты для отладки кода, компиляции, а также другие функции, например, подсветку синтаксиса, интеграцию с системами контроля версий и т. д. IDE позволяет программистам работать более эффективно, упрощая процесс написания, тестирования и отладки кода.

1. **Какие основные компоненты включает типичная IDE?**

* Текстовый редактор — используется для написания и редактирования исходного кода.
* Компилятор или интерпретатор: Позволяет преобразовать исходный код в исполняемый код. Компиляторы используются для языков, компилируемых в машинный код, а интерпретаторы - для языков, интерпретируемых построчно.
* Отладчик — инструмент для поиска и исправления ошибок в коде пошагово.
* Система управления версиями — интеграция с системами контроля версий позволяет отслеживать изменения в коде, вносить комментарии и совершать коммиты.

1. **В чем отличие между текстовым редактором и интегрированной средой разработки?**

Текстовый редактор - простой инструмент для написания и редактирования текста, включая исходный код. IDE же предоставляет более широкий спектр инструментов для разработки программного обеспечения. Это включает в себя текстовый редактор, компилятор или интерпретатор, отладчик, систему управления версиями.

1. **Какую роль играют плагины в работе с интегрированными средами разработки?**

Плагины представляют собой дополнительные модули, которые расширяют функциональность интегрированных сред разработки (IDE). Они могут добавлять новые функции, поддержку языков программирования, изменять внешний вид и другие возможности, что позволяет настраивать IDE под конкретные потребности разработчика

1. **Какие преимущества предоставляет использование IDE при разработке программного обеспечения?**

* Увеличение производительности — IDE предоставляют различные инструменты, такие как автодополнение кода, подсвечивание кода и ошибок, что помогает ускорить процесс написания и отладки кода.
* Повышение качества кода — IDE обычно предоставляют инструменты для контроля качества и форматирования, что помогает выявлять и исправлять ошибки и поддерживать стандарты написания кода.
* Интеграция с внешними сервисами — многие IDE интегрированы с системами управления версиями, что облегчает совместную работу и управление проектами.
* Поддержка различных языков программирования — многие IDE поддерживают несколько языков программирования и фреймворков, что позволяет разработчикам работать с разнообразными технологиями в единой среде.
* Расширяемость — большинство IDE поддерживают плагины и расширения, позволяющие настраивать среду разработки под конкретные потребности и добавлять новые функции и инструменты.

**Задача 1: Выберите 2 популярные интегрированные среды разработки и сравните их основные характеристики**

При выборе редактора кода для работы над проектом перед разработчиком часто встает вопрос о том, какой инструмент лучше всего подходит для решения его задач. Рассмотрим два популярных редактора – PhpStorm от компании JetBrains и VSCode от Microsoft.

Основные характеристики

Обе программы предоставляют набор функций, необходимых для разработки на разных языках программирования. Они поддерживают системы контроля версий, такие как Git, предоставляют инструменты для отладки и анализа кода, а также обеспечивают подсветку синтаксиса и автоматическое завершение кода.

Однако, есть различия. Например, PhpStorm имеет более широкий набор инструментов для работы с PHP, включая интеграцию с популярными фреймворками, такими как Laravel и Symfony.

Преимущества PhpStorm:

– Широкий функционал: PhpStorm предоставляет широкий спектр инструментов для разработки на PHP, таких как отладка, анализ кода, автодополнение и другие.

– Интеграция с другими продуктами: PhpStorm хорошо интегрируется с другими продуктами компании JetBrains, что может быть удобно для разработчиков, использующих несколько продуктов.

– В Phpstorm есть идеальный поиск. Если нужно найти файл, класс, настройку или информацию по git, двойной клик по shift открывает поисковик по этим параметрам.

Недостатки PhpStorm:

– Высокая стоимость: PhpStorm стоит дороже, чем Visual Studio Code.

– Ограниченная поддержка других языков: PhpStorm в основном ориентирован на разработку на PHP, и поддержка других языков может быть ограничена.

Преимущества Visual Studio Code:

– Низкая стоимость: Visual Studio Code значительно дешевле, чем PhpStorm.

– Поддержка множества языков: Visual Studio Code поддерживает множество языков программирования, включая JavaScript, TypeScript, Python и многие другие.

Недостатки Visual Studio Code:

– Меньший функционал: Visual Studio Code не предоставляет такого широкого спектра инструментов, как PhpStorm.

– Не такая хорошая интеграция с другими продуктами: Visual Studio Code менее интегрирован с другими продуктами, чем PhpStorm.

Если вы в первую очередь PHP-разработчик, то PHPStorm — очевидный выбор. Он предлагает отличную поддержку PHP с такими функциями, как завершение кода, отладка и тестирование. Однако если вы работаете с несколькими языками программирования или хотите попробовать разное, Visual Studio Code может оказаться лучшим выбором, поскольку он предлагает поддержку широкого спектра языков и платформ.

**Задача 2: Создайте простое приложение на языке программирования вашего выбора, используя одну из интегрированных сред разработки, и продемонстрируйте процесс создания, компиляции и запуска**

Я выбрала написать приложение на языке PHP, так как этот язык наиболее используется мной в последнее время. Как обсуждалось в задаче 1, наиболее удобной для этого языка IDE является PhpStorm.

* Создание проекта:

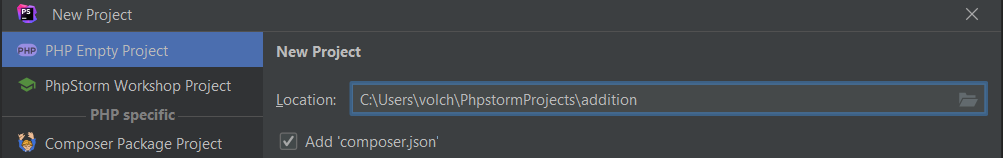
Открываем PhpStorm



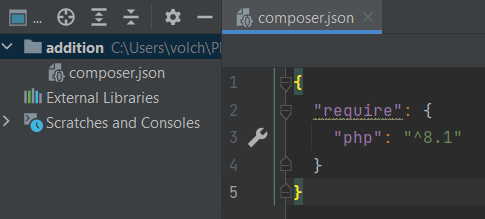
Выбираем File | New Project



Создаём пустой проект c файлом composer.json. Там хранятся все необходимые зависимости



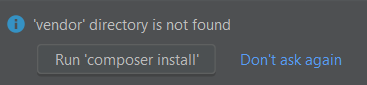
В файле composer.json пишем одну из последних версий языка, чтобы именно она использовалась в коде



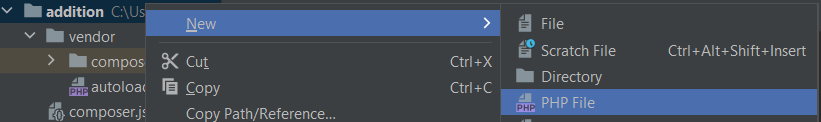
IDE подсказывает, что пакеты не установлены

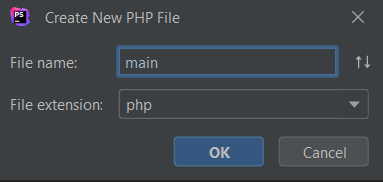


А также предлагает их установить. Нажмём на эту кнопку

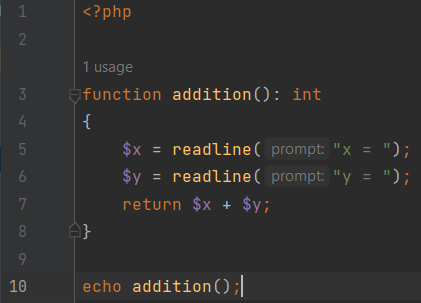


Теперь можно добавить новый файл, где будем писать код

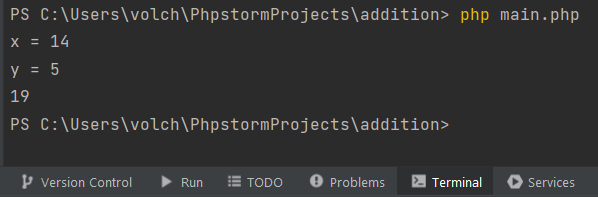




Напишем код для сложения двух целых чисел



Так как язык интерпретируемый, то нам не нужно компилировать. Запустим код в терминале и посмотрим ответ



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

1. **Что такое потоковая диаграмма в контексте GitHub и какие основные элементы она включает?**

Потоковая диаграмма в контексте GitHub представляет собой визуальное представление процесса работы над проектом с использованием системы контроля версий GitHub. Она обычно включает основные этапы или шаги работы над задачей или проектом, а также демонстрирует последовательность выполнения этих шагов.

В потоковой диаграмме GitHub обычно выделяются следующие элементы:

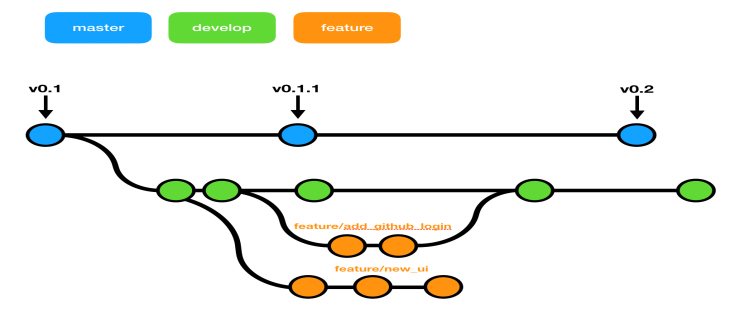
* Workspace – это рабочее пространство, где располагаются все файлы и директории, над которыми мы работаем. Здесь происходит редактирование, создание и удаление файлов.
* Index – это промежуточное хранилище измененных файлов перед фиксацией изменений (коммитом).
* Local repository – это локальное хранилище, куда коммитятся изменения. Здесь также хранятся ветки.
* Remote repository – это централизованное удаленное хранилище, где хранятся версии проекта. Оно позволяет делиться изменениями между несколькими компьютерами и членами команды.

Для синхронизации изменений между локальным и удаленным репозиториями используются операции, такие как push (отправка изменений из локального репозитория в удаленный) и pull (получение изменений из удаленного репозитория в локальный).

1. **Какую роль играют ветки (branches) в потоковой диаграмме GitHub и каким образом они помогают в организации работы над проектом?**

Ветки играют ключевую роль в потоковой диаграмме GitHub, обеспечивая эффективную организацию работы над проектом. Вот основные функции и преимущества веток:

* Создание различных версий кода — ветки позволяют создавать отдельные версии кода, например, ветка продакшена (master), ветка разработка (develop) и ветки для работы над конкретными функциями или задачами (feature). Это позволяет изолировать различные изменения и эксперименты, обеспечивая четкую структуру разработки.
* Параллельная работа над разными функциями — разработчики могут создавать и работать над несколькими ветками параллельно. Это позволяет разделить работу на небольшие задачи и управлять ими независимо друг от друга, что ускоряет процесс разработки.
* Отслеживание изменений и истории разработки — GitHub автоматически отслеживает и сохраняет историю изменений в каждой ветке, что облегчает восстановление предыдущих версий кода и отслеживание прогресса работы над проектом.



1. **Какие основные принципы лежат в основе архитектуры среды GitHub, позволяющей эффективно управлять версиями кода и совместной разработкой?**

Такими принципами являются:

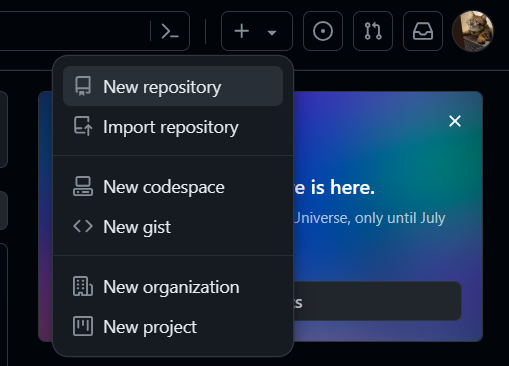
* Git — GitHub полностью интегрирован с Git, распределенной системой контроля версий. Это позволяет разработчикам отслеживать изменения в коде, создавать ветки для параллельной разработки, вносить правки и объединять их в основную ветку.
* Ветки — это разные версии кода проекта, которые позволяют работать над различными функциями или исправлениями параллельно.
* Pull Requests — это запросы на внесение изменений в код проекта. Они позволяют разработчикам обсуждать и проверять предлагаемые изменения перед их интеграцией в основную ветку.

1. **Каким образом создаётся локальный репозиторий на базе существующего удалённого репозитория GitHub?**

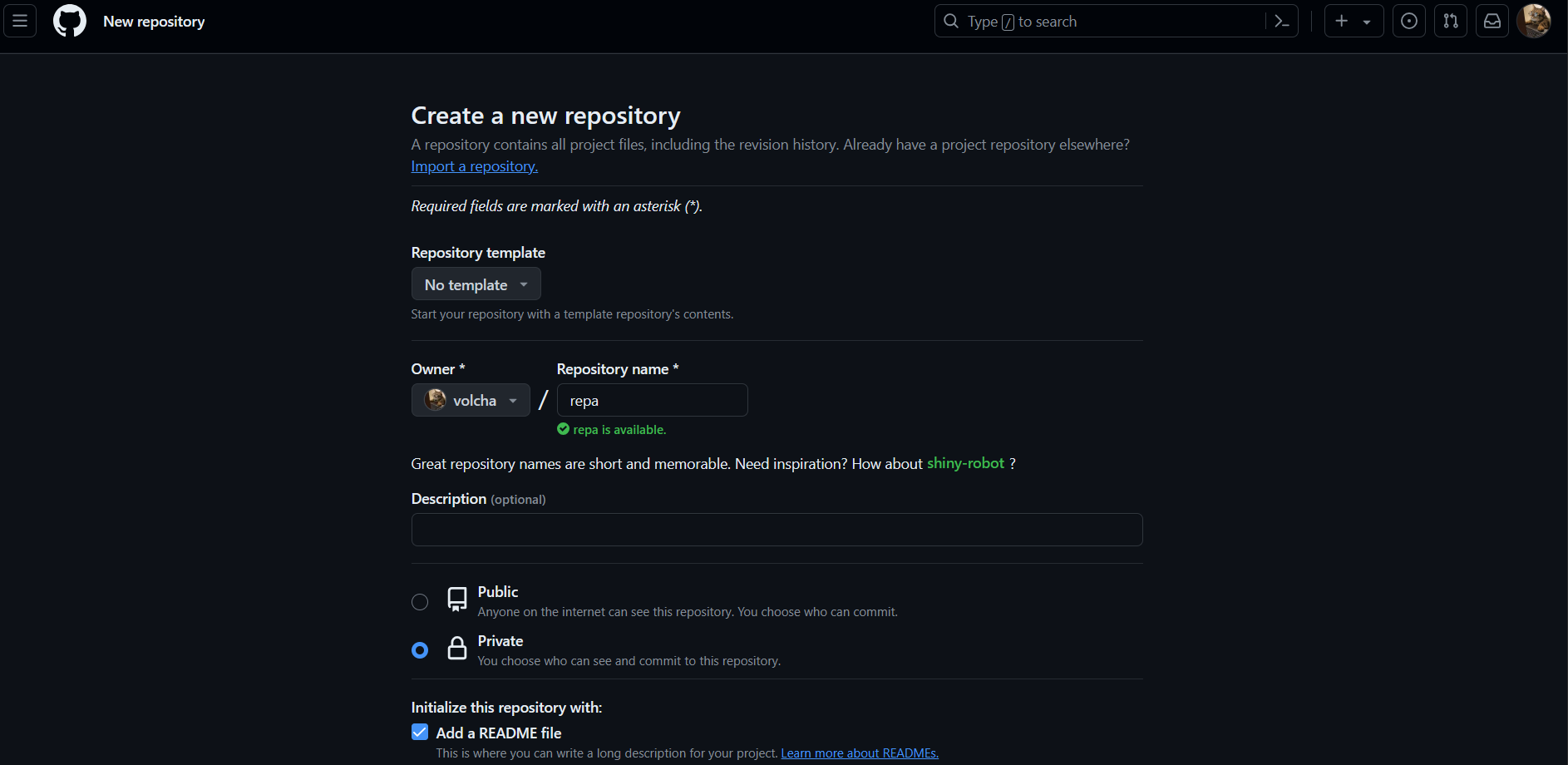
* Заходим в терминале с помощью cd в папку, где будет храниться репозиторий
* копируем HTTPS адрес удалённого репозитория из GitHub
* В терминале вводим: git **clone** <your repository name>
* С помощью cd проваливаемся внутрь проекта

1. **Какие шаги необходимо выполнить для создания системного репозитория в GitHub и какие преимущества это может предоставить при работе в команде?**

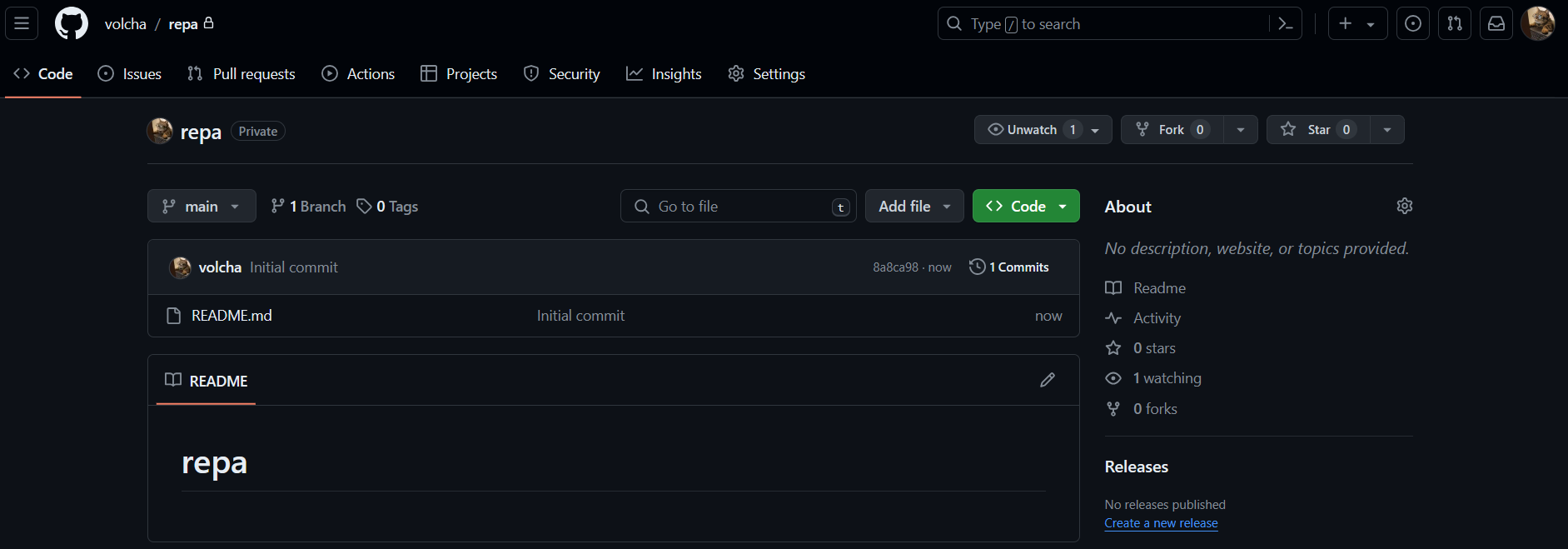
На странице своего профиля в GitHub нажать на New repository

****

Задать параметры, такие как название, добавление файла ля описания проекта, приватность:



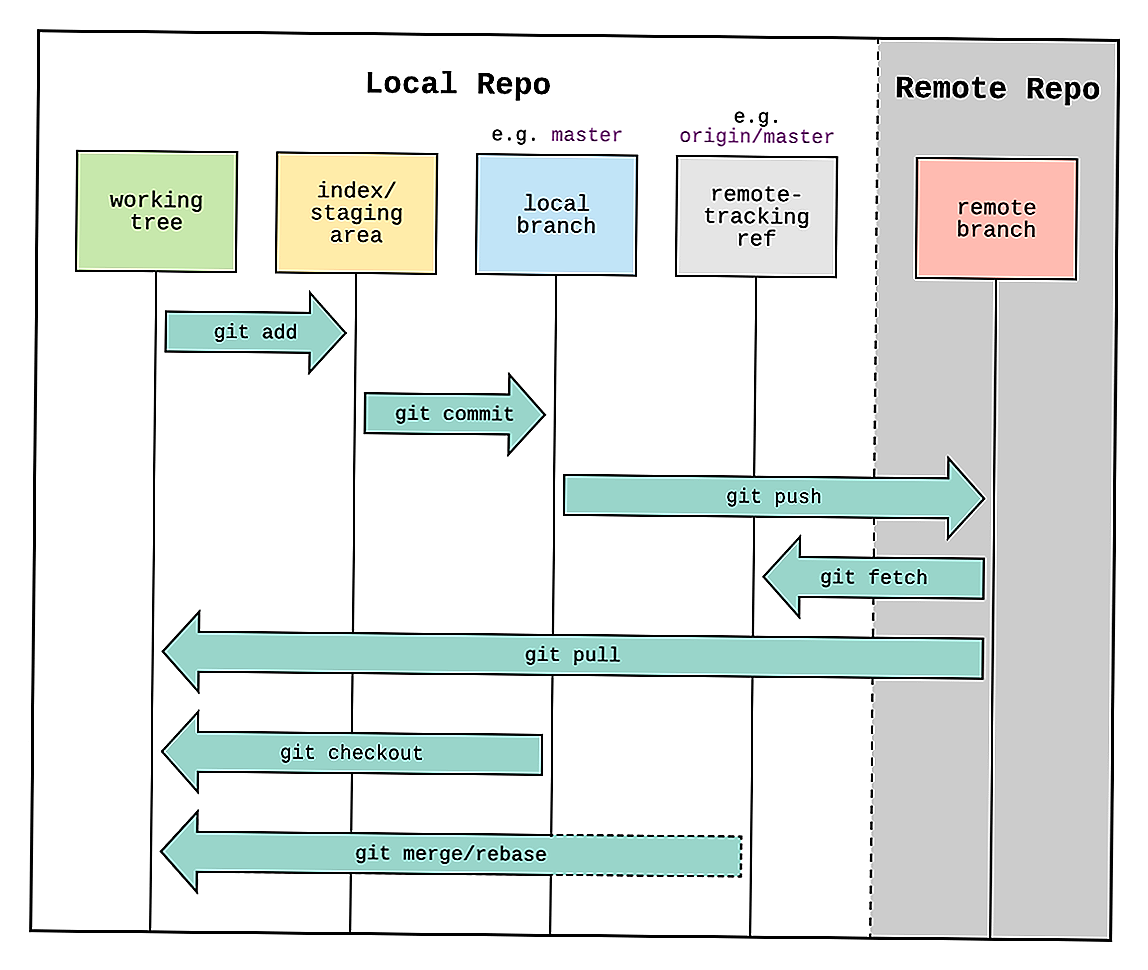
Репозиторий создан:



Преимущества создания системного репозитория в GitHub при работе в команде включают:

* Централизованное хранение — все файлы и конфигурации хранятся в одном месте, что облегчает доступ к ним.
* Общий доступ для команды — участники команды могут легко получить доступ к файлам, клонировать репозиторий и использовать его в своей работе.
* Управление версиями и история изменений — GitHub обеспечивает контроль версий, позволяя отслеживать изменения, вносить комментарии к ним и возвращаться к предыдущим версиям при необходимости.

**Задача 1: Создайте потоковую диаграмму для типичного рабочего процесса разработки программного обеспечения с использованием GitHub, от начала работы до завершения**

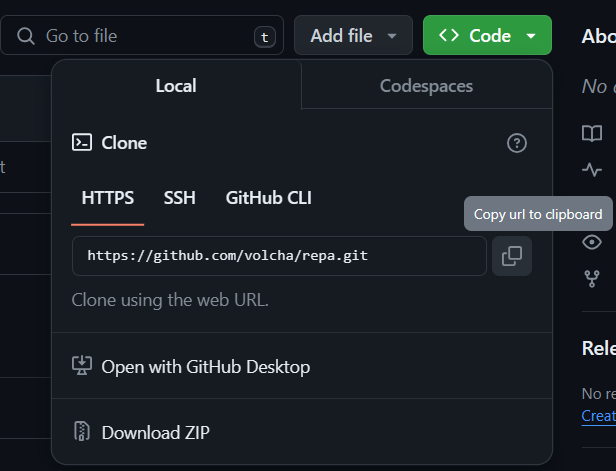


**Задача 2: Проведите практическое упражнение по созданию локального репозитория на вашем компьютере, связанного с удалённым репозиторием на GitHub, и продемонстрируйте основные этапы этого процесса**

Заходим в папку, куда будем клонировать удалённой репозиторий, с помощью командной строки



Копируем адрес репозитория, созданного в вопросе 5



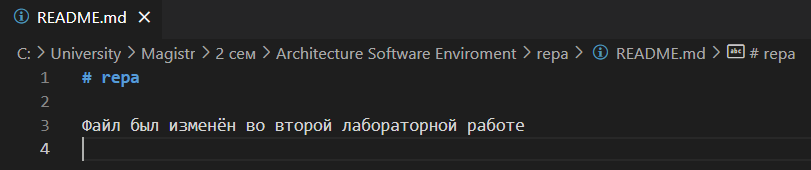
В терминале вводим строку, чтобы склонировать репозиторий локально



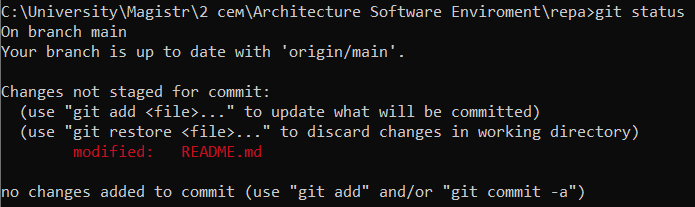
Провалимся внутрь проекта



В папке лежит только README файл, добавим туда информацию



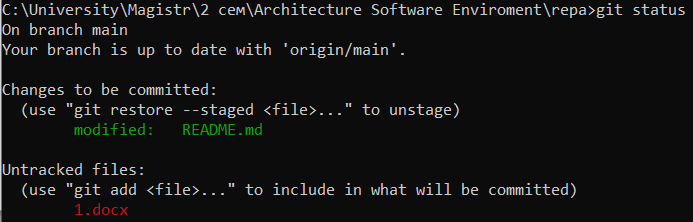
Git пишет, что файл был изменён



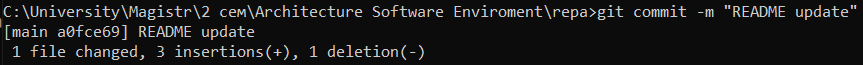
Фиксируем, что нам нужно изменить именно файл README



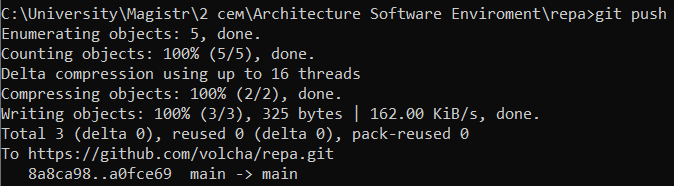
Также добавим ещё текстовый файл, чтобы показать, что он не был зафиксирован, а значит не будет закомичен

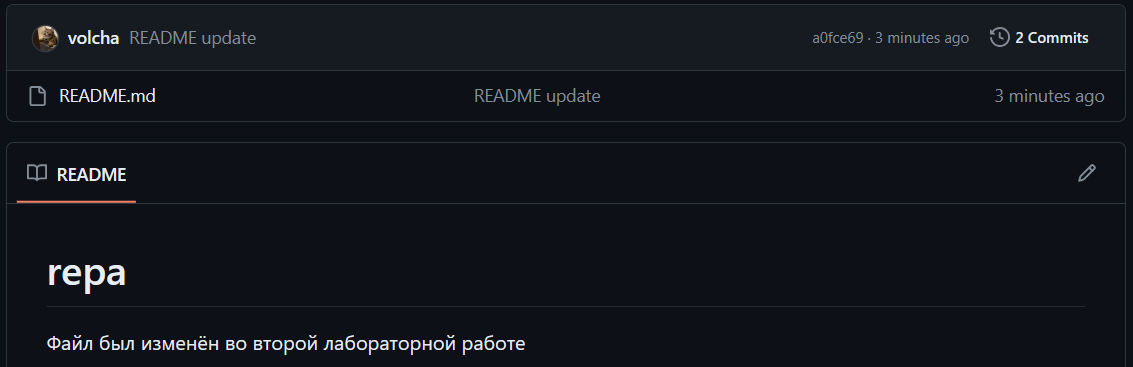


Закоммитим



И запушим на GitHub





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

1. **Что представляет собой профайл в среде Visual Studio Code (VS Code) и какова его роль при работе с Python?**

Профайл в Visual Studio Code (VS Code) - это набор настроек и параметров, определяющих окружение и поведение редактора для конкретного пользователя. В контексте работы с Python, профайл позволяет настроить среду разработки с учетом индивидуальных предпочтений и потребностей разработчика, включая выбор и настройку расширений для работы с Python, отображение и темы оформления, а также настройку средства автодополнения и подсказок. Позволяет делить данной настройкой с другими пользователями.

1. **Какие виды профайлов существуют в VS Code для работы с Python, и как они различаются по функционалу?**

* Общий профайл (Default Profile) — этот профайл используется по умолчанию и предоставляет базовые настройки для работы с любым языком программирования, включая Python. Он включает в себя основные функции редактора и поддержку расширений.
* Python профайл — для работы с Python можно также создать специальный профайл, который будет содержать дополнительные настройки и расширения, оптимизированные для работы с этим языком. Например, этот профайл может включать в себя расширение Python для Visual Studio Code, интегрированные средства отладки, автодополнение и другие инструменты, специфические для Python.
* Python Data Science — этот профайл в Visual Studio Code предназначен для разработки в области науки о данных с использованием Python. Он включает специальные расширения и инструменты, оптимизированные для работы с данными, их анализа, машинного обучения и визуализации.

Каждый вид профайла в VS Code для работы с Python имеет свои особенности и настройки, которые позволяют адаптировать среду разработки под конкретные потребности и предпочтения разработчика.

1. **В чем заключается значение использования профайлов при распределенной разработке программного обеспечения с использованием Python и VS Code?**

* Использование профайлов при разработке программного обеспечения с Python и VS Code позволяет стандартизировать окружение разработки, обеспечивая единые настройки и конфигурации для всех участников команды.
* Упрощает вход новых разработчиков в проект.
* Уменьшает количество ошибок из-за несовместимых настроек у разных участников

1. **Какова типичная структура профайла в VS Code для работы с Python, и какие основные настройки обычно включаются в такой профайл?**

Профайл в VS Code включает в себя следующие элементы:

* Настройки — в файле settings.json.
* Расширения — список расширений, включённых в текущий профайл.
* Привязки клавиш — в файле keybindings.json.
* Snippets — в файлах {language}.json.
* Пользовательские задачи — в файле tasks.json.

Основные настройки профайла в Visual Studio Code для работы с Python включают:

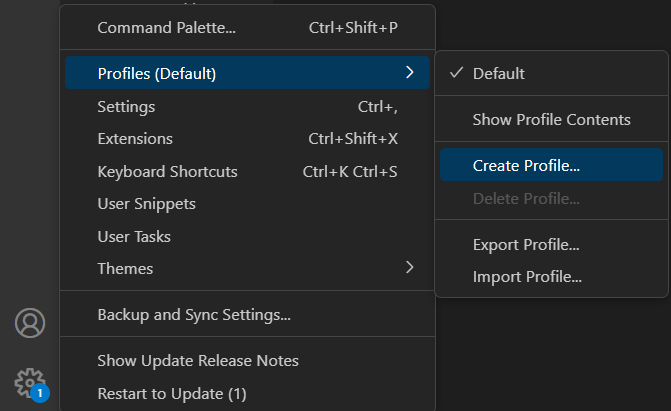
* Интерпретатор Python — определяет версию для проекта.
* Библиотеки и пакеты — определяет зависимости проекта и используемые библиотеки.
* Настройки автодополнения кода — Параметры автодополнения кода, такие как импорт модулей и подсказки по функциям. Это улучшает удобство и быстроту написания кода.
* Форматирование кода — Правила форматирования кода, такие как отступы и расстановка скобок. Это обеспечивает единообразие и читаемость кода.
* Правила проверки кода на наличие ошибок и потенциальных проблем — это помогает выявлять и исправлять ошибки на ранних этапах разработки.
* Запуск и отладка — параметры запуска и отладки кода, такие как точки останова и режим отладки.

1. **Как можно оптимизировать профайл в VS Code для увеличения производительности и удобства работы при разработке на Python?**

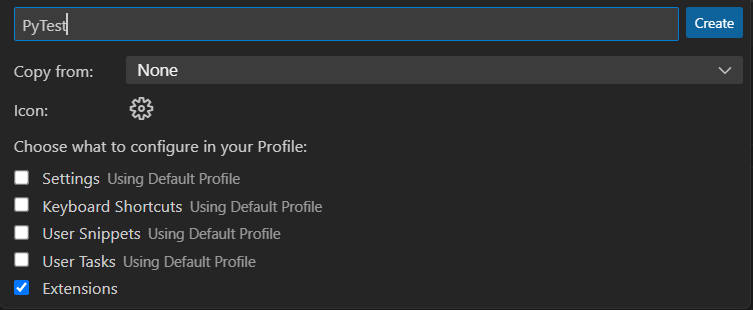
* Выбор эффективного интерпретатора — используйте легковесные и быстрые версии интерпретатора Python, такие как Python 3.8 или более поздние версии.
* Установка необходимых расширений — установите только необходимые расширения для работы с Python, чтобы избежать лишней нагрузки на ресурсы системы и улучшить производительность.
* Использование виртуальных сред — используйте виртуальные среды Python для изоляции зависимостей проекта и оптимизации использования памяти и ресурсов.
* Горячие клавиши — настройте горячие клавиши для часто используемых действий.
* Настройка автоматического форматирования — включите автоматическое форматирование кода при сохранении файлов. Это позволяет поддерживать единый стиль кодирования и улучшает читаемость кода.

**Задача 1: Создайте профиль в VSC для работы с Python**

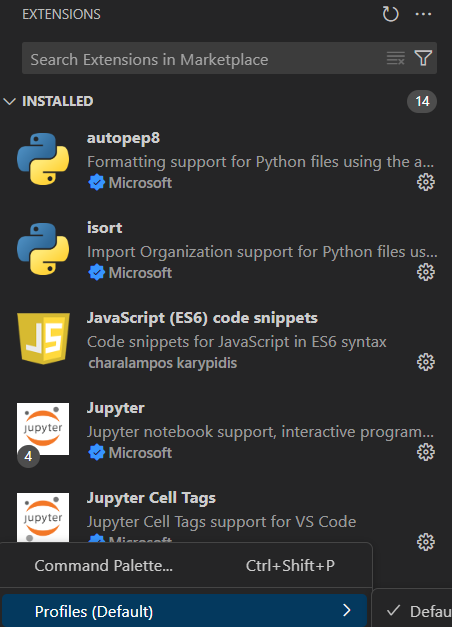
Нажмём на “создать профиль”



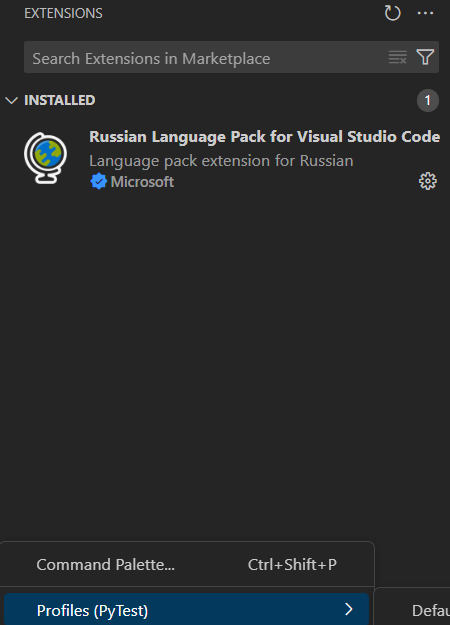
Назовём его PyTest



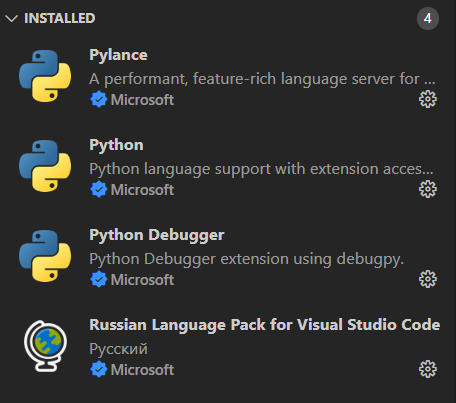
Можно заметить, что в Дефолтном профиле уже скачаны различные разрешения



А в PyTest почти ничего нет



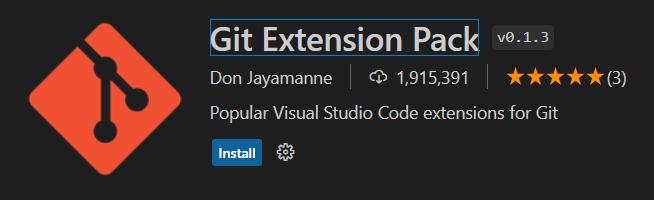
В новый профиль установим нужные для него расширения



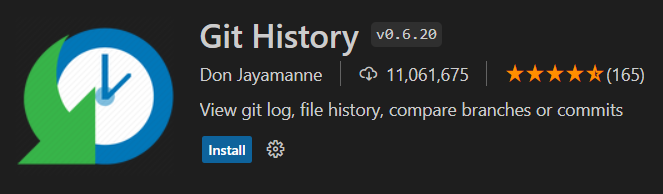
Таким образом, для разных задач и языков можно создавать различные профили, заточенные для конкретных целей

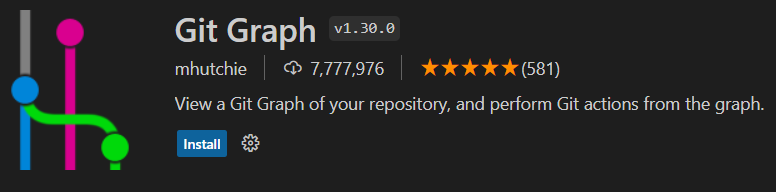
**Задача 2: Разработайте структуру профиля в VSC, которая бы учитывала требования к распределённой разработке, включая настройки для коллективной работы над проектом, контроля версий и совместном редактировании кода**

Для этого можно скачать такие расширения как



Для работы с гитом





Для удобного отслеживания изменений

**Лабораторная 4. Разработка потоковых диаграмм их реализация в среде VS Code.**

1. **Что представляют собой потоковые диаграммы, и какие основные элементы включает типичная потоковая диаграмма?**

Потоковая диаграмма данных (Data Flow Diagram) - это диаграмма, которая отображает потоки данных между системами и базами данных.

DFD Нотация содержит следующие объекты:

* Процесс (Process). Последовательность действий, которые нужно предпринять, чтобы данные были обработаны. Например “Регистрация клиента” или  “Обработать заказ”.
* Внешние сущности (External Entity). Объекты, которые не входят в систему, но являются для неё источником, либо получателем данных. Например клиентский сайт.
* Хранилища данных (Data Storage). Внутренние хранилища данных, в которых сохраняется / извлекается информация, используемая в процессах. Например хранилище заявок или данных о клиентах.
* Поток данных (Data flow). На диаграмме показывает направление движения данных.

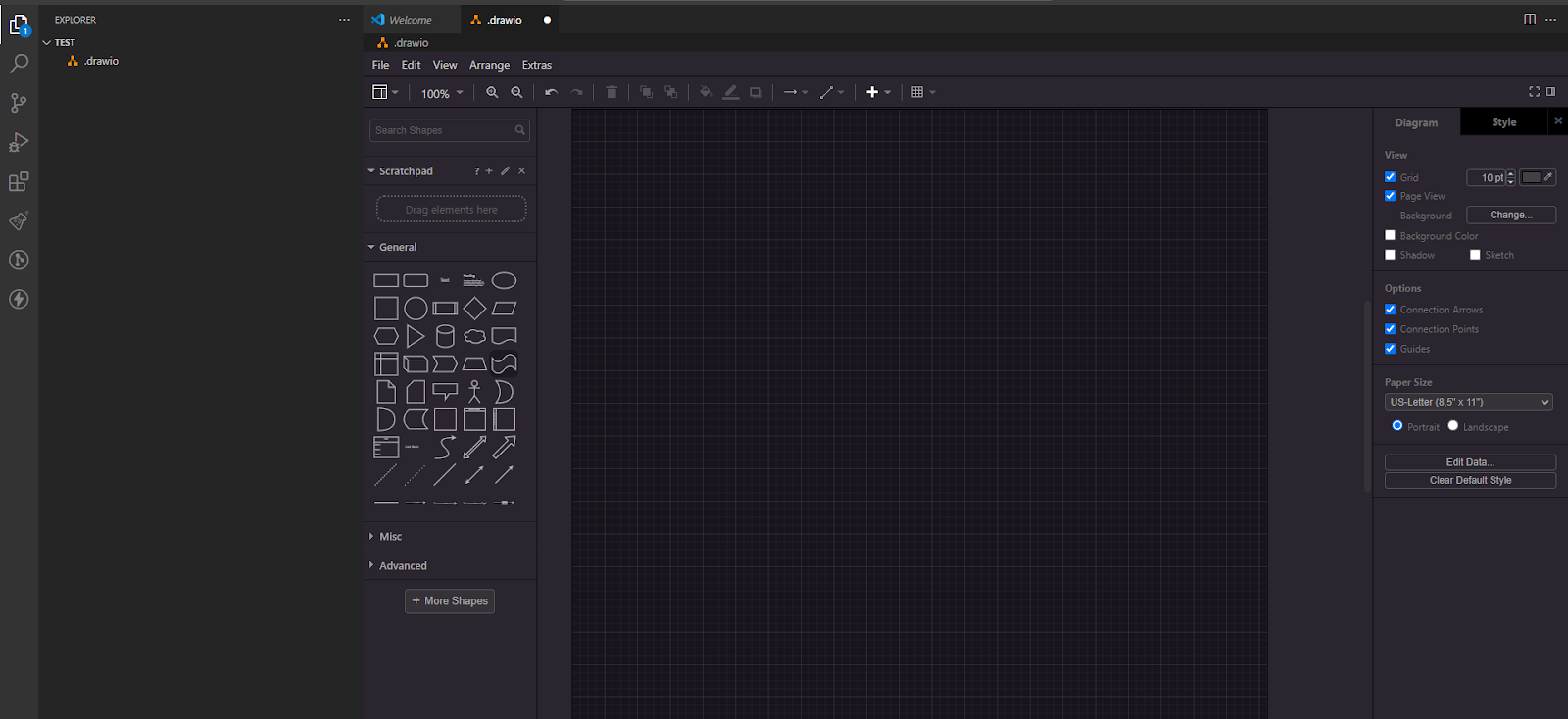
1. **Какова роль разработки потоковых диаграмм в процессе создания программного обеспечения, какие преимущества они предоставляют разработчикам?**

Потоковая диаграмма визуализирует поток данных в системе. Это облегчает понимание, поддержку и разработку системы. С её помощью можно увидеть архитектурные проблемы и все взаимодействия модулей. Также позволяет поделиться своим видением с другими участниками проекта.

1. **Как можно создать потоковую диаграмму в среде Visual Studio Code (VS Code), и какие инструменты или расширения могут быть использованы для этого?**

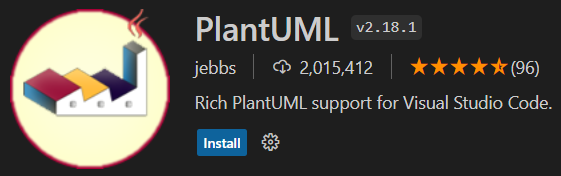
VSC имеет множество инструментов по созданию диаграмм. Диаграммы можно создавать в визуальном редакторе, манипулируя самими объектами диаграммы.

Например, есть расширение “Draw.io Integration”, которое интегрирует сервис draw.io в VSC:



Но, на практике, бывает быстрее и удобнее писать формальные описания диаграмм в виде текста. Самым популярным инструментом является PlantUML, который автоматически генерирует UML диаграммы из кода.

В VS Code есть расширение PlantUML.



1. **Что такое диаграмма взаимодействий в рамках UML (Unified Modeling Language), и какие основные виды взаимодействий она позволяет моделировать?**

UML (Unified Modeling Language - Унифицированный язык моделирования). UML предоставляет набор правил для создания моделей.

Диаграмма взаимодействий в рамках Unified Modeling Language (UML) - это графическое представление взаимодействия между объектами системы.

Основные виды взаимодействий, которые UML позволяет моделировать:

* Диаграмма классов — описывает общую структуру иерархии классов системы, их атрибуты, методы, интерфейсы и взаимосвязи.
* Диаграмма развертывания — показывает программные компоненты и их взаимосвязи. Предлагает наглядное представление, где именно развернут каждый программный компонент.
* Диаграмма состояний — описывает поведение сущности, и то, как оно изменяется в зависимости от внешних и внутренних событий.
* Диаграмма деятельности — описывает процесс от начала и до его конца. Представляет из себя набор операций и то, как каждое действие ведёт к следующему.
* Диаграмма последовательности — показывает последовательность сообщений между объектами в хронологическом порядке.

1. **Каким образом диаграммы взаимодействий могут быть полезны при проектировании и анализе систем на уровне UML?**

Диаграммы взаимодействия имеют следующие плюсы:

* Визуализация взаимодействий между объектами — они позволяют разработчикам наглядно представить, как объекты в системе взаимодействуют друг с другом.
* Понимание порядка выполнения — особенно важно при разработке сложных систем, где порядок операций критичен.
* Анализ требований — диаграммы взаимодействий могут использоваться для проверки того, соответствует ли разрабатываемая система функциональным требованиям.
* Облегчение коммуникации между участниками проекта — диаграммы предоставляют удобный способ общения между разработчиками, аналитиками, тестировщиками и другими заинтересованными сторонами.
* Обнаружение проблем в проектировании — при рассмотрении диаграмм взаимодействий можно обнаружить ошибки или недостатки в системе.
* Использование в тестировании — диаграммы взаимодействий могут служить основой для разработки тестовых сценариев.

**Задача: Постройте диаграмму взаимодействий в UML, моделирующую обмен сообщениями между пользователем и системой при регистрации нового пользователя в веб-приложении.**

@startuml Процесс регистрации пользователя

actor "Клиент" as Client

entity "Сервис регистрации" as Service

database "База данных" as Database

Client -> Service: Отправить данные регистрации (имя, email, пароль)

activate Service

    Service -> Database: Проверить, существует ли пользователь

    activate Database

        Database -> Database: Поиск в базе данных

        Database -> Service: Результат проверки

    deactivate Database

    alt Если пользователь уже существует

        Service -> Client: Ошибка: пользователь уже существует

    else Если новый пользователь

        Service -> Database: Сохранить нового пользователя

        activate Database

            Database -> Database: Запись данных пользователя

            Database -> Service: Подтверждение сохранения

        deactivate Database

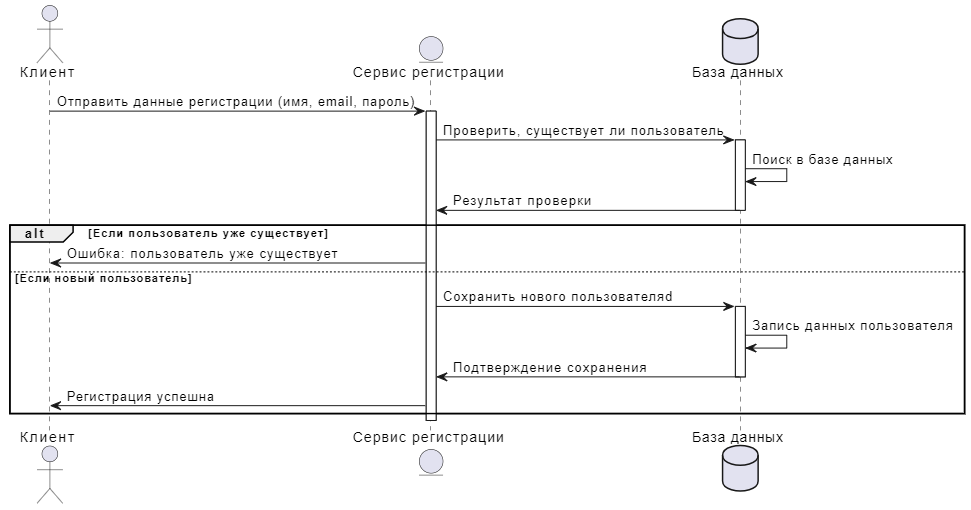
        Service -> Client: Регистрация успешна

    end

deactivate Service

@enduml

Зарендерим диаграмму, нажав alt+D



Здесь видно главное преимущество PlantUML по сравнению с ручным построением диаграммы: процесс описан в виде кода, его можно легко модифицировать, даже для сложных диаграмм.

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

1. **Как создать новую ветвь (branch) в репозитории на GitHub c использованием командной строки и через веб-интерфейс?**

С использованием командной строки:

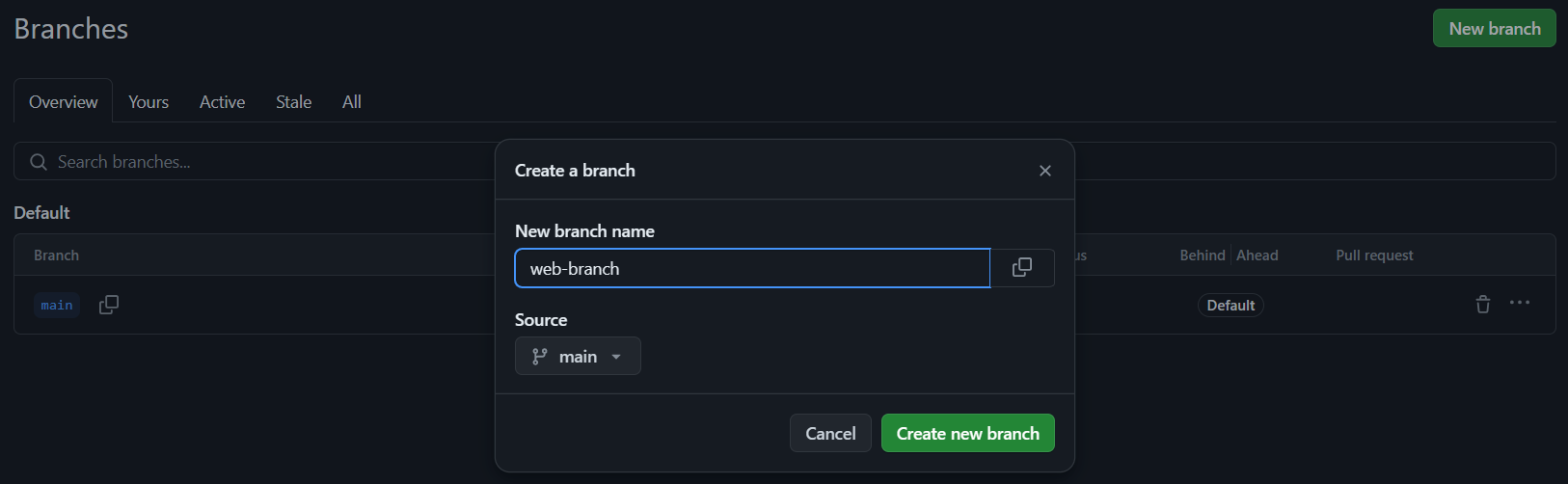


Через веб-интерфейс:

Зайти в Branch



New branch --> имя ветви --> Create new branch



1. **Какие основные принципы следует учитывать при работе с ветками Git для эффективной распределённой разработки?**

* Частые и небольшие коммиты — чтобы упростить отслеживание изменений, сократить вероятность конфликтов при слиянии и обеспечить более гибкую историю разработки.
* Основная ветка (обычно называется main или master) — должна отражать стабильное состояние проекта. Все изменения должны быть интегрированы в основную ветку только после тщательного тестирования и проверки качества кода.
* Создание отдельных веток для функциональности — для каждой новой функциональности или задачи создайте отдельную ветку.
* Регулярное слияние и обновление — регулярно сливайте изменения из основных веток в рабочие ветки, чтобы поддерживать их актуальность и избегать накопления большого количества изменений.

1. **Как осуществить слияние (merge) ветвей в Git на GitHub, чтобы объединить изменения из одной ветви в другую?**

git checkout <имя ветки, в которую нужно залить код (например, develop)>

git merge <имя\_ветки, откуда нужно взять код (например, origin/develop)>

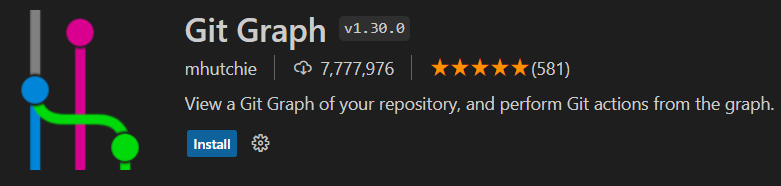
1. **В чём заключается понятие конфликтов слияния (merge conflicts) при объединении ветвей, и как их решить с помощью среды VSC?**

Конфликты слияния (merge conflicts) возникают при попытке объединить две ветки, когда Git не может автоматически определить, какие изменения следует применить. Это может произойти, когда две ветки внесли изменения в одну и ту же часть файла или когда одна ветка удалила файл или строку, которую другая ветка изменила.

Для решения конфликтов слияния в среде Visual Studio Code (VS Code) следуйте этим шагам:

1. **Открытие файла с конфликтом**: В редакторе VS Code перейдите в раздел "Source Control" (или используйте команду "Ctrl+Shift+G"), где будут отображены файлы с конфликтами. Щелкните по файлу с конфликтом для открытия его в редакторе.
2. **Визуальное разрешение конфликта**: VS Code поможет вам визуально разрешить конфликты, отображая их в специальном режиме. Вам будет предложено выбрать, какие изменения сохранить: из текущей ветки (HEAD), из ветки, которую вы сливаете (Incoming Change), или оба варианта. Выберите нужные изменения, удалив лишние метки "<<<<<<<", "=======" и ">>>>>>>", а затем сохраните файл.
3. **Пометка конфликтов как разрешенных**: После разрешения конфликтов отметьте файл как разрешенный для слияния, нажав на кнопку "Mark as Resolved" в панели инструментов Source Control.
4. **Завершение слияния**: После разрешения всех конфликтов и пометки файлов как разрешенных завершите слияние, выполните коммит и, если необходимо, отправьте изменения на удаленный репозиторий.
5. **Какие инструменты предоставляет VSC для удобного отслеживания и управления ветками Git при работе над проектом на GitHub?**

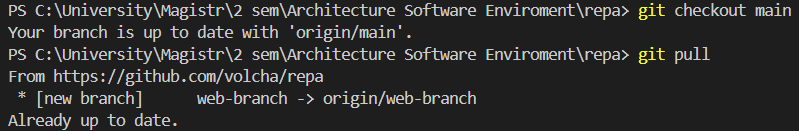
Такое расширение как Git Graph. Это визуальное представление истории коммитов, ветвей и слияний в репозитории Git



**Задача 1: Создайте новую ветвь в вашем репозитории на GitHub, добавьте в неё изменения через VSC, а затем выполните слияние этой ветви с основной ветвью master/main**

Новая ветка в репозитории “web-branch” была создана в вопросе 1.

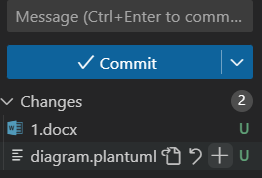
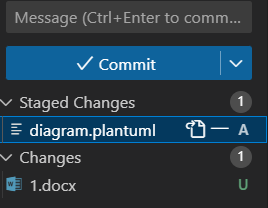
Загружаем ветку



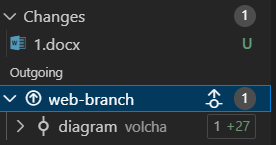
Переключаемся на неё



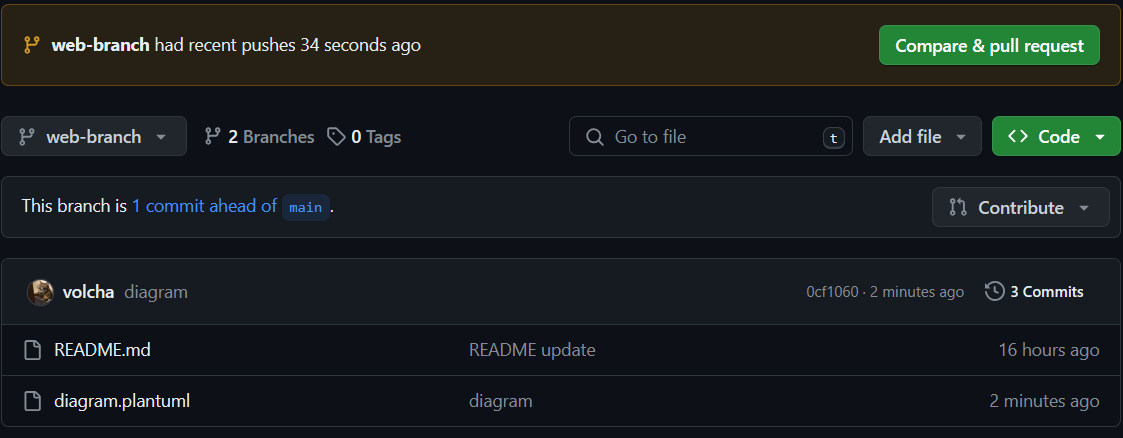
Выбираем код с диаграммой для добавления его в новую ветку

Делаем коммит



И пушим в GitHub

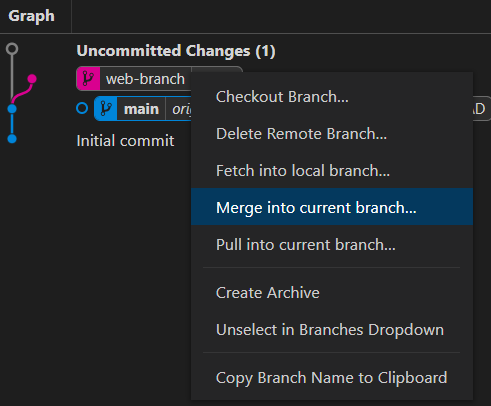


Файл добавился в GitHub

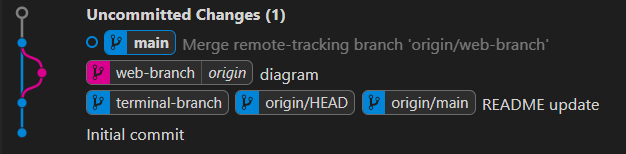
Чтобы сделать слияние с main можно: перейти в ветку main



И нажать

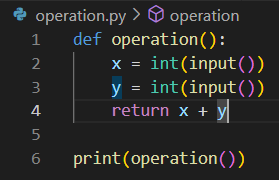


Ветки слиты

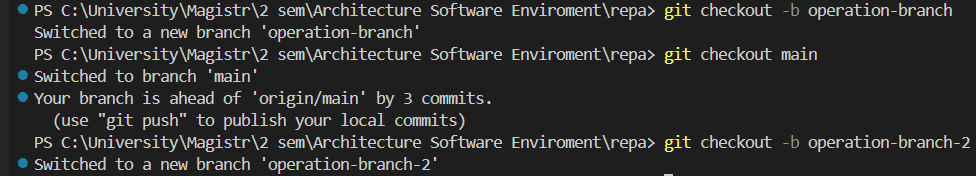


**Задача 2: Разрешите конфликты слияния двух веток в вашем проекте, используя возможности сравнения и редактирования кода в VSC для обеспечения успешного объединения изменений**

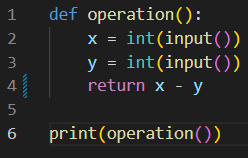
Добавим файл operation.py в main ветку и закоммитим его



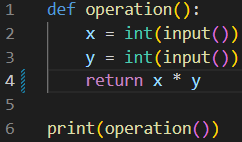
Создаём 2 ветки

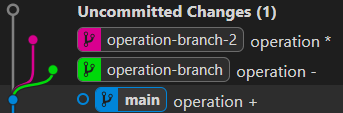


Перейдём в первую и изменим там код и закомитим его в этой ветке

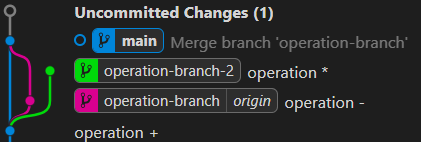


Изменим код во второй ветке

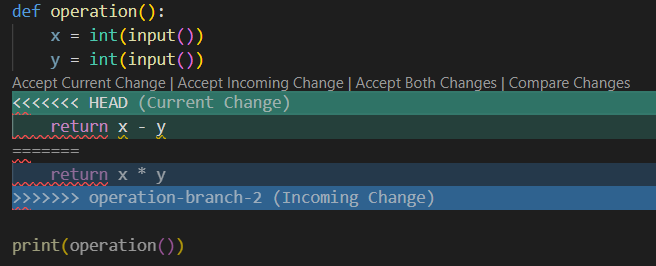




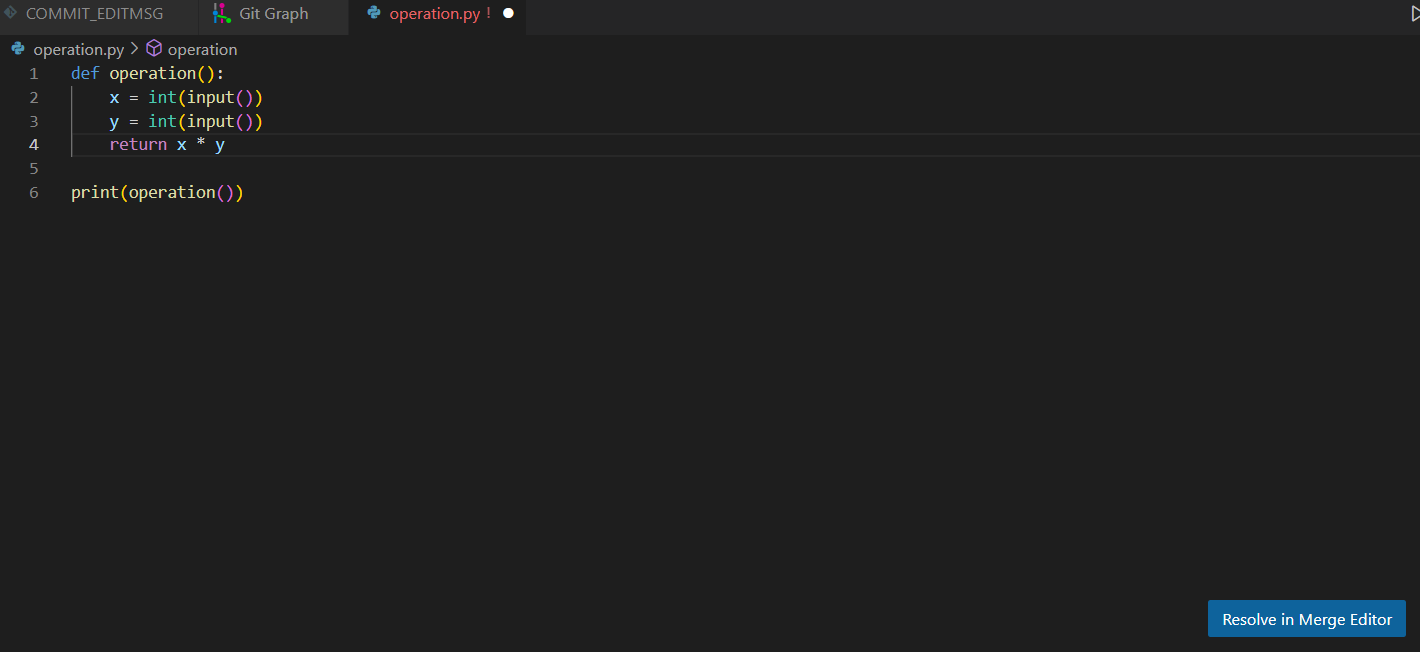
merge изменения из первой ветки в main

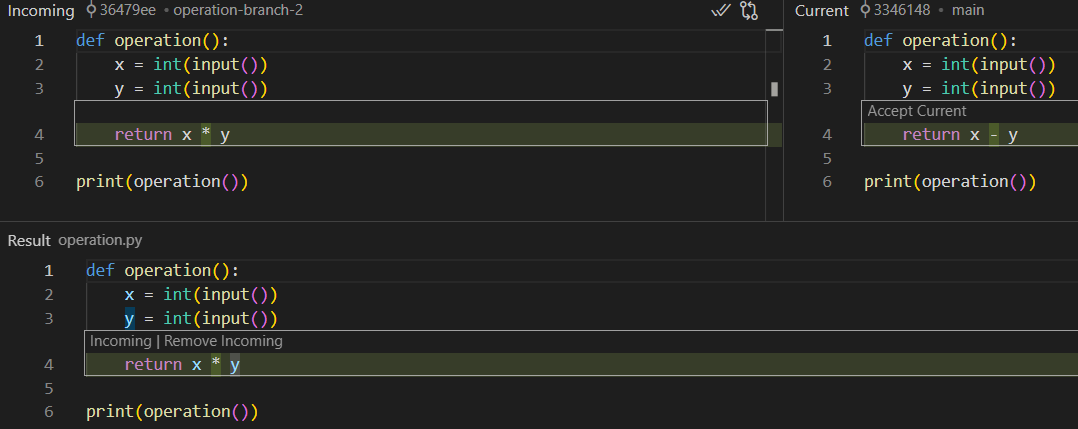


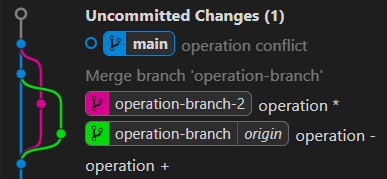
merge изменения из второй ветки в main. Возникает конфликт:



Оставим изменения из второй ветки







**Лабораторная 6. Файлы конфигураций (settings), создание User Task, настройка редактора VS Code.**

1. **В чём разница между файлами settings.json и settings.override.json?**

settings.json:

Этот файл содержит основные настройки приложения, обычно включает все необходимые параметры и значения, которые требуются для работы приложения, его изменять нельзя.

settings.override.json:

Этот файл используется для переопределения настроек, указанных в settings.json. Часто используется для тестирования, чтобы не влиять на основную конфигурацию. Параметры, указанные в этом файле, имеют приоритет над соответствующими параметрами в settings.json. То есть, если одно и то же значение присутствует в обоих файлах, будет использовано значение из settings.override.json.

1. **Как создать новую пользовательскую задачу в VS Code?**

Создание новой пользовательской задачи (task) в Visual Studio Code (VS Code) позволяет автоматизировать различные процессы, такие как сборка, тестирование и запуск скриптов.

* Откройте меню Terminal и выберите Configure Tasks..., затем Create tasks.json file from template.
* Выберите подходящий шаблон или начните с пустого файла.
* В открывшемся файле tasks.json добавьте новую задачу

1. **Какие настройки редактора VS Code можно настроить для повышения производительности при работе с файлами конфигурации?**

Чтобы повысить производительность при работе с файлами конфигурации в VS Code, можно настроить следующие параметры редактора в файле settings.json:

* Автозавершение

"editor.quickSuggestions": {

"other": true,

"comments": false

}

Включает быстрые подсказки только для кода, исключая комментарии

* Минимизация использования памяти

"files.trimTrailingWhitespace": true,

"files.trimFinalNewlines": true,

"files.insertFinalNewline": false

Автоматическое удаление лишних пробелов и пустых строк

* Ограничение числа открытых файлов:

"files.maxMemory": 4096

Установите эту настройку для ограничения количества памяти, используемой редактором для хранения открытых файлов.

* Настройки для работы с большими проектами:

"files.exclude": {

"\*\*/.git": true,

},

"search.exclude": {

"\*\*/.git": true,

}

Исключение ненужных файлов и папок из поиска и отображения.

1. **Как указать путь к файлу конфигурации для пользовательской задачи?**

Чтобы указать путь к файлу конфигурации (settings) для пользовательской задачи в VS Code, можно использовать переменные пути в файле tasks.json в параметре args.



${workspaceFolder} используется для указания корневой папки рабочего пространства.

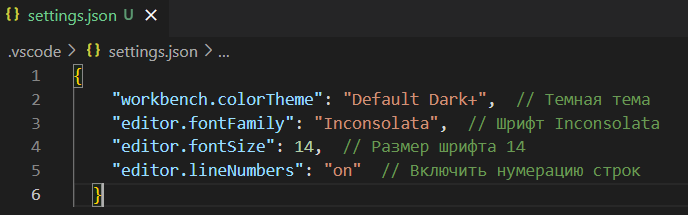
path/to/your/config/file.json — путь к вашему файлу конфигурации относительно корневой папки рабочего пространства.

1. **Можно ли использовать переменные среды (env) в файлах конфигурации VS Code?**

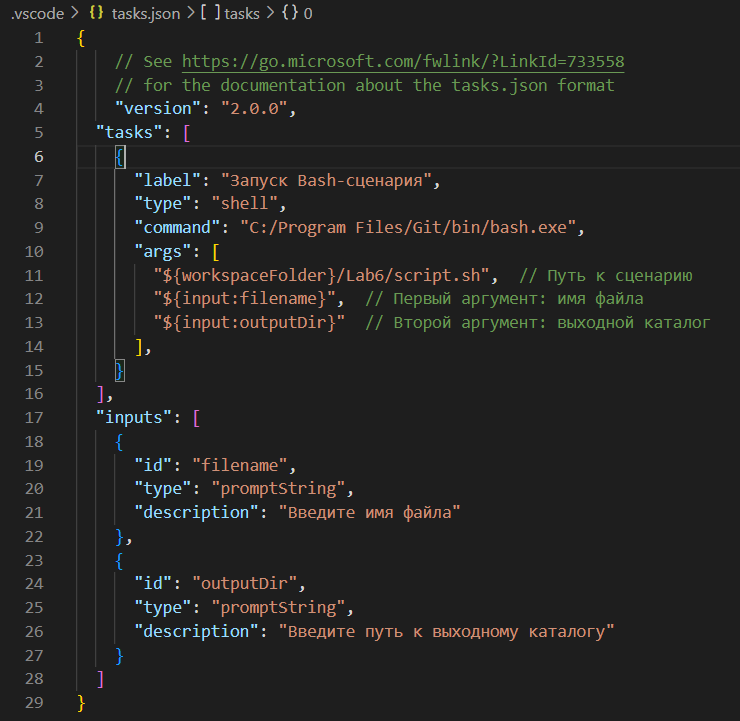
Да, в файлах конфигурации VS Code можно использовать переменные среды. Это позволяет динамически задавать значения на основе текущей среды. Переменные среды могут быть использованы в файлах, таких как tasks.json, settings.json. Например ${env:DATABASE\_URL} - URL подключения к базе данных

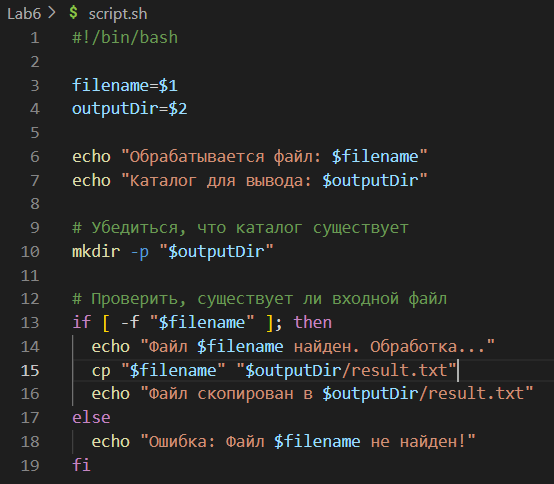
**Задача 1: Создайте файл конфигурации в VS Code, который будет определять следующие настройки редактора:**

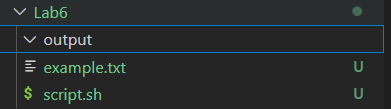
* **Тема: Dark**
* **Шрифт: Inconsolata**
* **Размер шрифта: 14**
* **Включить нумерацию строк**

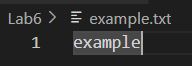


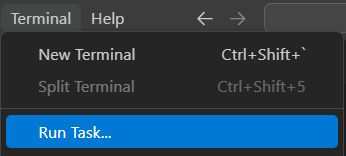
**Задача 2: Создайте пользовательскую задачу в VS Code для запуска сценария bash. Задача должна выполняться в терминале, и ей необходимо передавать два аргумента: имя файла и выходной каталог**

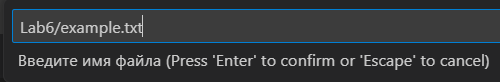


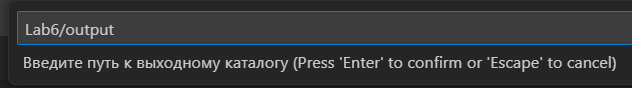


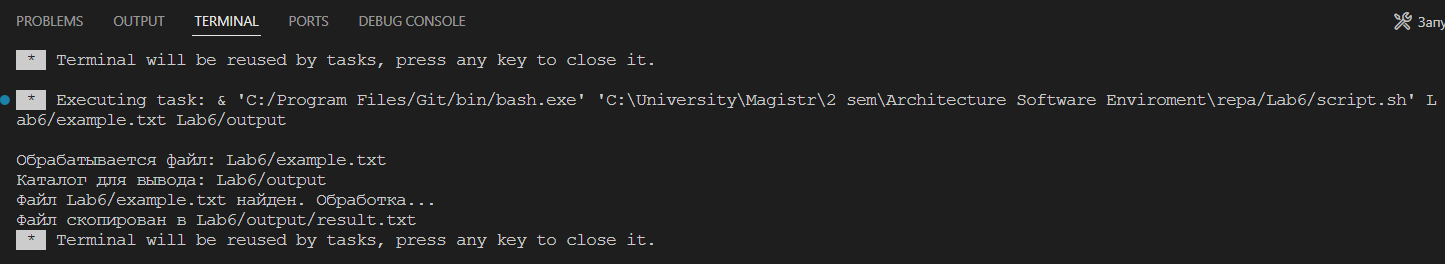


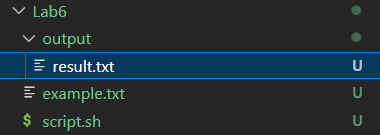














**Лабораторная 7. Создание и работа в профиле для формирования документации.**

1. **Какие расширения для VS Code наиболее полезны при работе с документацией?**

Для работы с документацией в VS Code полезны расширения, которые улучшают форматирование и управление текстом. Например, **Markdown All in One** ускоряет написание Markdown, добавляя например создание оглавлений и предварительный просмотр. **Prettier** нужен для автоформатирования, сохраняя единообразие стиля. Для управления ссылками и заметками можно использовать **Markdown Link Check**, что упрощает создание и проверку связанных документов. **TODO Highlight** выделяет ключевые слова TODO, FIXME и другие в коде и документации, делая их более заметными и помогая отслеживать незавершенные задачи и замечания в документации.

1. **Какие функции VS Code облегчают процесс написания и форматирования текста в документации?**

* Автодополнение: Ускоряет вставку Markdown-тегов.
* Подсветка синтаксиса: Делает структуру текста более читаемой.
* Предпросмотр Markdown: Показывает форматированный текст в реальном времени.
* Автосохранение: Предотвращает потерю данных.
* Интеграция с Git: Упрощает управление версиями и совместную работу.
* Сниппеты: позволяют быстро вставлять часто используемые фрагменты текста.

1. **Как настроить среду разработки в VS Code для работы с различными форматами документации, такими как Markdown, reStructuredText и HTML?**

* Markdown

Установить расширение Markdown All in One для автодополнения и предпросмотра. Настройть Prettier для автоформатирования Markdown. Включить предпросмотр с помощью встроенной функции (клавиша Ctrl+Shift+V).

* reStructuredText

Установить расширение reStructuredText для подсветки синтаксиса и проверки. Использовать команду "Preview" для просмотра reStructuredText -файлов.

* HTML

Установить HTML Language Support и Live Server для подсветки синтаксиса и локального сервера. Настроить Emmet для ускорения написания HTML-кода. Запускайть предпросмотр в браузере через Live Server (правый клик → "Open with Live Server").

1. **Какие инструменты и функции предоставляет VS Code для проверки правописания и стиля текста в документах?**

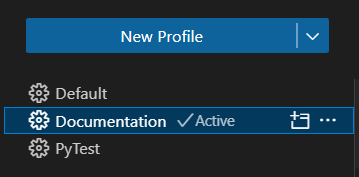
* Code Spell Checker: Проверяет орфографию в текстах, коде и комментариях. Поддерживает настраиваемые словари.
* Spell Right: Обеспечивает проверку правописания на нескольких языках, включая автоматическое определение языка.
* Расширение Grammarly добавляет проверку грамматики, пунктуации и стиля в документах.
* Markdownlint проверяет соответствие Markdown-документов стилевым рекомендациям (например, пустые строки, длина строк).
* Write Good анализирует текст и указывает на стилистические ошибки (пассивный залог, клише и т.п.).
* Сниппеты позволяют вставлять часто используемые фразы и шаблоны, что снижает вероятность ошибок в текстах.

1. **Какие возможности предоставляет VS Code для совместной работы над документацией и отслеживания изменений?**

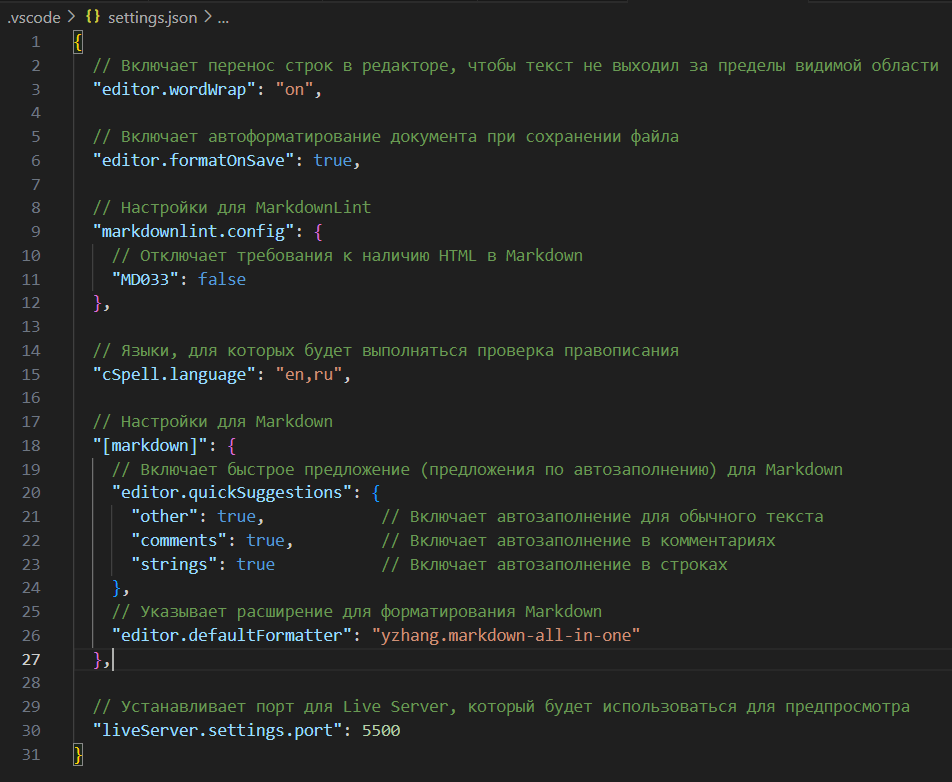
VS Code поддерживает совместную работу над документацией через интеграцию с **Git** и сервисами, такими как GitHub и GitLab, что позволяет отслеживать изменения, выполнять коммит и проводить ревью кода. Функция **Live Share** дает возможность нескольким пользователям редактировать документы в реальном времени и обмениваться комментариями. Встроенная система комментариев в коде и возможность создавать Pull Requests напрямую из редактора упрощают коммуникацию. История версий позволяет вернуться к предыдущим изменениям, а уведомления об обновлениях помогают координировать работу команды.

**Задача 1: Создайте шаблон профиля для документации в VS Code, включающий настройки среды, стандарты оформления, а также предустановленные расширения для работы с различными форматами документов**

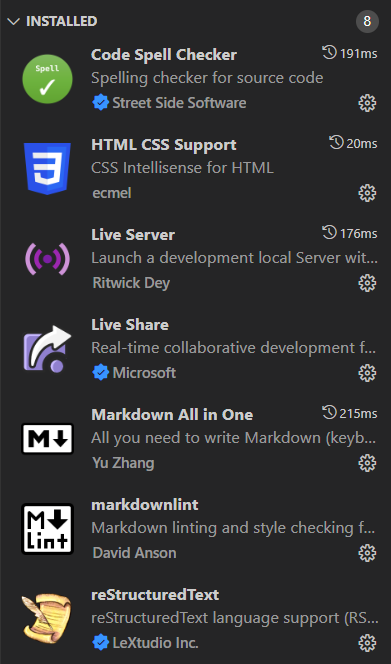
Создать новый профиль Documentation



Настройка среды в settings.json:

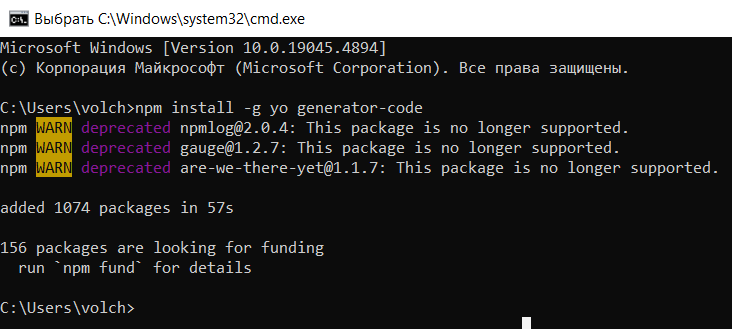


Для этого профиля используются вот такие расширения:

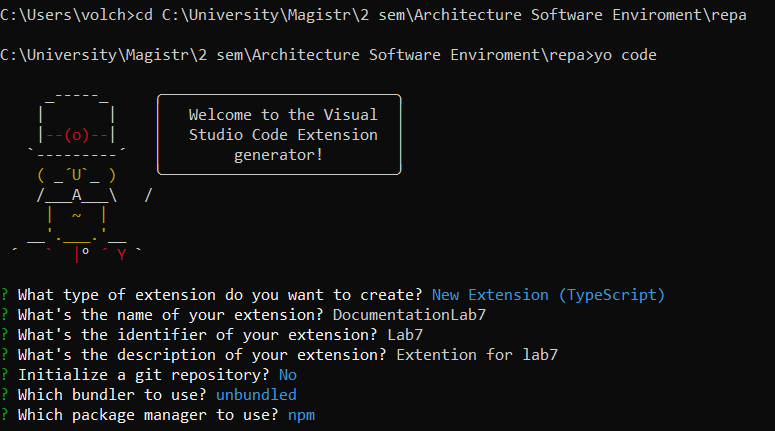


**Задача 2: Напишите расширение для VS Code, которое автоматически генерирует содержание (ТОС) для документов на основе структуры заголовков и разделов**

Установка Yeoman генератора VS Code:

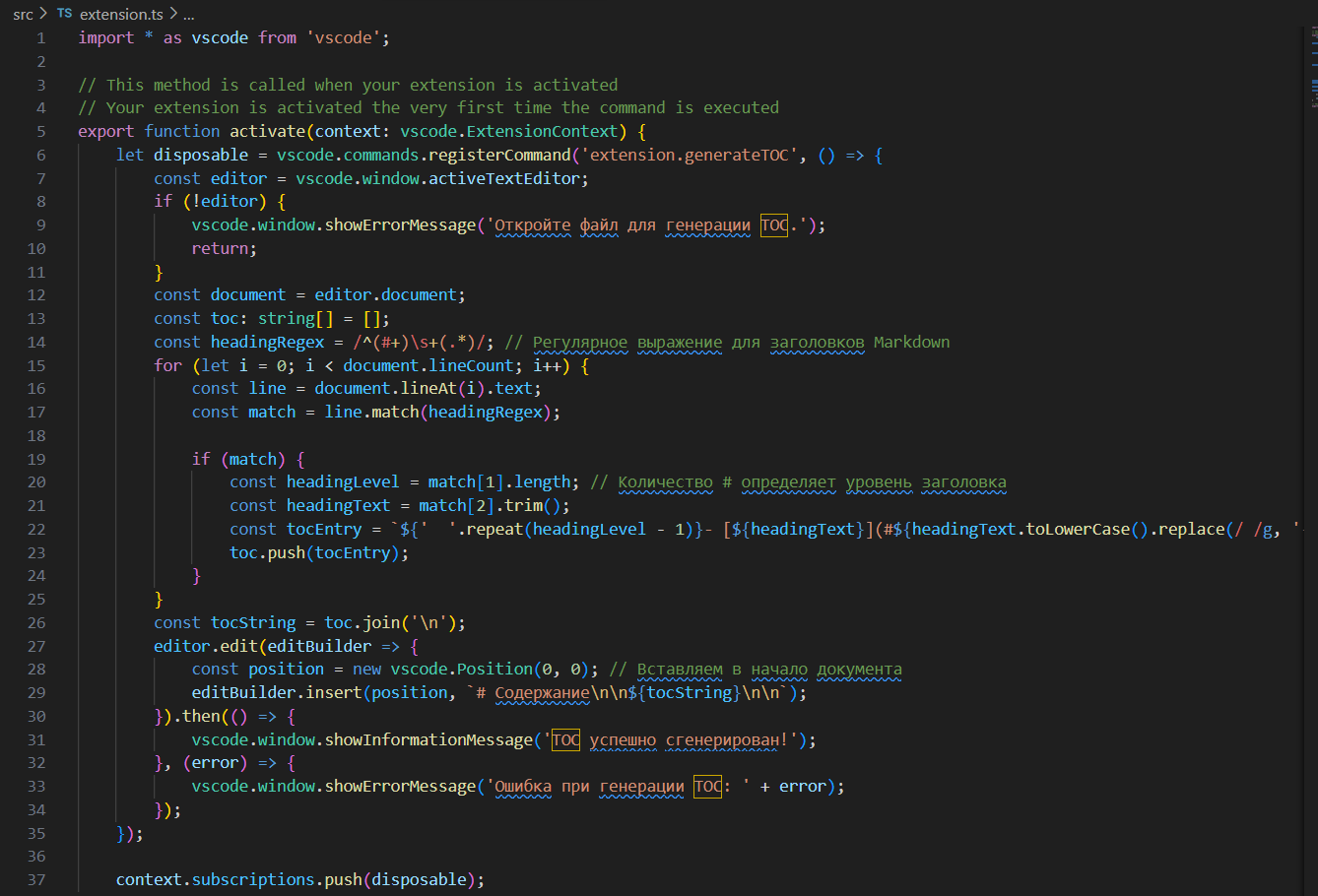


Запустить генератор:





В файле src/extension.ts:



Настройка package.json:



Теперь тестируем расширение:

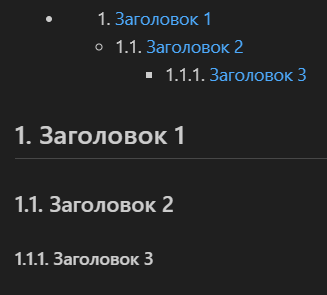
Запустить расширение, нажав F5



Создать файл Markdown и добавить несколько заголовков

Открыть командную палитру (Ctrl+Shift+P) и ввести " Generate TOC". Нажать Enter.

Содержание сгенерировано



**Лабораторная 8. Создание и работа в профиле VS Code для Data Science.**

1. **Какие есть полезные расширения для VS Code для работы в области Data Science?**

Для работы в области Data Science в VS Code есть несколько полезных расширений, которые делают процесс более удобным и продуктивным. В первую очередь **Python** и **Jupyter** — они обеспечивают полноценную работу с Python-кодом и ноутбуками прямо в редакторе. **Pylance** добавит улучшенное автодополнение и типизацию, что упростит написание сложного кода.

Для визуализации данных удобно использовать **Plotly Visualizations** и инструменты для работы с **pandas DataFrame**. В случае работы с базами данных поможет **SQLite Viewer**. Для совместной работы и контроля версий можно использовать **Live Share**, а для улучшения качества кода подойдёт **Pylint**.

1. **Как настроить среду разработки в VS Code для работы с Jupiter Notebook?**

Чтобы настроить среду для работы с Jupyter Notebook в VS Code, сначала нужно установить расширения Python и Jupyter из магазина расширений. Затем убедиться, что Python установлен на вашем компьютере, и при необходимости создайть виртуальное окружение.

Открыть или создать новый файл с расширением .ipynb, редактор автоматически активирует режим Jupyter. Вы сможете запускать ячейки, просматривать результаты прямо в коде и подключать различные ядра (kernels) для выполнения. Для полноценной работы убедитесь, что все необходимые библиотеки, такие как numpy или pandas, установлены в вашем окружении.

1. **Какие функции VS Code упрощают работу с данными и визуализацией?**

С помощью **Jupyter Notebook** можно удобно запускать и тестировать код в ячейках, а результаты (например графики) сразу отображаются в редакторе.

**Data Preview** позволяет быстро просматривать содержимое CSV и JSON в табличном виде.

Для визуализации данных расширения поддерживают интерактивные графики прямо в ноутбуках, включая библиотеки **Plotly** и **Matplotlib**.

При работе с большими наборами данных можно также использовать встроенные терминалы и инструменты для управления базами данных, например **SQLite Viewer**, для локальной работы с SQL-запросами и результатами.

1. **Какие возможности предоставляет VS Code для совместной работы над проектами в области Data Science?**

VS Code поддерживает совместную работу над проектами с помощью **Live Share**, позволяя нескольким участникам одновременно редактировать код, запускать ячейки Jupyter и комментировать в реальном времени. Это особенно удобно для парного программирования и анализа данных.

**GitLens** облегчает командную работу через Git, предоставляя визуализацию истории изменений, сравнение версий и отслеживание авторства кода. С помощью встроенной интеграции с GitHub можно легко делиться ноутбуками, управлять ветками и решать конфликты.

1. **Какие средства предоставляет VS Code для управления зависимостями и виртуальными средами в Python?**

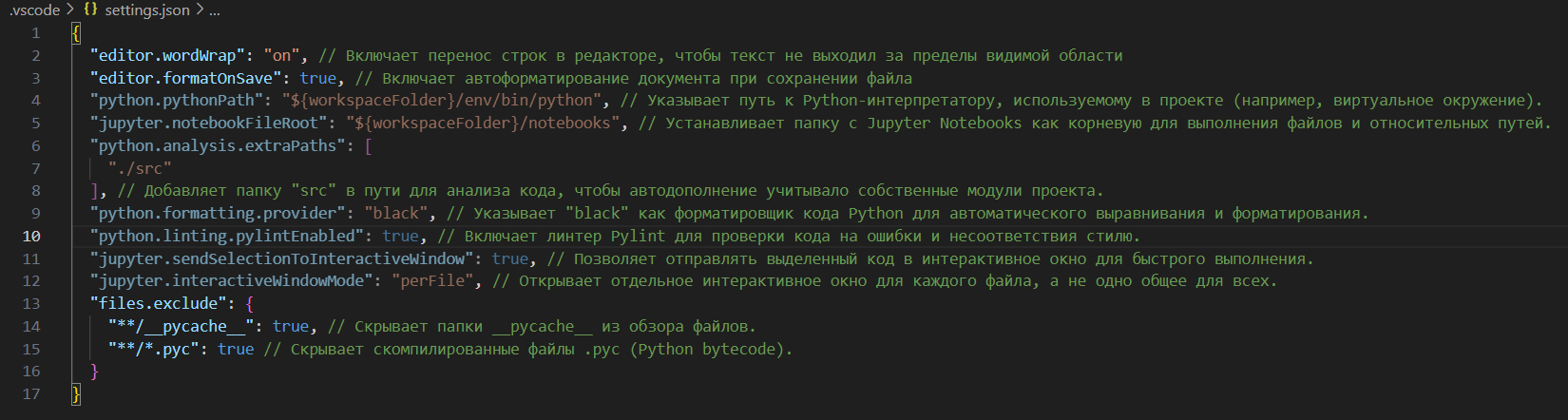
Создание виртуальных окружений: Встроенный терминал позволяет создавать среды с помощью команд, например, python -m venv venv или conda create -n myenv python=3.9.

Автоматическое распознавание окружений: VS Code автоматически обнаруживает виртуальные среды и предлагает использовать их при открытии проекта.

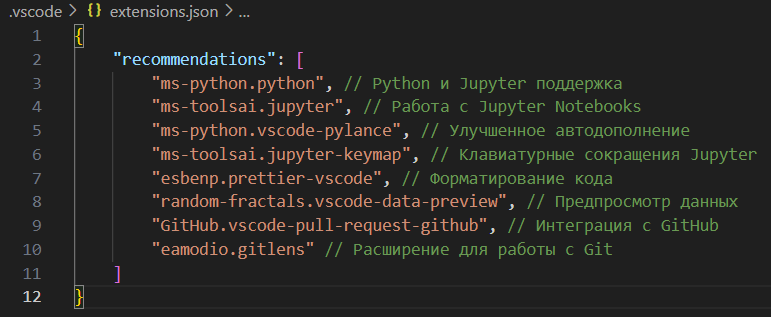
Автоматическая установка пакетов: При запуске кода редактор предлагает установить отсутствующие библиотеки, если они не найдены в текущем окружении.

**Задача 1: Создайте шаблон профиля для Data Science проектов в VS Code, включающий настройки рабочей области, расширения и конфигурационные файлы**

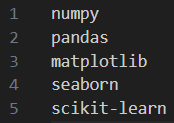
Настройки рабочей области для удобной работы с Jupyter и Python в .vscode/settings.json



Список рекомендуемых расширений для Data Science проекта в .vscode/extensions.json



Зависимости, необходимые для проекта в requirements.txt



**Задача 2: Напишите скрипт или расширение для VS Code, которое автоматически собирает и визуализирует базовую статистику о данных из выбранного файла**

Код скрипта analyze\_data.py

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import os

def load\_data():

    """Загружает данные из файла statistics.csv в папке Lab8."""

    file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'statistics.csv')

    if not os.path.exists(file\_path):

        raise FileNotFoundError(f"Файл {file\_path} не найден.")

    return pd.read\_csv(file\_path)

def visualize\_data(df):

    """Создает базовую визуализацию данных."""

    print("Статистическая сводка:")

    print(df.describe(), "\n")

    # Гистограммы числовых признаков

    df.hist(bins=20, figsize=(15, 10))

    plt.suptitle("Гистограммы числовых признаков")

    plt.show()

    # Корреляционная матрица

    plt.figure(figsize=(10, 6))

    sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")

    plt.title("Корреляционная матрица")

    plt.show()

def main():

    """Главная функция для анализа данных."""

    try:

        df = load\_data()

        visualize\_data(df)

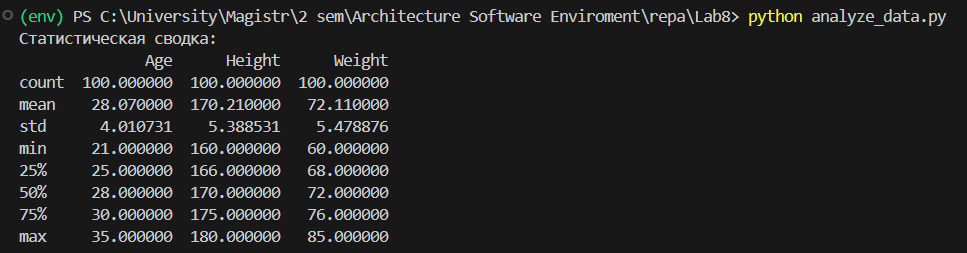
    except Exception as e:

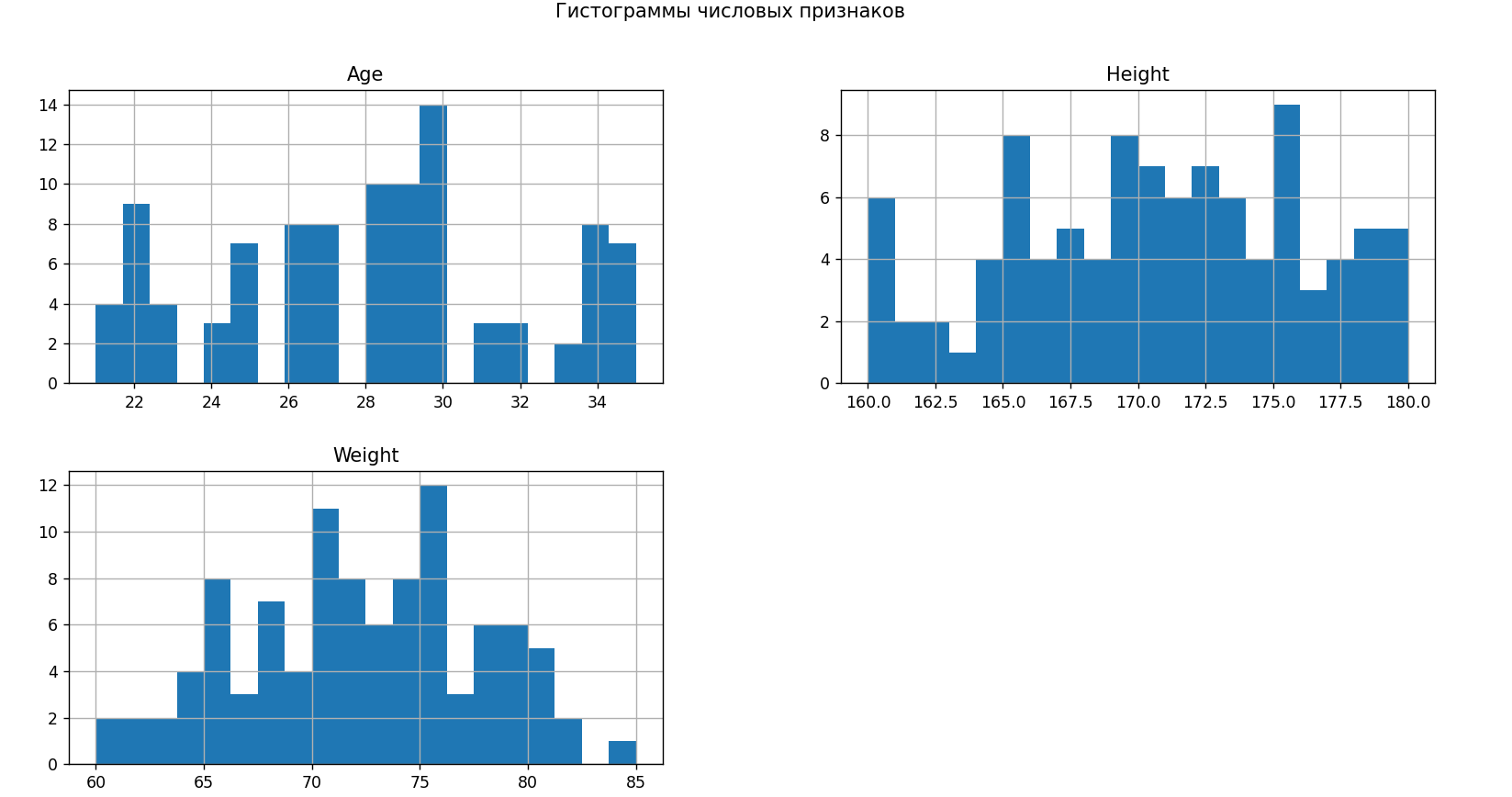
        print(f"Ошибка: {e}")

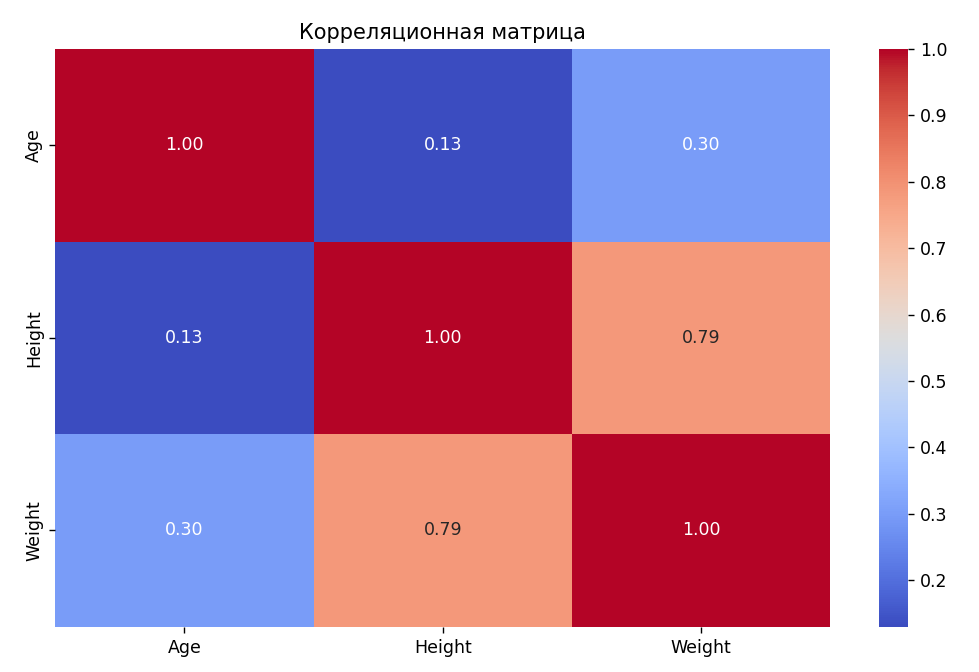
if \_\_name\_\_ == "\_\_main\_\_":

    main()

Результат выполнения:







**Лабораторная 9. Создание и отладка веб-приложений с использованием расширения VS Code для браузера.**

1. **Какие возможности предоставляет расширение VS Code для браузера при разработке и отладке веб-приложений?**

* Редактирование кода: Легкий доступ к редактированию файлов проекта в браузере.
* Поддержка языков: Подсветка синтаксиса и автодополнение для HTML, CSS, JavaScript и других языков.
* Расширяемость: Установка расширений для добавления новых функций и поддержки технологий.
* Интеграция с Git: Управление версиями (коммиты, слияния).
* Отладка: Установка точек останова и отслеживание выполнения JavaScript-кода.
* Предпросмотр изменений: Просмотр изменений в реальном времени в браузере.
* Консоль разработчика: Выполнение команд и тестирование кода через встроенную консоль.

1. **Как настроить расширение для работы с вашим проектом веб-приложения?**

* Установить необходимое расширение из магазина расширений VS Code.
* Настроить файл launch.json для определения параметров запуска и отладки.
* Указать URL веб-приложения и выбрать браузер, в котором будет происходить отладка в launch.json.

1. **Каковы основные шаги по отладке веб-приложения с использованием расширения?**

* Установить точки остановки в исходном коде, чтобы указать места, где выполнение должно приостанавливаться.
* Выполнить действия в браузере, чтобы инициировать выполнение кода. Когда программа достигнет точки остановки, можно детально рассмотреть внутренние взаимодействия между функциями и методами.
* Использовать панель отладки для управления выполнением кода и просмотра значений локальных и глобальных переменных на момент остановки, а также информации о текущем состоянии приложения.

1. **Как использовать расширение для взаимодействия с элементами DOM и отладки проблем с производительностью?**

* Убедиться, что установлено расширение для отладки (например, Debugger for Chrome).
* Настроить файл launch.json, чтобы указать URL приложения.
* Запустить отладку, установив точки останова в JavaScript-коде, чтобы следить за изменениями состояния приложения при взаимодействии с DOM.
* Использовать инструменты разработчика в браузере (обычно доступные через F12), чтобы исследовать элементы DOM. Можно изменять атрибуты, стили и содержимое элементов в реальном времени.
* Включить вкладку Performance в инструментах разработчика браузера, чтобы записывать и анализировать производительность приложения. Это поможет выявить узкие места и оптимизировать код.
* Обратить внимание на длительность выполнения скриптов, рендеринг и обработку событий.
* На основе анализа производительности внести изменения в код, чтобы улучшить скорость работы приложения.

1. **Какие дополнительные функции расширения полезны для повышения эффективности разработки веб-приложений?**

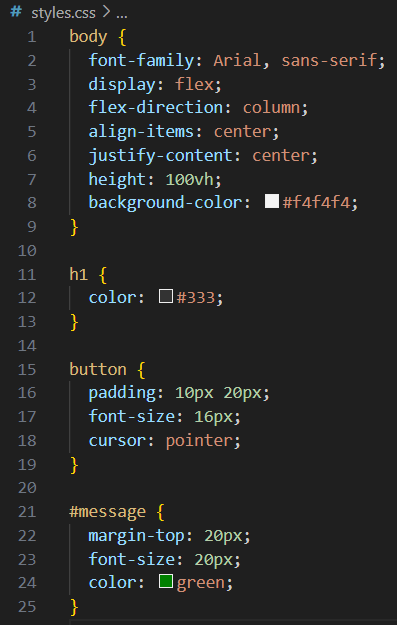
* Интеграция Git: Удобное управление версиями с помощью встроенной поддержки Git, позволяющей выполнять коммиты, слияния и создание веток.
* Live Server: Расширение для автоматического обновления страницы в браузере при сохранении изменений, что ускоряет процесс разработки.
* Bracket Pair Colorizer: Визуально различает пары скобок для улучшения читаемости кода.
* IntelliSense для CSS классов в HTML: Улучшает автодополнение классов CSS при редактировании HTML.

**Задача 1: Создайте простое веб-приложение с использованием HTML, CSS, JavaScript. Используйте расширение VS Code для браузера для запуска и отладки приложения в браузере. Ознакомьтесь с основными возможностями отладки, такими как установка точек остановки и проверка значений переменных**

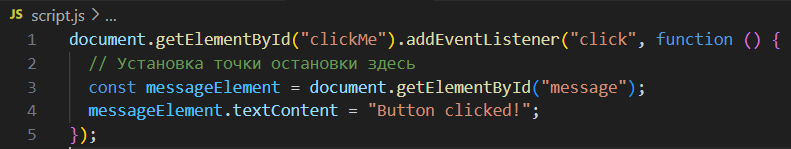
Index.html

****

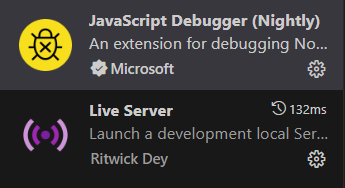
Styles.css



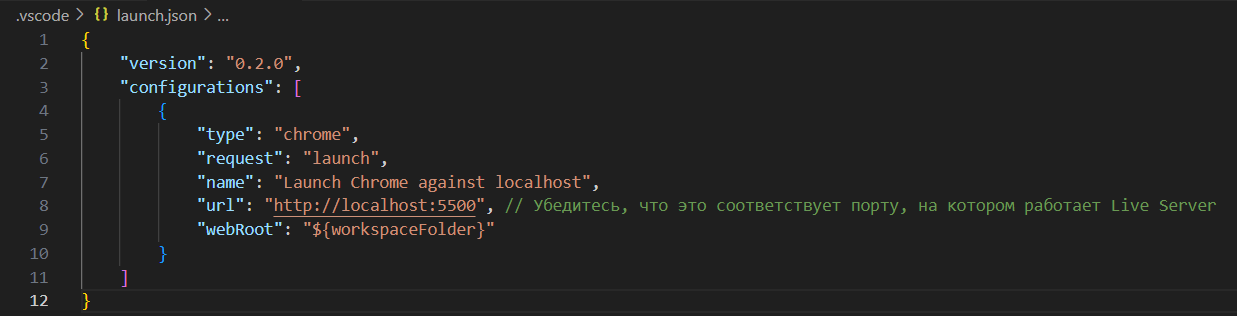
Script.js



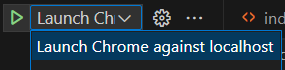
Нужно установить Live Server и Debugger

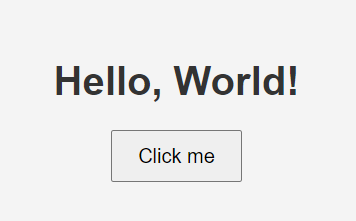


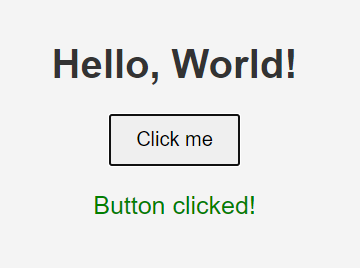
Launch.json



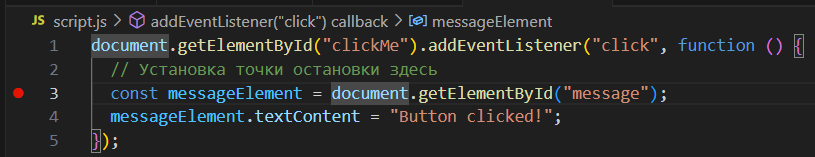




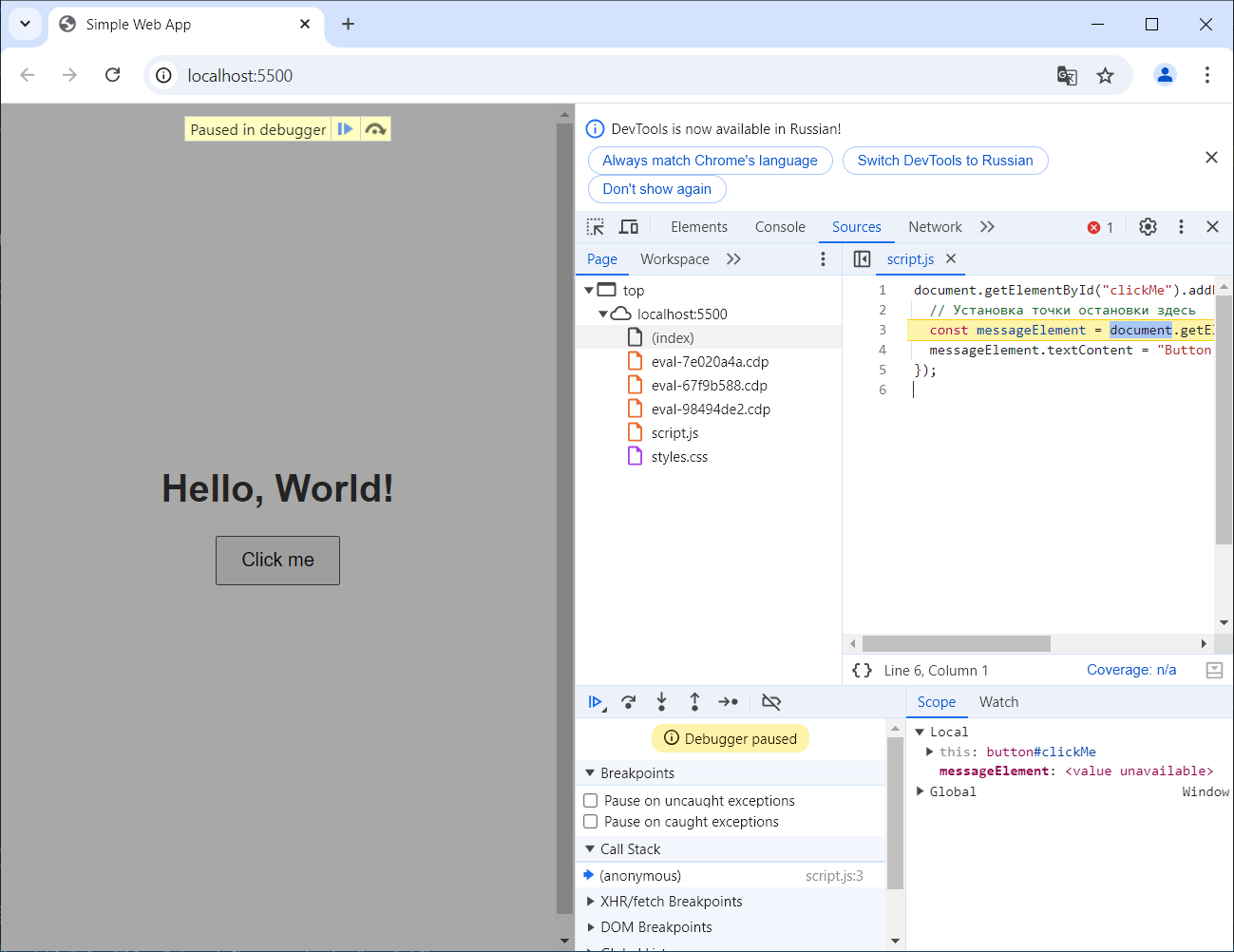




Ставим точку остановки, где она прописана в коде в js



Брейкпоинт сработал, внизу можно посмотреть переменные



**Задача 2: Создайте более сложное веб-приложение, использующее фреймворк Vue. Используйте расширение VS Code для браузера для отладки проблем с производительностью, таких как утечка памяти или узкие места в производительности. Опробуйте различные инструменты расширения для профилирования и анализа производительности приложения**

* Установка Vue:

npm install -g @vue/cli

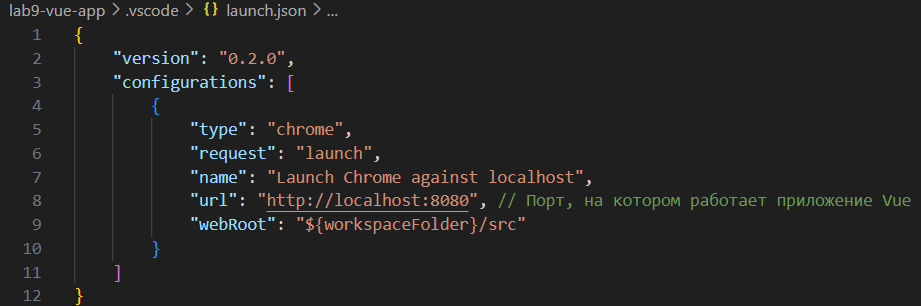
* Создание нового проекта Vue:

vue create lab9-vue-app

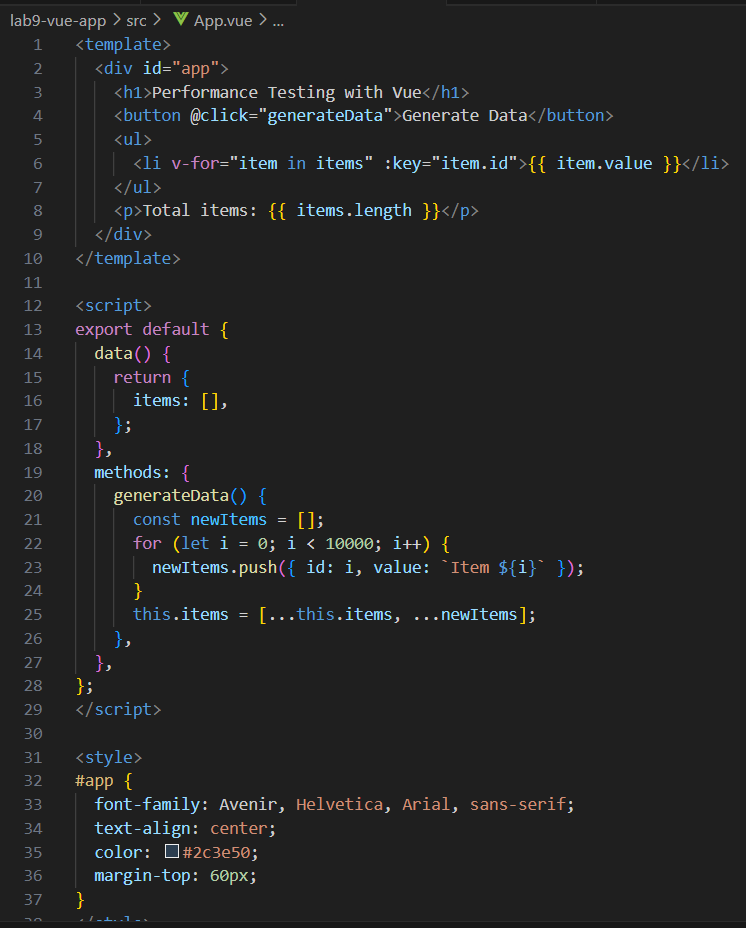
* Переход в папку проекта:

cd lab9-vue-app

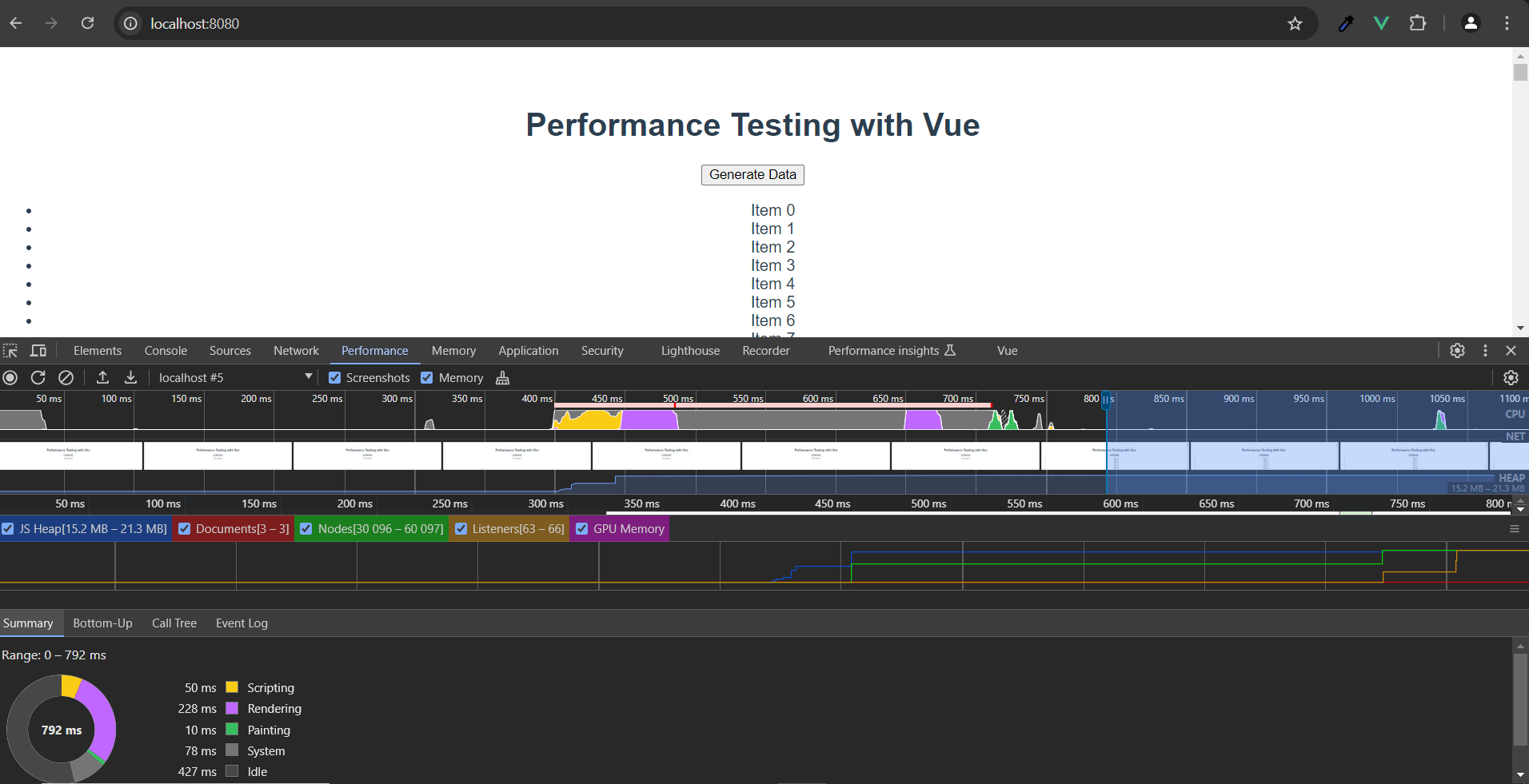
* launch.json



* src/App.vue



* Запустить приложение через npm run serve
* Открыть инструменты разработчика Chrome (нажмите F12).
* Перейти на вкладку Performance:
* Нажать на кнопку "Record" (кнопка с красной точкой) и выполнить действие (нажать кнопку "Generate Data").
* Остановить запись после того, как действие завершится.
* Обратите внимание на "Memory" и "CPU" usage. Если есть утечки памяти, вы можете увидеть, как память накапливается, когда вы создаете данные.



**Лабораторная 10. Создание и работа в профиле VS Code для AI приложений: NumPy, TensorFlow и Python.**

1. **В чём преимущества использования профиля VS Code для разработки приложений искусственного интеллекта?**

* Изоляция окружений – каждый профиль может иметь уникальные расширения и настройки, профили позволяют загружать только нужные плагины (например, TensorFlow, PyTorch, Jupyter), настраиваются под нужные задачи.
* Удобство переключения проектов – можно быстро переходить между ИИ, веб-разработкой или другими проектами.
* Интеграция с Jupyter Notebooks – VS Code имеет встроенную поддержку Jupyter Notebooks, что позволяет комбинировать код, изображения, текстовые описания и графики в едином документе. Это особенно полезно для исследования данных, прототипирования моделей и демонстрации результатов.

1. **Какие расширения VS Code необходимы для работы с NumPy, TensorFlow и Python?**

* Python – поддержка автодополнения, отладки, форматирования и запуска Python-кода (для разработки приложений искусственного интеллекта необходимо поставить пакеты NumPy, TensorFlow, это можно сделать с помощью пакетного менеджера pip).
* Jupyter – работа с ноутбуками Jupyter прямо в VS Code (удобно для экспериментов с NumPy и TensorFlow).
* Pylance – быстрое автодополнение и статический анализ кода для Python.
* TensorFlow Snippets – шаблоны кода для TensorFlow, упрощающие написание базовых операций

1. **Как настроить среду Python в профиле VS Code для разработки приложений искусственного интеллекта?**

* Установить необходимые расширения через магазин расширений VS Code.
* Создание виртуальной среды. В терминале VS Code выполнить:

python -m venv venv

* Затем активировать её:

venv\Scripts\activate

* Выбрать интерпретатор:

Нажать Ctrl + Shift + P, ввести Python: Select Interpreter и выбрать созданную виртуальную среду.

* Установка необходимых библиотек. В активированной виртуальной среде установить библиотеки:

pip install numpy tensorflow jupyter matplotlib

1. **Какие возможности отладки и профилирования предоставляет VS Code для приложений искусственного интеллекта?**

* Отладка Python-кода

**Точки остановки (Breakpoints)**: Позволяют приостановить выполнение программы в нужном месте для анализа состояния переменных.

**Шаги отладки**: Выполняйте код построчно (Step Into/Over) для отслеживания ошибок.

**Просмотр переменных**: Следите за значениями переменных в реальном времени в панели Variables.

* Отладка Jupyter-ноутбуков

Возможность устанавливать точки остановки прямо в ноутбуках и отлаживать отдельные ячейки.

* Профилирование производительности

Python Profiling: С помощью интеграции с библиотеками, такими как cProfile, можно измерять время выполнения функций и строить отчёты. Отчёты можно проанализировать с помощью визуализаторов, таких как SnakeViz.

* Интеграция с TensorBoard

Запуск TensorBoard прямо в VS Code для отслеживания метрик обучения моделей TensorFlow и PyTorch.

1. **Как интегрировать Git в профиль VS Code для управления версиями кода приложений искусственного интеллекта?**

Чтобы интегрировать Git в профиль VS Code для управления версиями ИИ-приложений, сначала установите Git и убедитесь, что он добавлен в PATH. В VS Code встроена поддержка Git, поэтому при открытии проекта автоматически будут обнаружены репозитории. Если репозиторий ещё не инициализирован, можно запустить команду git init через встроенный терминал.

После этого в панели Source Control станет доступно отслеживание изменений, коммитов и создание веток. Можно привязать профиль к удалённому репозиторию, используя команду git remote add origin <URL> и выполнять команды git push и pull прямо из интерфейса. Для удобной аутентификации с GitHub или GitLab можно использовать расширение GitHub Pull Requests and Issues.

Настройки профиля можно дополнить файлом .gitignore для исключения временных файлов (например, логов TensorBoard или весов моделей).

**Задача 1: Создайте новый профиль VS Code для разработки приложений искусственного интеллекта. Установите необходимые расширения и настройте среду Python для работы с NumPy и TensorFlow**

Из расширений поставить:

* Python – поддержка Python.
* Jupyter– работа с ноутбуками .ipynb.
* Pylance – интеллектуальное автодополнение.
* TensorFlow Snippets – готовые фрагменты кода для TensorFlow.

• Создать виртуальную среду. В терминале VS Code выполнить:

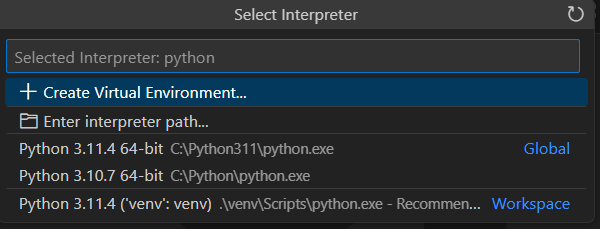


• Затем активировать её:



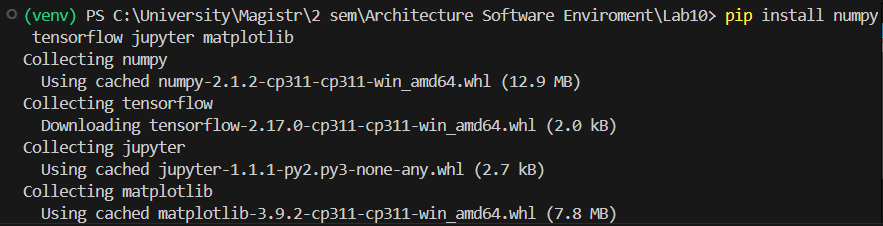
• Выбрать интерпретатор:

Нажать Ctrl + Shift + P, ввести Python: Select Interpreter и выбрать созданную виртуальную среду.



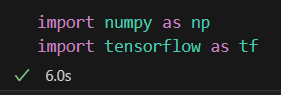
• Установка необходимых библиотек. В активированной виртуальной среде установить библиотеки:

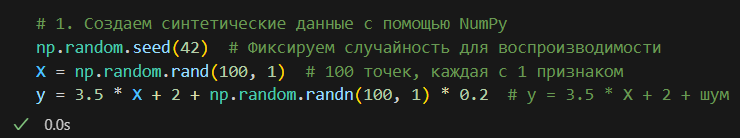
pip install numpy tensorflow jupyter matplotlib

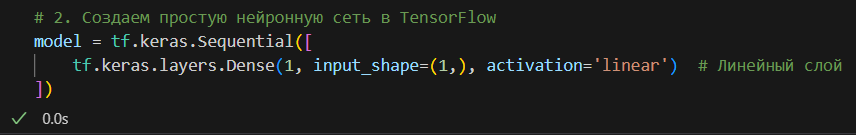


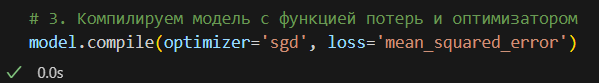
**Задача 2: Создайте простой скрипт Python, который использует NumPy и TensorFlow для обучения модели машинного обучения. Профилируйте скрипт в VS Code**

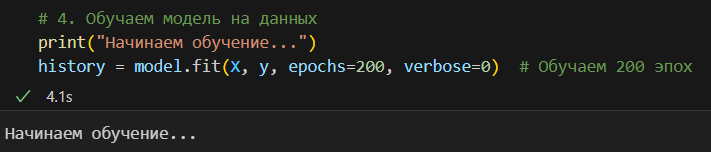
Линейная регрессия - пример простого скрипта на Python, который использует NumPy и TensorFlow для создания и обучения модели линейной регрессии. Этот скрипт демонстрирует, как создать модель, обучить её на синтетических данных и выполнить предсказание.

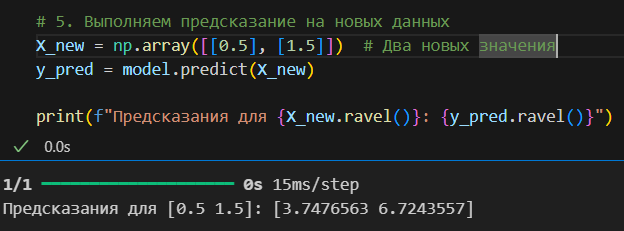


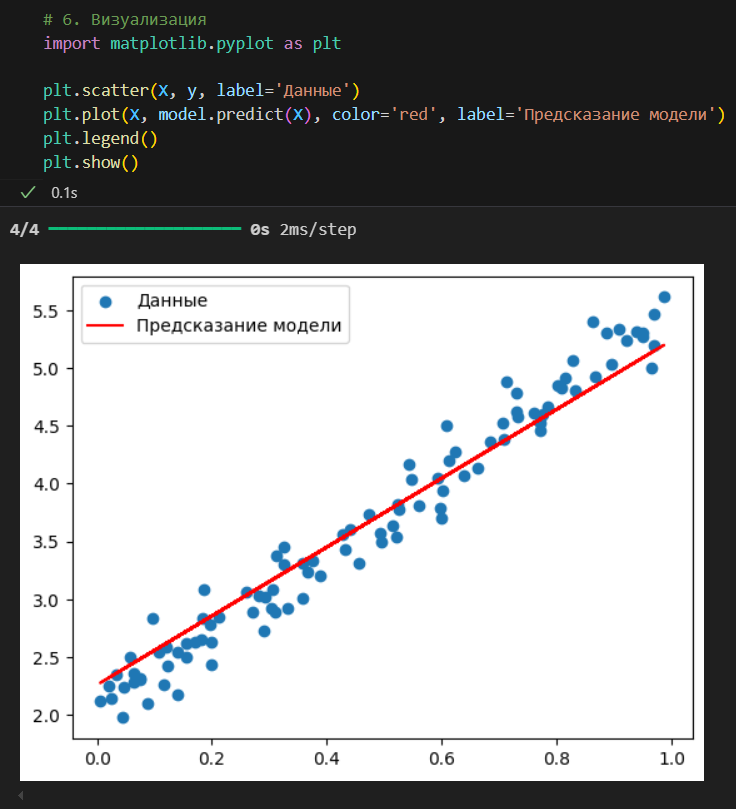












Профилирование Jupyter-ноутбука в VS Code (Профилирование позволяет оценить время, затрачиваемое на выполнение отдельных операций в программе)

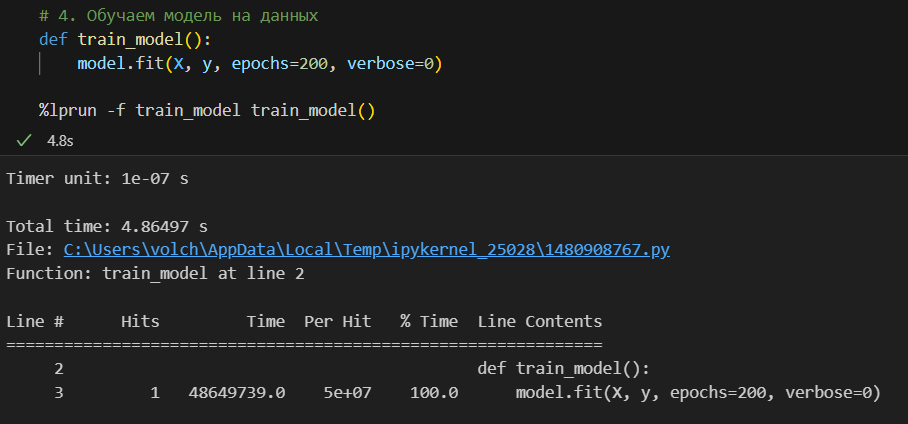
* Установка библиотеки профилирования: Если она не установлена, запустите в новой ячейке:

!pip install line\_profiler

* Добавьте магическую команду профилирования: В первой ячейке ноутбука подключите line\_profiler:

%load\_ext line\_profiler

* Оберните интересующую функцию профилирования: Например, чтобы профилировать обучение, измените код так:



* Анализ результатов профилирования:

Результаты профилирования покажут, сколько времени заняло выполнение каждой строки в функции. Вы можете использовать эти данные для оптимизации.

**Лабораторная 11. Использование Docker для разработки и развертывания приложений.**

1. **Какие основные преимущества использования Docker в процессе разработки и развёртывания приложений?**

Изоляция среды – Каждое приложение работает в контейнере с собственными зависимостями и конфигурацией. Это предотвращает конфликты между версиями библиотек.

Портативность – Контейнеры с приложениями могут работать одинаково на разных платформах (локально, на сервере или в облаке).

Быстрое развертывание и масштабирование – Запуск контейнеров занимает секунды.

Консистентность между окружениями – "Работает у меня — работает везде": одинаковая среда разработки, тестирования и продакшена.

Эффективное использование ресурсов – Контейнеры потребляют меньше ресурсов, чем виртуальные машины.

Микросервисная архитектура – Упрощает создание и управление микросервисами, так как каждый сервис можно упаковать в отдельный контейнер.

Docker повышает эффективность работы разработчиков, упрощая управление приложениями на всех этапах их жизненного цикла.

1. **Опишите процесс создания Docker-образа приложения.**

* Создание Dockerfile – Создайте файл Dockerfile с инструкциями для сборки образа (например, базовый образ, копирование файлов, установка зависимостей, команды для запуска).
* Подготовка приложения – Убедитесь, что все необходимые файлы (код, зависимости, конфигурации) находятся в нужных директориях.
* Сборка образа – Выполните команду:



Это создаст образ на основе инструкций в Dockerfile.

* Запуск контейнера – Запустите контейнер на основе созданного образа:



* Тестирование – Проверьте, что приложение работает корректно (например, откройте в браузере).

1. **Как использовать Docker Compose для управления несколькими контейнерами?**

* Создание docker-compose.yml – Описать все необходимые сервисы (контейнеры), их образы, порты, зависимости и сети:



* Запуск контейнеров – Выполнить команду для запуска всех сервисов:



Параметр -d запускает контейнеры в фоновом режиме.

* Проверка статуса – Проверить запущенные контейнеры:



* Остановка и удаление контейнеров – Чтобы остановить и удалить все запущенные контейнеры:



* Перезапуск при изменениях – Внести изменения в код или конфигурации и выполнить:



Это пересоберёт образы и перезапустит контейнеры.

1. **Какие стратегии развёртывания приложений в Docker вы знаете?**
2. **Recreate (Полное пересоздание)**  
   – Контейнеры останавливаются и удаляются, затем создаются новые с обновлённым образом.  
   *Простая стратегия, но может вызывать простой приложения.*



1. **Rolling Update (Постепенное обновление)**  
   – Контейнеры обновляются один за другим, сохраняя работоспособность приложения. Используется в Docker Swarm и Kubernetes.  
   *Снижает риск и позволяет избежать простоя.*
2. **Blue-Green Deployment**  
   – Два окружения: активное (**Blue**) и новое (**Green**). После успешного тестирования трафик переключается на новое окружение.  
   *Гарантирует отсутствие простоев и возможность быстрого отката.*
3. **Canary Deployment**  
   – Новая версия разворачивается для небольшой группы пользователей. При успешном результате трафик постепенно увеличивается.  
   *Уменьшает риски при выпуске крупных изменений.*
4. **A/B-тестирование**  
   – Разворачиваются две версии (A и B) для тестирования разных вариантов приложения на сегментах пользователей.  
   *Используется для экспериментов и оптимизации.*
5. **Shadow Deployment (Теневое развёртывание)**  
   – Новая версия получает копию трафика основного сервиса без взаимодействия с пользователями.  
   *Применяется для тестирования производительности и мониторинга.*

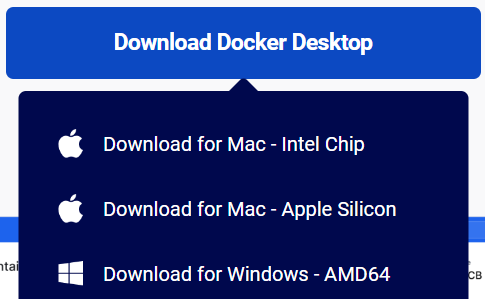
Эти стратегии позволяют выбирать подходящий подход в зависимости от требований к стабильности, скорости развёртывания и возможности отката изменений.

1. **Как обеспечить безопасность и управление образами Docker?**

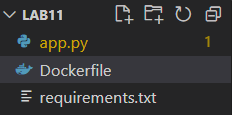
* Использование официальных и проверенных образов – Загружайть образы только из доверенных источников, таких как Docker Hub или внутренний реестр.
* Регулярное сканирование образов – Проводить проверку уязвимостей в образах с помощью инструментов (например, Trivy, Clair, Snyk).
* Подпись и валидация образов – Использовать Docker Content Trust (DCT) для подписи и проверки подлинности образов.
* Ограничение прав контейнеров – Запускать контейнеры с минимальными правами.
* Сегментация сетей – Изолировать контейнеры в разных сетях.
* Мониторинг и логирование – Настроить логирование событий для обнаружения аномалий.
* Ограничение ресурсов – Установить лимиты на использование CPU и памяти

**Задача 1: Создайте Docker-образ для простого веб-приложения, например приложения Django. Определите соответствующий файл Dockerfile и протестируйте образ, запустив его локально**

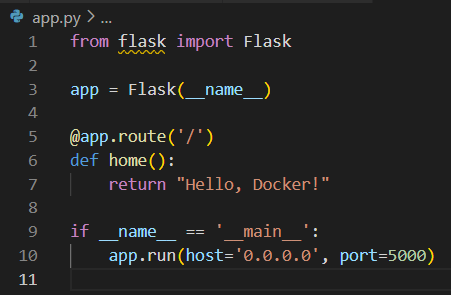
Для создания Docker-образа простого веб-приложения на Python (Flask - это легковесный веб-фреймворк для языка Python, который предоставляет минимальный набор инструментов для создания веб-приложений) для начала нужно установить Docker на официальном сайте



Создать проект, который будет состоять из таких файлов:



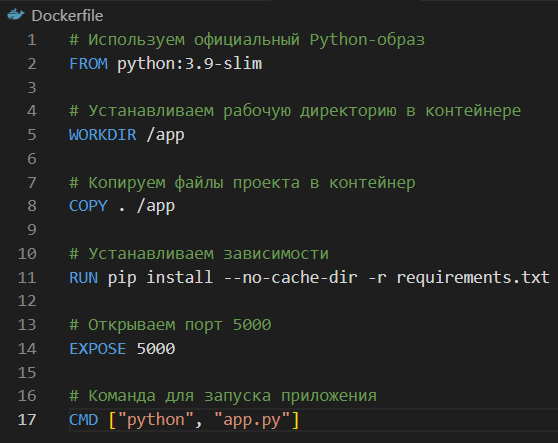
В app.py код, выводящий Hello, Docker:



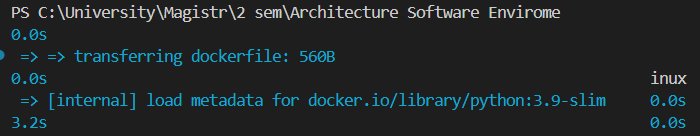
В requirements.txt:



Dockerfile описывает инструкции для сборки Docker-образа:



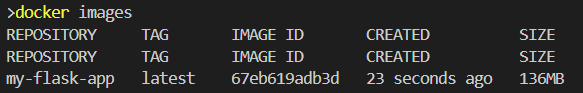
Далее открыть терминал в директории Lab11 и выполнить команду, чтобы собрать Docker-образ:



…



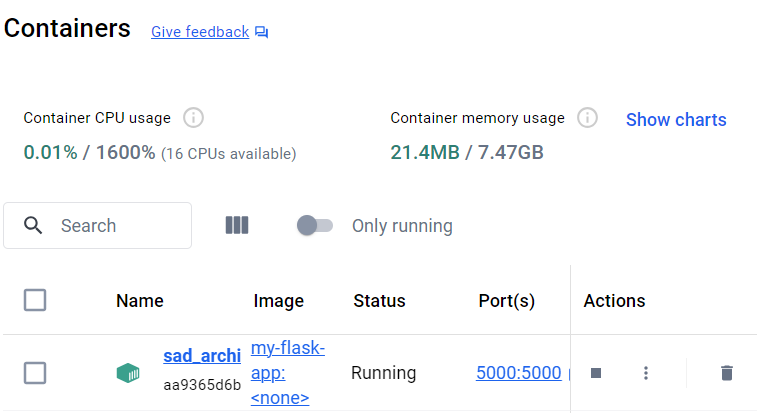
Проверить, что образ создан:



my-flask-app в списке образов

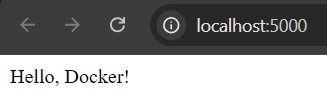
Запустить контейнер на основе созданного образа:



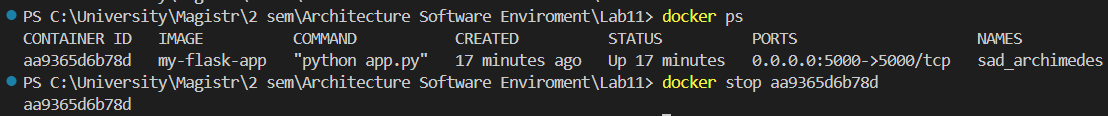


Открыть браузер и перейти по адресу:

<http://localhost:5000>

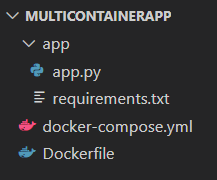


Чтобы остановить контейнер, нужно узнать его ID:



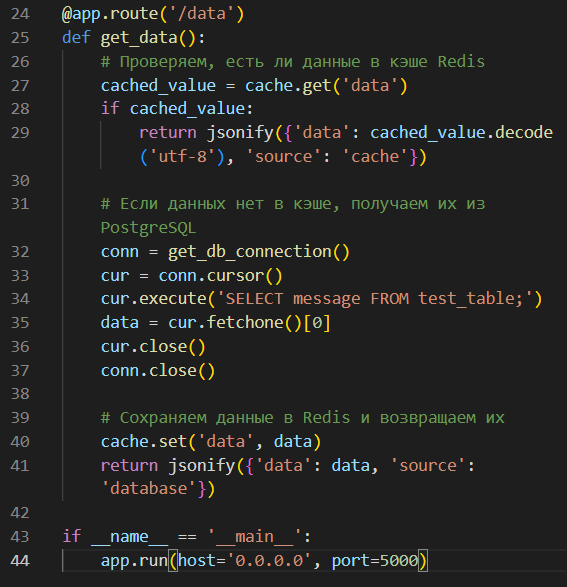
**Задача 2: Используйте Docker Compose для создания и управления стеком Docker, состоящим из нескольких контейнеров, которые взаимодействуют друг с другом. Например, вы можете создать стек, включающий контейнеры веб-приложений, базы данных и кэша. Разверните стек на локальной машине.**

Создать проект, который будет состоять из таких файлов:

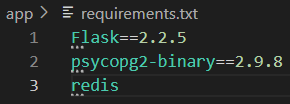


В app.py код будет обращаться к базе данных PostgreSQL и кэшировать данные с помощью Redis:

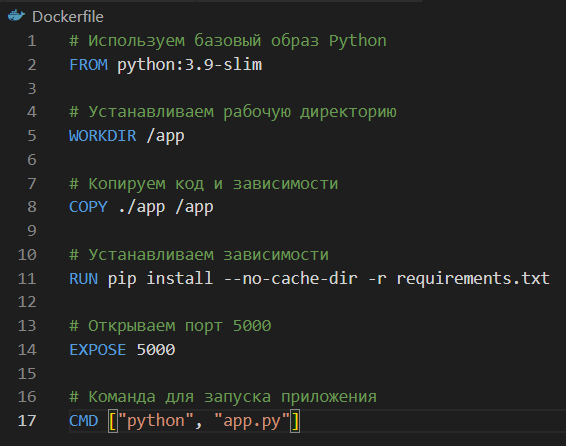




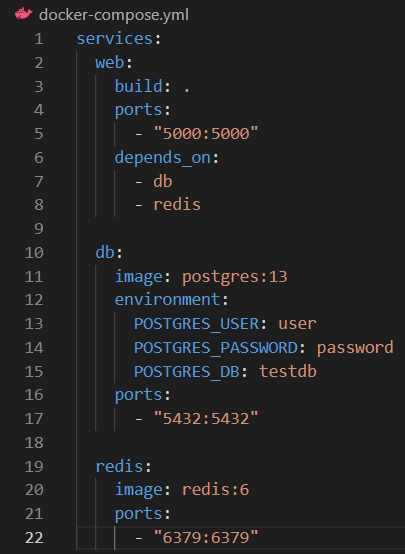
В requirements.txt зависимости – необходимые библиотеки:



Dockerfile для Flask-приложения:

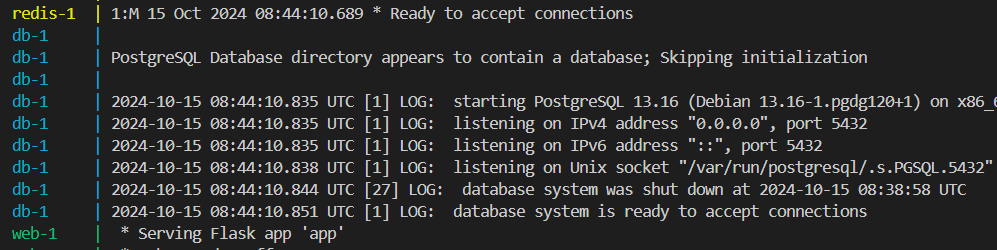


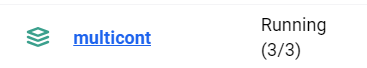
Docker Compose файл (docker-compose.yml) определяет три сервиса: Flask, PostgreSQL и Redis:



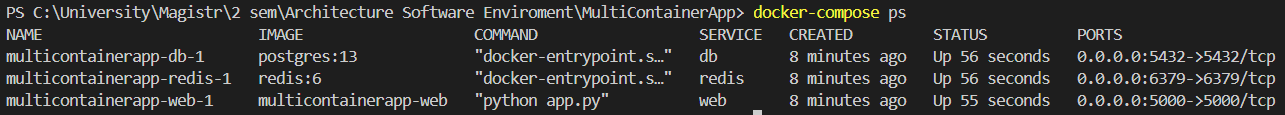
Сборка и запуск стека:



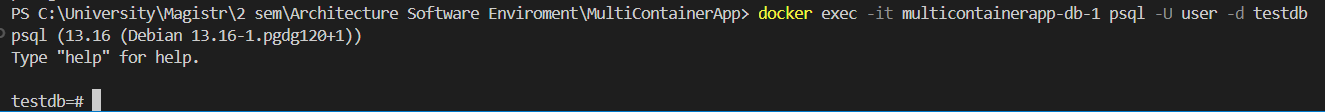




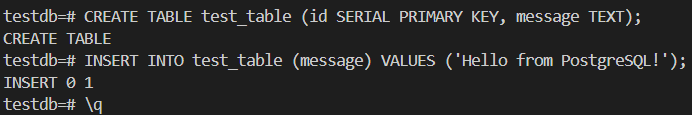
Проверка состояния контейнеров в новом терминале:



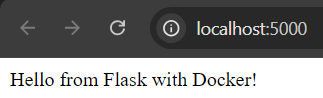
Команда, чтобы войти в контейнер PostgreSQL:



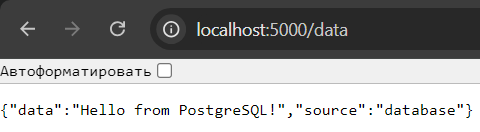
Затем выполнить SQL-команды (Создание таблицы и добавление данных в PostgreSQL):



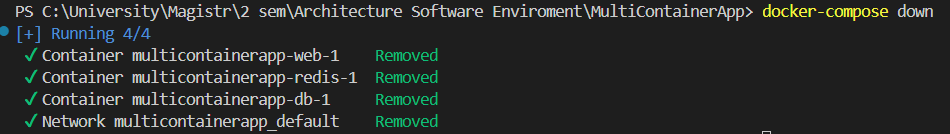
Проверка главной страницы:



Проверка данных из PostgreSQL и Redis:



Чтобы остановить и удалить все контейнеры:



**Лабораторная 12. Использование ChatGPT для программирования прикладных задач.**

1. **Какие преимущества и недостатки использования ChatGPT для программирования прикладных задач?**

Преимущества использования ChatGPT для программирования

1. Быстрая генерация кода: Можно быстро получить примеры, шаблоны и функции.
2. Решение ошибок и багов: Помогает находить и исправлять ошибки в коде.
3. Обучение и документация: Объясняет концепции и технологии на доступном языке.
4. Экономия времени: Автоматизирует рутинные задачи.
5. Поддержка разных языков и фреймворков: Может работать с Python, JavaScript, SQL, Vue.js, и другими инструментами.

Недостатки использования ChatGPT для программирования

1. Ошибки и неточности: Код может содержать баги или не учитывать нюансы.
2. Отсутствие понимания контекста: Генерируемые решения могут не соответствовать специфическим требованиям проекта.
3. Ограниченные знания: Модель ограничена датой обучения и не знает самых новых технологий или изменений.
4. Неоптимальный код: Предлагаемые решения могут быть менее производительными или не следовать лучшим практикам.
5. Зависимость от интернета: Для работы с ChatGPT необходимо иметь стабильное интернет-соединение, что может быть проблемой в некоторых местах.
6. **Как ChatGPT может помочь в автоматизации процессов разработки программного обеспечения?**

* Генерацию кода для стандартных шаблонов и функций.
* Генерация README, комментариев и технической документации.
* Автоматическое создание тестов
* Предложение улучшений и более чистого кода.
* Помощь в отладке.

1. **Какие типы задач особенно подходят для решения с помощью ChatGPT?**

ChatGPT особенно эффективен в задачах, связанных с генерацией шаблонного кода, автоматизацией рутинных процессов, написанием документации, созданием тестов и отладкой. Он также хорошо справляется с объяснением сложных концепций, помощью в поиске ошибок и предложением вариантов оптимизации. Такие задачи требуют быстрой обработки информации и применения стандартных решений, где креативность и глубинное понимание предметной области не всегда критичны.

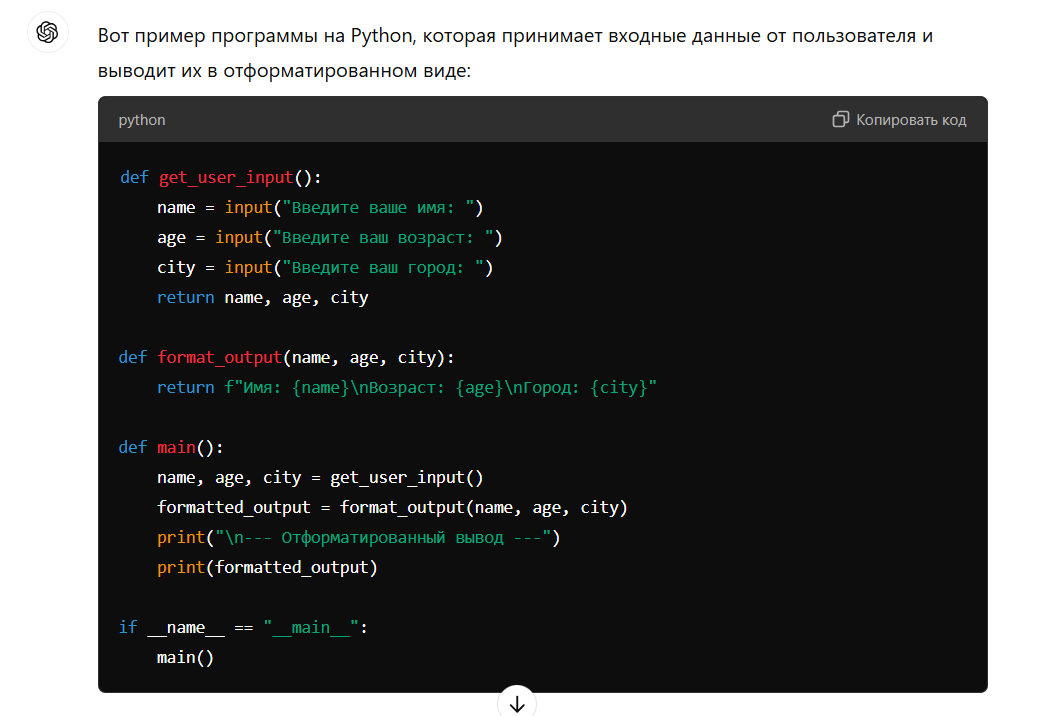
1. **Как обеспечить, чтобы код, сгенерированный ChatGPT, был надёжным и безопасным?**

Чтобы обеспечить надёжность и безопасность кода, сгенерированного ChatGPT, необходимо тщательно проверять его на наличие ошибок. Важно дополнительно тестировать сгенерированный код с помощью тестов, а также обеспечить его соответствие требованиям безопасности, таким как защита от SQL-инъекций. Также полезно провести код-ревью с участием опытных разработчиков, чтобы выявить потенциальные проблемы и улучшить качество кода.

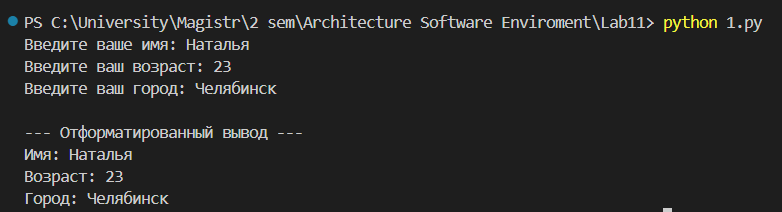
1. **Каковы этические соображения при использовании ChatGPT для программирования?**

Этические соображения при использовании ChatGPT для программирования включают вопросы интеллектуальной собственности, так как код может содержать заимствованные фрагменты из других источников. Существует также риск создания уязвимого или ненадежного кода, что может повлиять на безопасность пользователей. Важно учитывать возможность снижения навыков разработчиков при чрезмерном использовании. Наконец, необходимо учитывать, как использование таких инструментов влияет на трудовой рынок и рабочие места в сфере программирования.

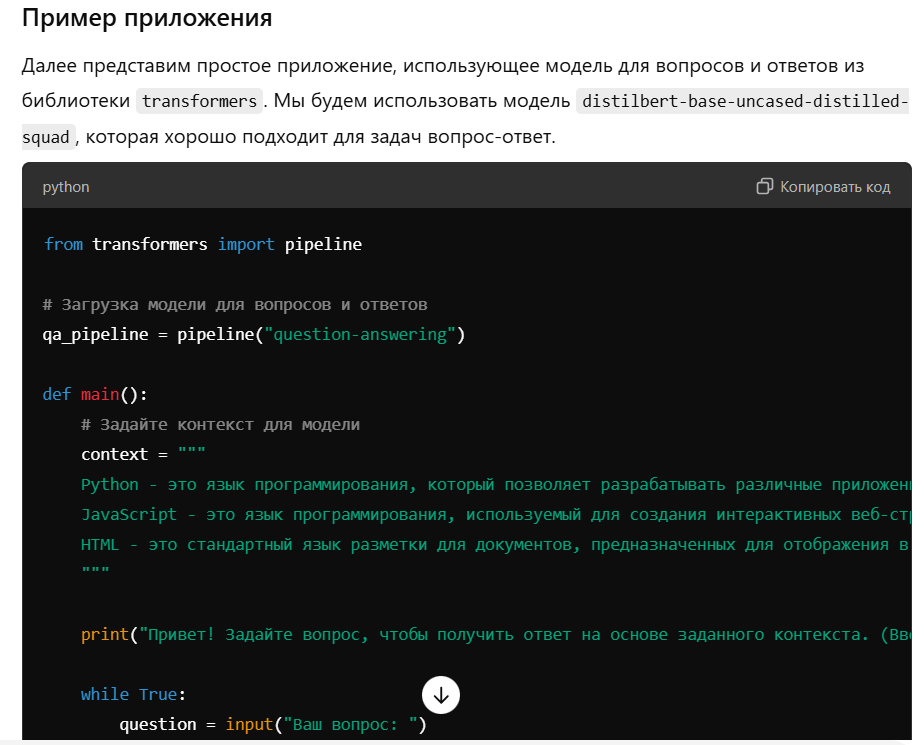
**Задача 1: Используя ChatGPT, разработайте программу на Python, которая будет принимать входные данные от пользователя и будет выводить их в отформатированном виде**

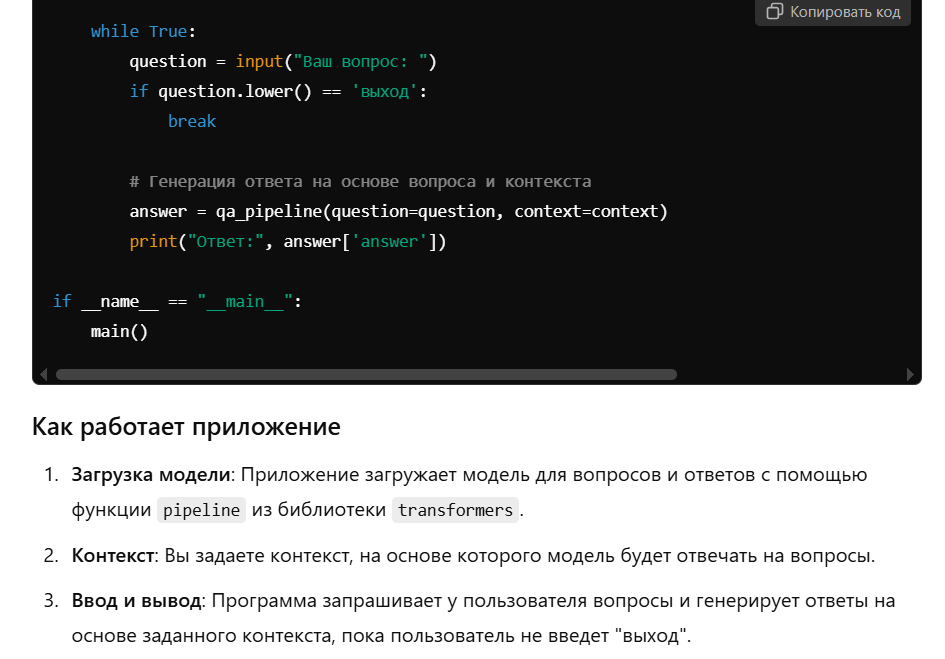


Проверим код:



**Задача 2: Создайте приложение на Python, которое использует ChatGPT для генерации ответов на вопросы пользователей на основе заданного набора данных**





Проверим код:

