Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

специальность 09.02.07 «Информационные системы и программирование»

Квалификация: Программист

ПРАКТИЧЕСКИЕ РАБОТЫ

ПО МДК 04.02 «Обеспечение качества функционирования компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент  группы П50-3-22  Сливина Диана Герардовна | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_ К.С. Образцова  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 года |

Москва 2025

[ПРАКТИЧЕСКАЯ РАБОТА №1 3](#_Toc208232721)

ПРАКТИЧЕСКАЯ РАБОТА №1

Цель работы: 1. Выполнить сборки и развертывание контейнеров из примера, и описать своими словами то, что именно вы сделали и как.

2. Описать вкладки в Docker (включая вкладки внутри собранного образа и контейнера)

3. Написать свои Doсkerfile и собрать контейнеры с программами из архива.

Ход работы.

1. Программа Python

Для этой программы необходимо

Для данного проекта необходимо использовать образ openjdk:alpine. Далее прописать название версии (необязательно) с помощью LABEL, после чего создать рабочую директорию внутри контейнера с помощью WORKDIR, далее скопировать программу из текущей директории хоста в корень с помощью COPY, а после запустить скрипт с помощью CMD.

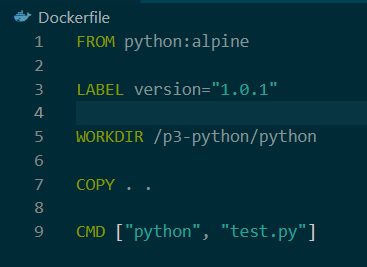


Рисунок 1 - Dockerfile программы на питоне

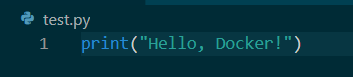


Рисунок 2 - Текст программы на питоне

Чтобы правильно собрать программу необходимо указать команду docker build, затем дополнительные параметры (например --no-cache собриает программу без кэша), после чего указать тэг для образа с помощью ключа -t и указать контекст сборки (в данном случае – текущая директория)

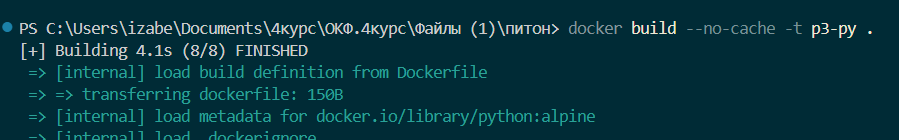


Рисунок 3 - Сборка программы на питоне

Чтобы запустить программу нужно сс помощью команды docker run создать контейнер, его можно назвать с помощью ключа –name, далее нужно указать имя образа.

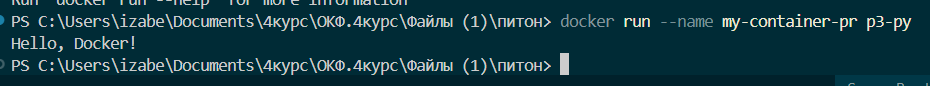


Рисунок 4 - Запуск программы на питоне

1. Сайт (одностраничный)

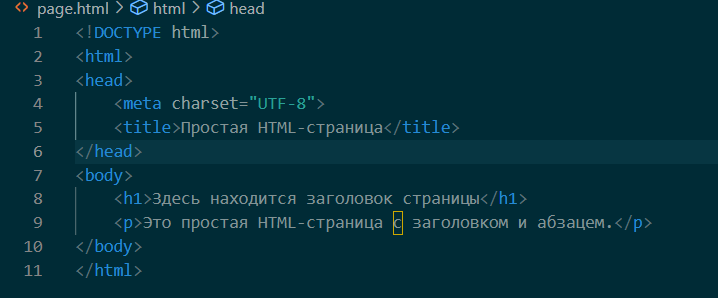


Рисунок 5 - Код сайта

В Dockerfile сайта создаётся образ nginx, после чего из хоста скопировать программу по пути /usr/share/nginx/html для корректного запуска и при необходимости записать номер порта через EXPOSE (но это необязательно и на действительный порт не влияет).

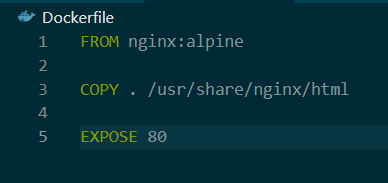


Рисунок 6 - Dockerfile сайта

При сборке сайта происходит всё то же самое, что и с программой на питоне.

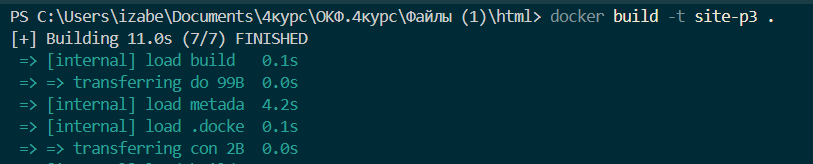


Рисунок 7 - Сборка сайта

Тем не менее, при запуске необходимо указать порт с помощью ключа -p, с помощью ключа -d запустить фоновый режим и вновь указать имя контейнера и имя образа. После чего через Docker можно перейти непосредственно на сайт.

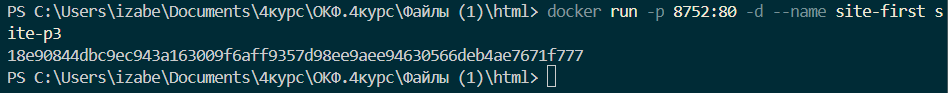


Рисунок 8 - Запуск сайта

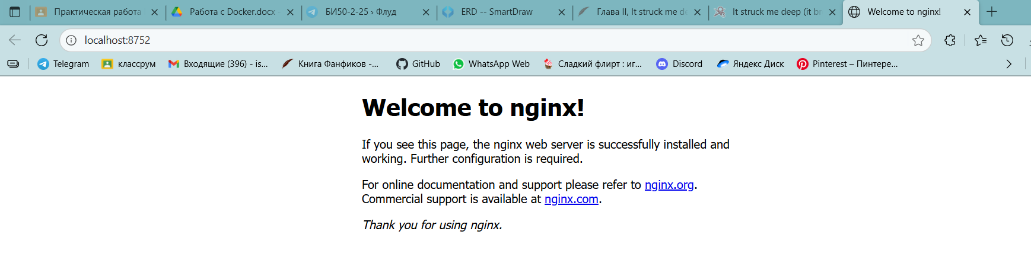


Рисунок 9 - Дефолтная страница nginx

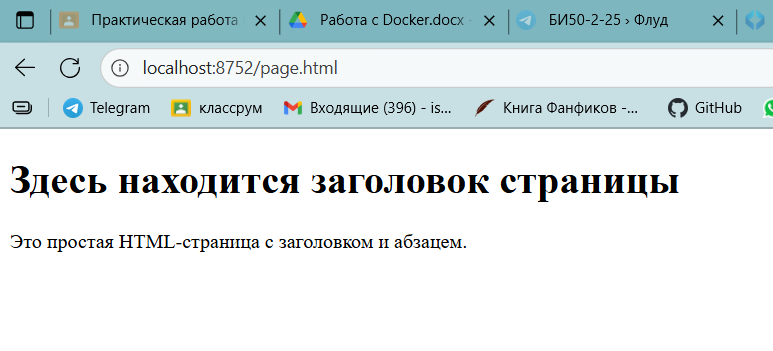


Рисунок 10 - Основная страница сайта

1. Калькулятор Java



Рисунок 11 - Код калькулятора на джаве

В Dockerfile для консольной программы на джава нужно помимо шагов, выполненных в файле для питон-программы, прописать RUN чтобы скомпилировать программу. Также использован образ openjdk.



Рисунок 12 - Dockerfile калькулятора на джаве

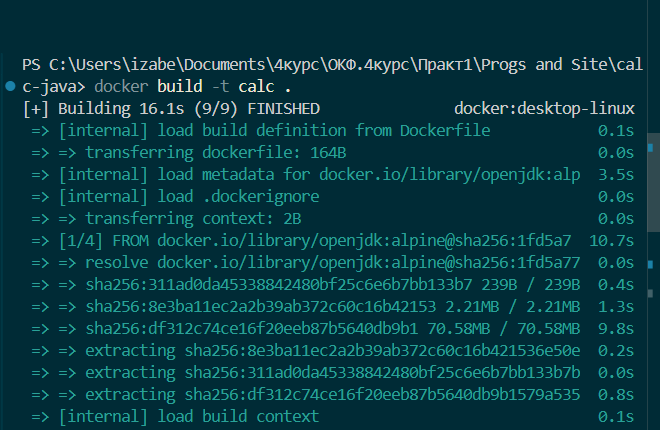


Рисунок 13 - Сборка калькулятора на джаве

Для того чтобы взаимодействовать с консолью необходимо прописать ключ -it (интерактивный терминал).

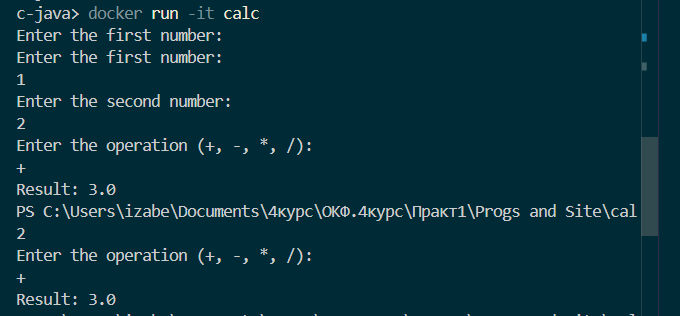


Рисунок 14 - Запуск калькулятора на джаве

1. Калькулятор Python

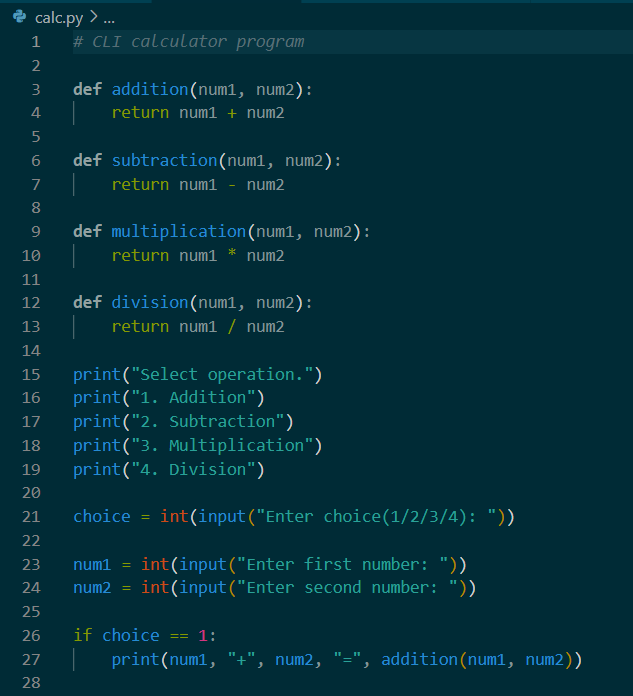


Рисунок 15 - Код калькулятора на питоне

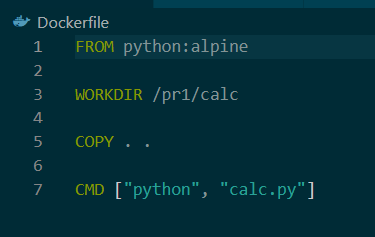


Рисунок 16 - Dockerfile калькулятора на питоне

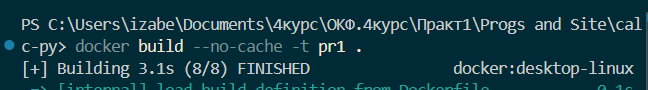


Рисунок 17 - Сборка калькулятора на питоне



Рисунок 18 - Запуск калькулятора на питоне

1. Калькулятор Dart

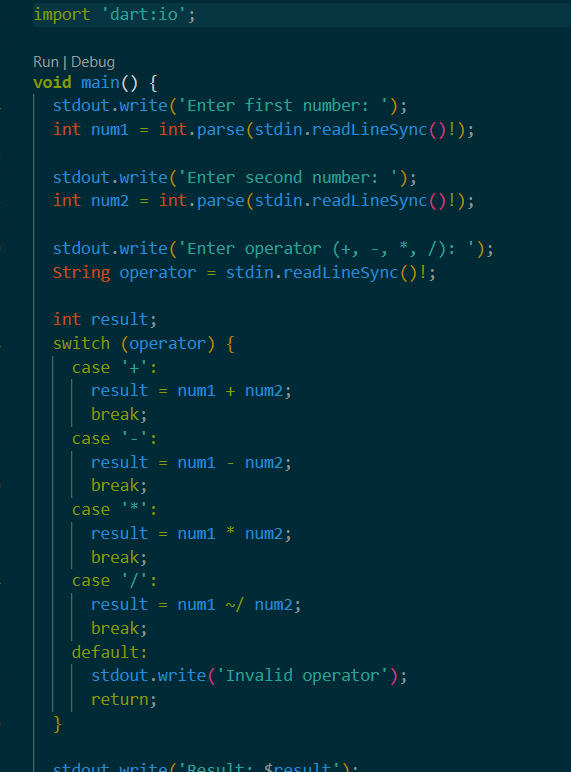


Рисунок 19 - Код калькулятора на дарте

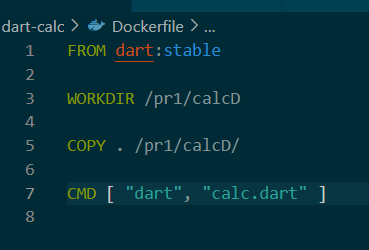


Рисунок 20 - Dockerfile калькулятора на дарте

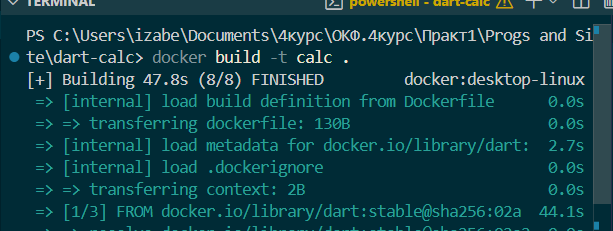


Рисунок 21 - Сборка калькулятора на дарте

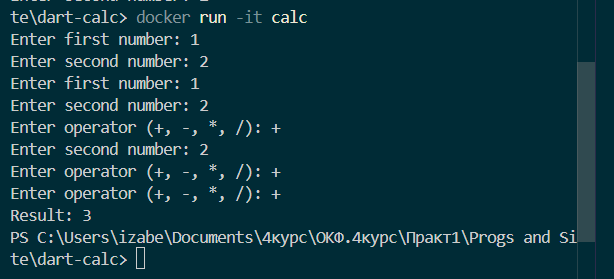


Рисунок 22 - Запуск калькулятора на дарте

1. Игра Java



Рисунок 23 - Код игры на джаве



Рисунок 24 - Dockerfile игры на джаве

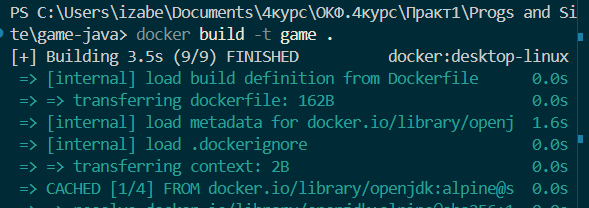


Рисунок 25 - Сборка игры на джаве

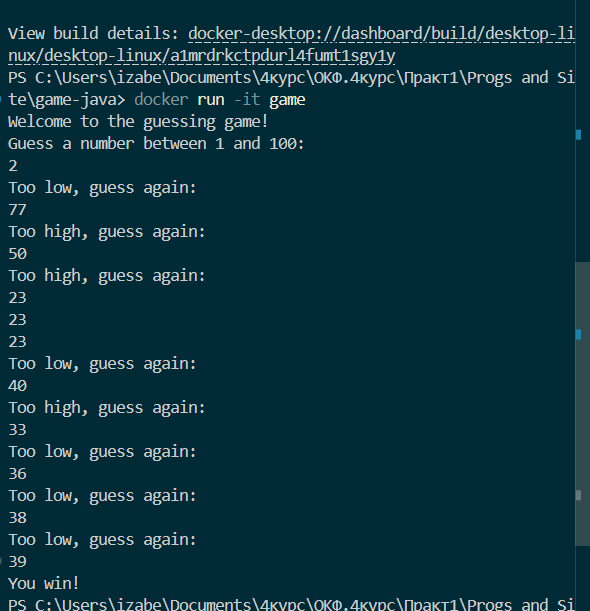


Рисунок 26 – Запуск игры на джаве

1. Камень-ножницы-бумага Python



Рисунок 27 - Код игры на питоне

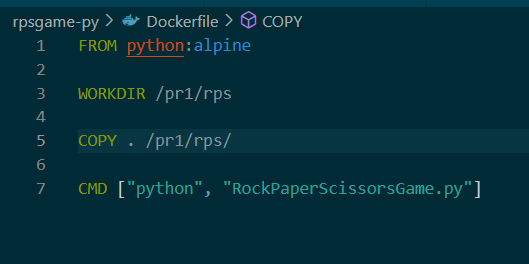


Рисунок 28 - Dockerfile игры на питоне

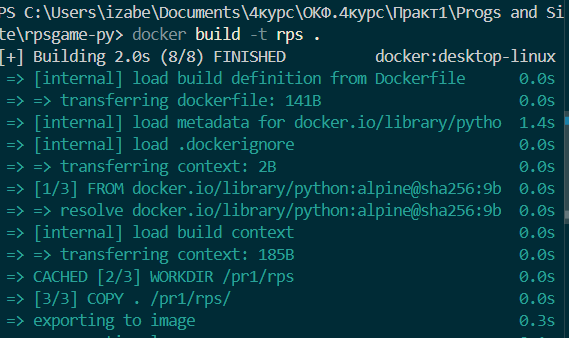


Рисунок 29 – Сборка игры на питоне

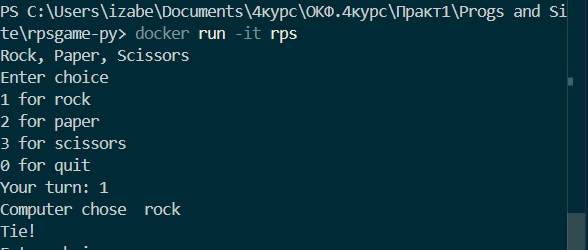


Рисунок 30 – Запуск игры на питоне

1. Сайт (многостраничный)

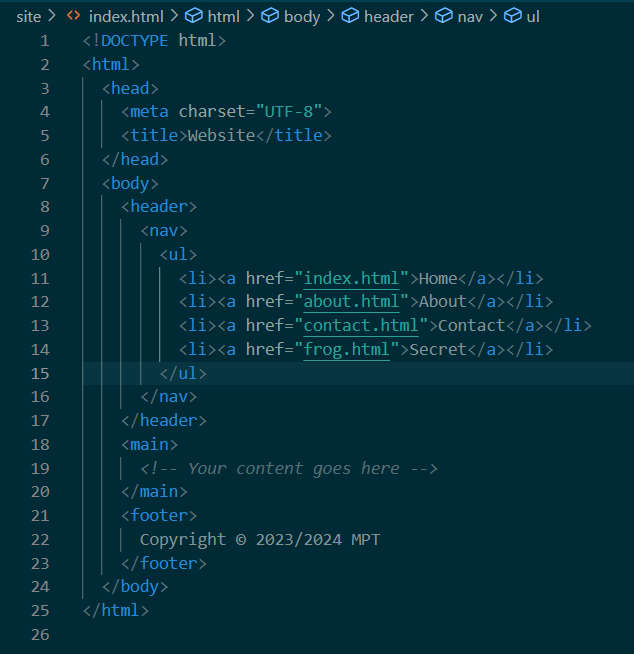


Рисунок 31 - Код многостраничного сайта.

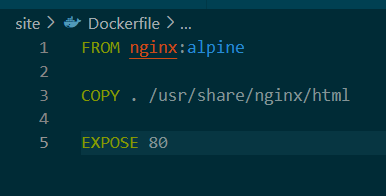


Рисунок 32 - Dockerfile многостраничного сайта.

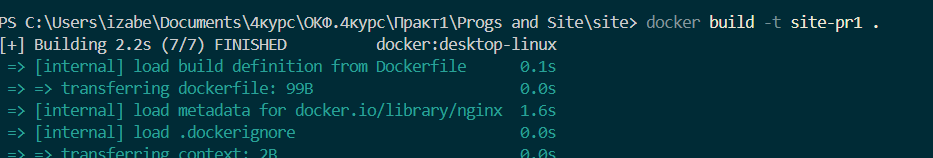


Рисунок 33 - Сборка многостраничного сайта

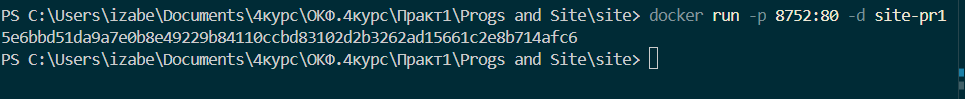


Рисунок 34 - Запуск многостраничного сайта

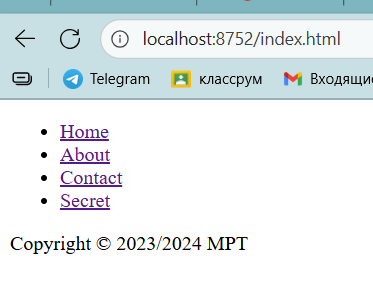


Рисунок 35 - Главная многостраничного сайта

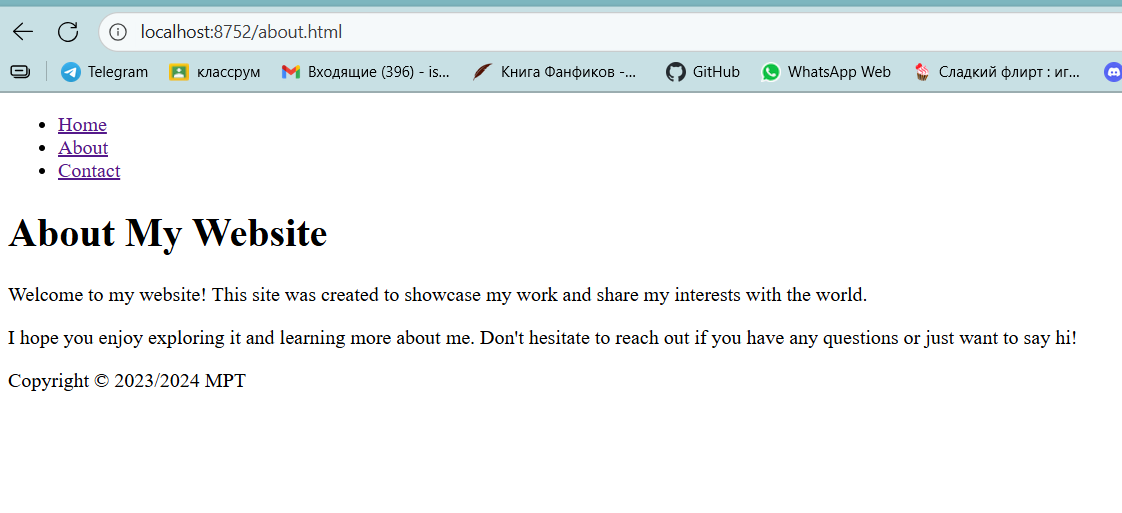


Рисунок 36 - Описание многостраничного сайта

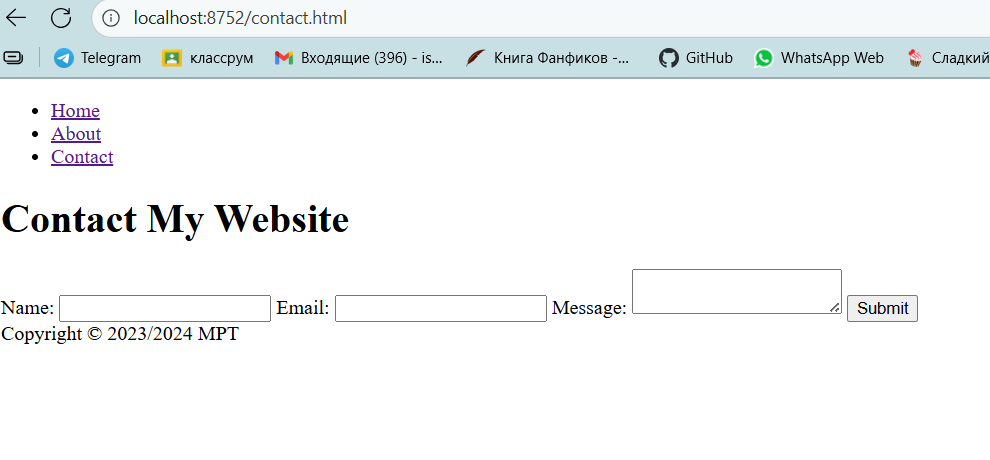


Рисунок 37 - Связь многостраничного сайта



Рисунок 38 - Секрет многостраничного сайта

1. Описание вкладок Docker

На странице контейнеров указана история контейнеров, их имя (id), образ к которому они принадлежат, порт (если есть), время последнего запуска, и состояние контейнера. Также на этой странице указан объём занимаем памяти и ресурсов устройства. Перейдя в конкретный контейнер можно просмотреть его файлы (во время запуска), логи, инспектирование.

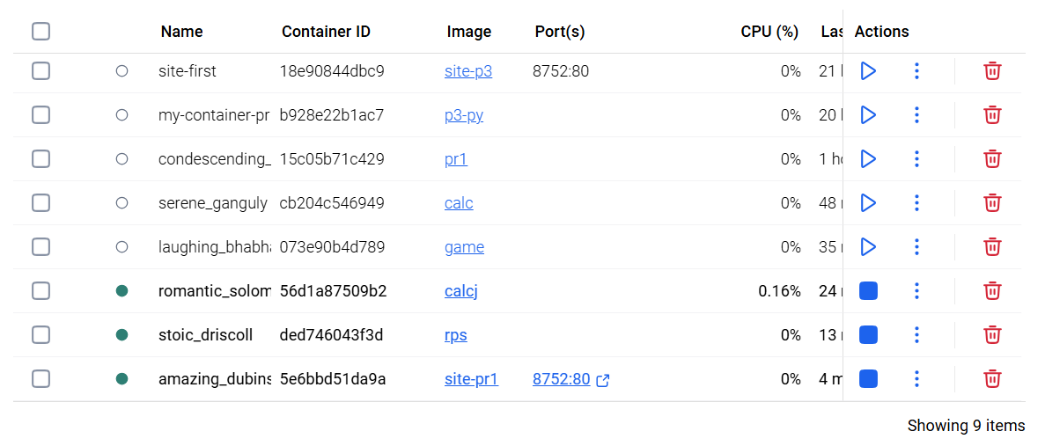


Рисунок 39 - Список контейнеров

На странице образов указаны все созданные образы, их номера, дата создания, статус и размер. Также на этой странице указан занятый объём памяти.



Рисунок 40 - Список образов

Также есть вкладка истории сборок.

Вывод: выполнена сборка и развертывание контейнеров из примера, описаны вкладки в Docker, написаны собственные Dockerfile к программам из архива и выполнена их сборка.

ПРАКТИЧЕСКАЯ РАБОТА №2

Цель работы: развернуть приложение time-db из примера в Docker. Описать каждый пункт -почему именно эти команды и компоненты вы создаете. Развернуть любое свое приложение в Docker которое имеет как минимум 3 компонента (например API+База данных+Adminer).

Ход работы.

Сперва нужно определить из каких компонентов состоит программа и к каждому отдельному компоненту написать свой докер файл.

Ниже представлены докер файл для API (Рис. 41) и докер файл для фронта (Рис. 42).

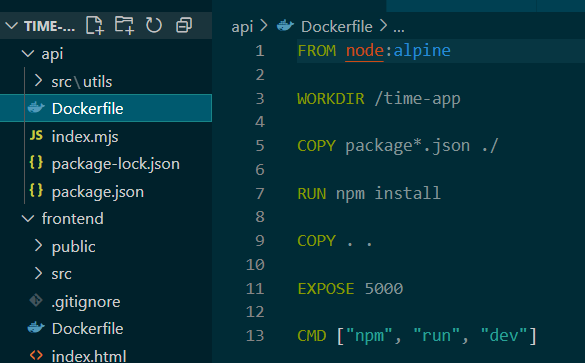


Рисунок 41 - Dockerfile API

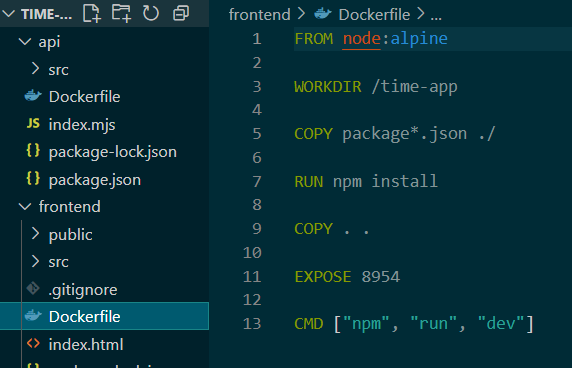


Рисунок 42 - Dockerfile frontend

Далее нужно создать docker-compose для того, чтобы объединить все имеющиеся сервисы

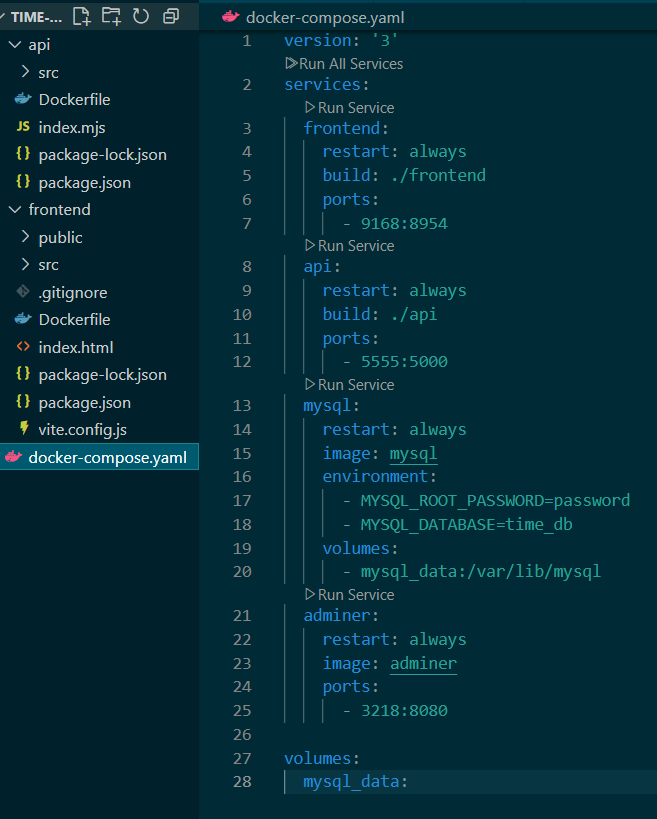


Рисунок 43 – Docker-compose

После этого нужно в терминале ввести команду docker-compose up и в докер-десктоп можно открыть все сервисы. Фронт (Рис. 44), апи (Рис. 45) и бд (Рис. 46).

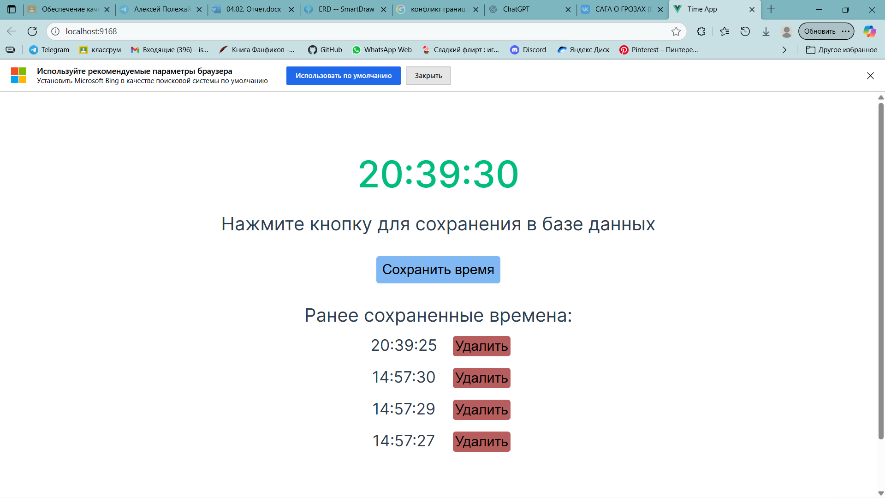


Рисунок 44 - Фронт

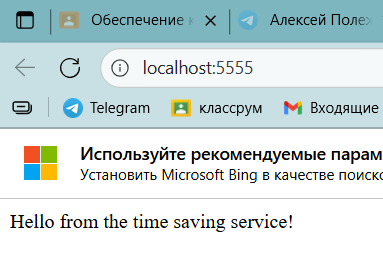


Рисунок 45 - API

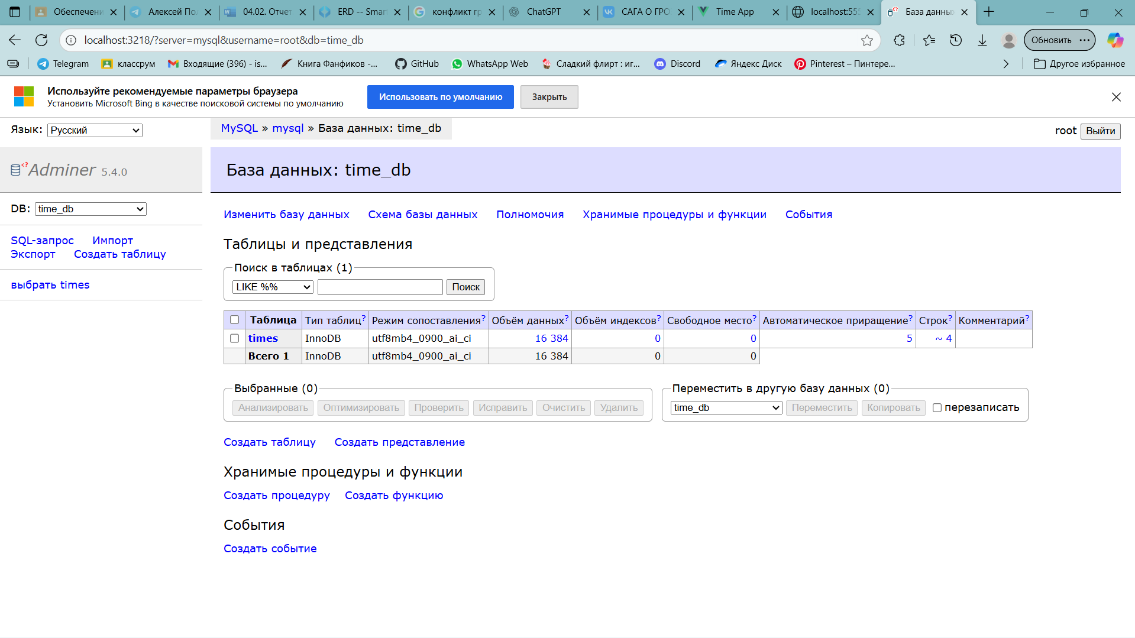


Рисунок 46 - База данных

Вывод: развернуто приложение time-db из примера в Docker. Описаны шаги выполнения. В результате практической работы изучены принципы работы docker-compose.

ПРАКТИЧЕСКАЯ РАБОТА №3

Цель работы: развернуть любое приложение, которое будет обновлять какую-либо информацию (не фото) через Bind Mounts и через Volume. Описать работу приложения, как работают данные компоненты, а также разницу между ними.

Ход работы.

Сперва необходимо написать страницу html с данными для изменений. В данном случае будет изменяться тест на странице. После чего написать докерфайл и создать образ (Рис. 48-49).

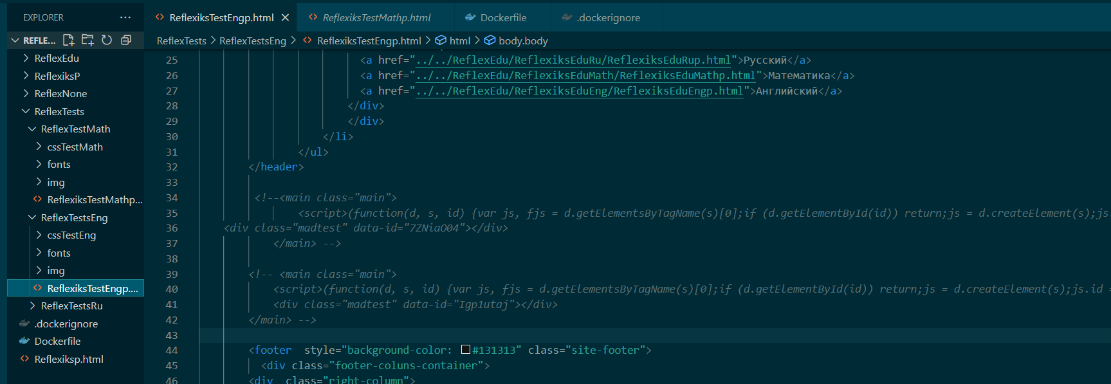


Рисунок 47 - Рабочий код

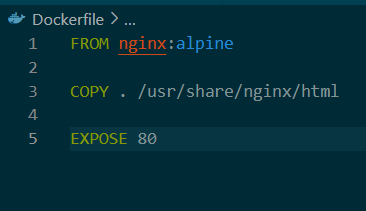


Рисунок 48 - Докер файл

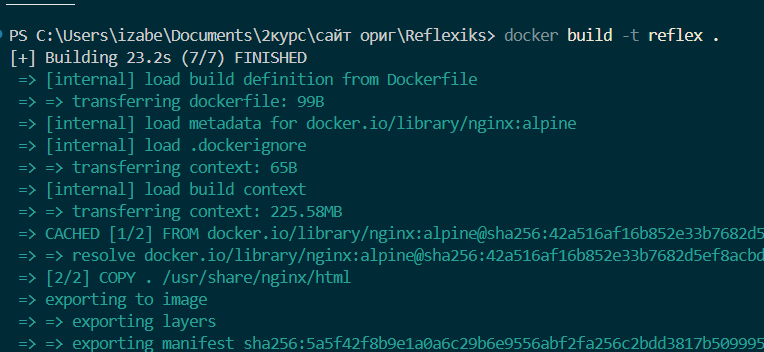


Рисунок 49 - Создание образа

Теперь можно проверить работоспособность образа.



Рисунок 50 - Запуск тестового контейнера

Далее нужно создать mount с помощью ключа -v. Теперь при изменении данных в папке ReflexTests они будут меняться и в контейнере (Рис. 53-56).

Mount – это способ примонтировать (подключить внешний ресурс хранения (например, диск, папку или volume) к файловой системе контейнера или системы так, чтобы его содержимое стало доступно по определённому пути) хранилище в контейнер с более гибкой настройкой источника, точки монтирования и дополнительных опций.

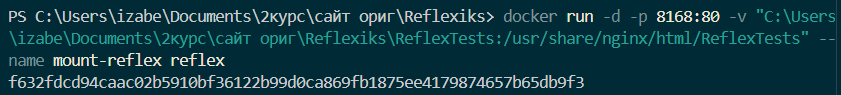


Рисунок 51 - Создание mount

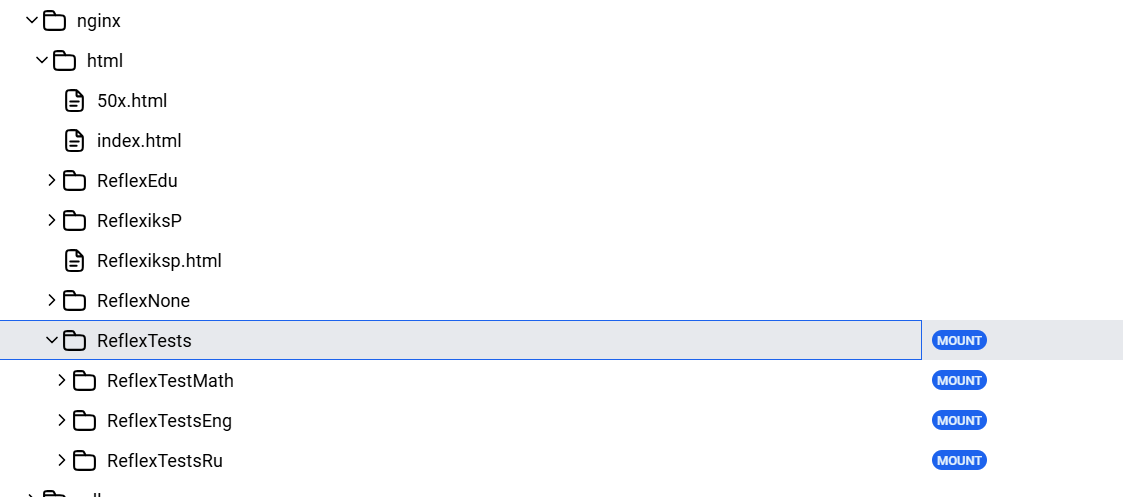


Рисунок 52 - Результат создания mount

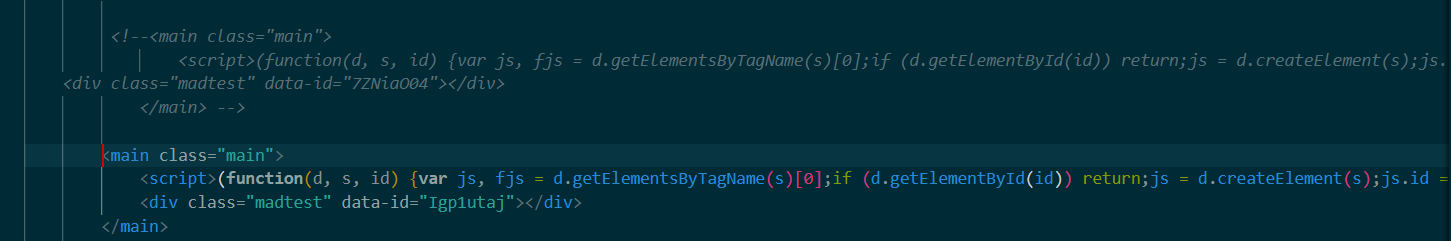


Рисунок 53 - Изначальный вариант кода

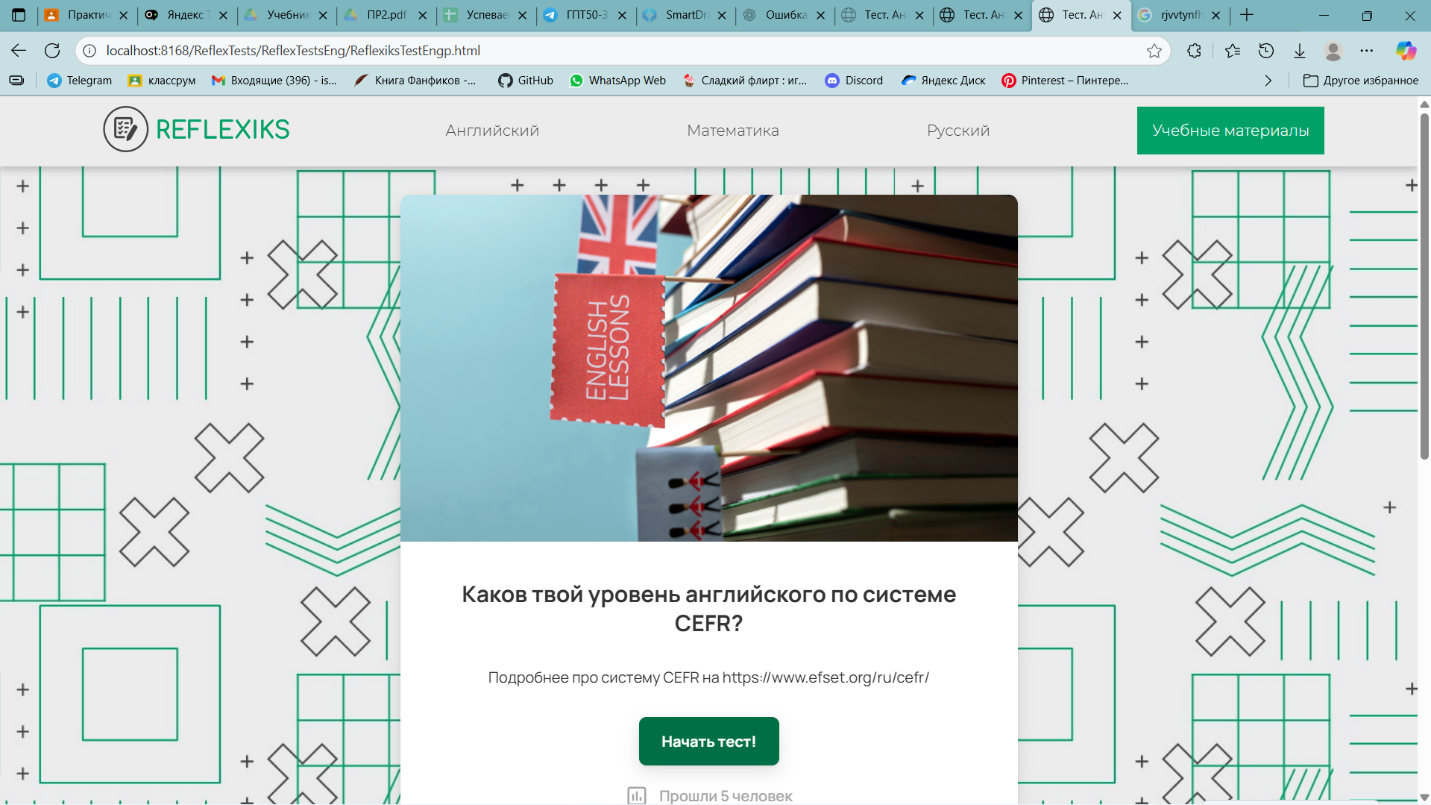


Рисунок 54 - Изначальный вариант страницы

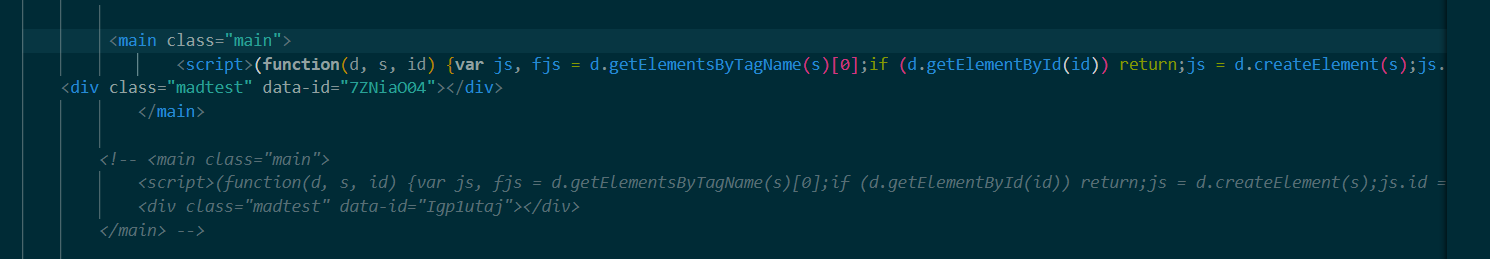


Рисунок 55 - Обновление кода

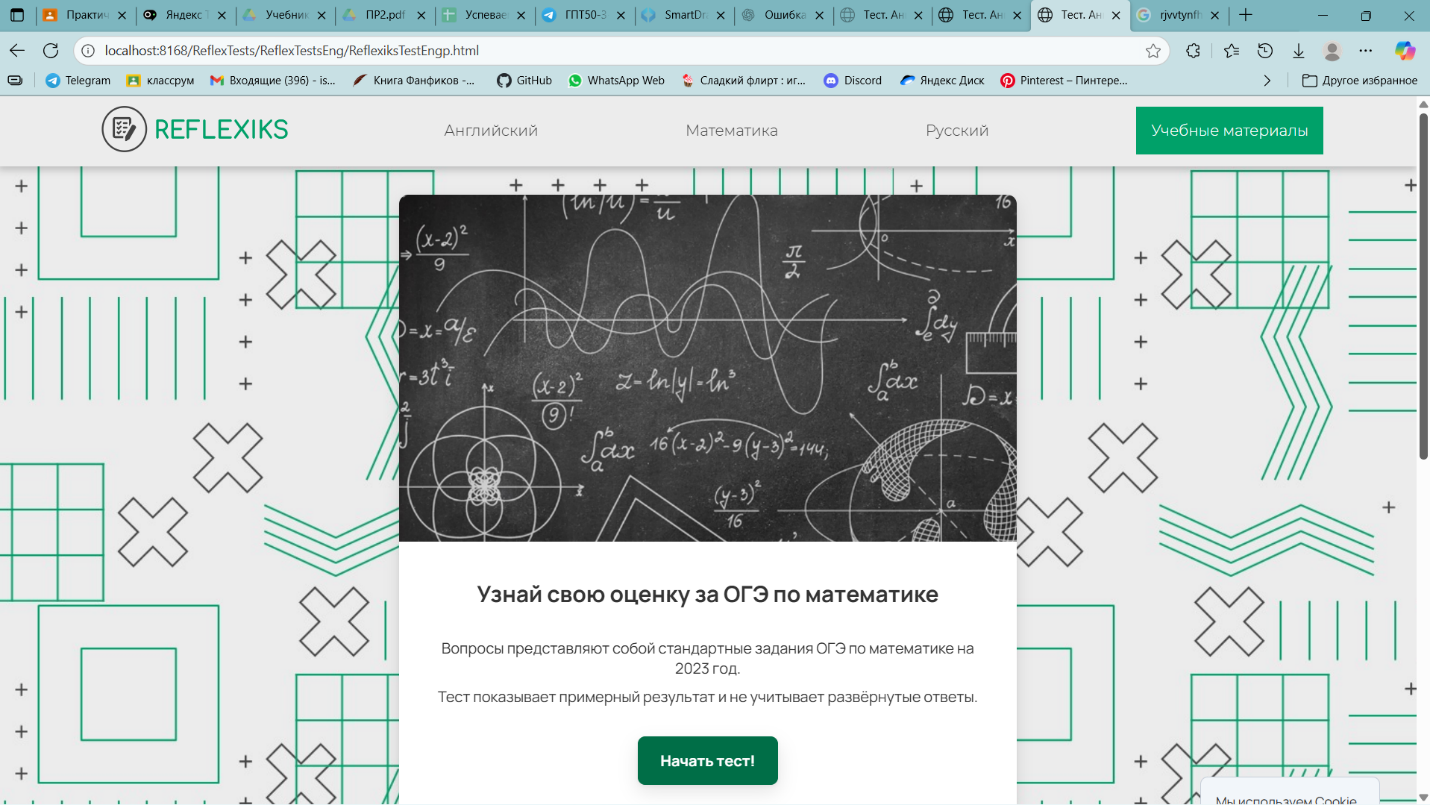


Рисунок 56 - Изменение страницы

Далее нужно создать volume с помощью соответствующей команды и запустить новый контейнер.

Volume – это специализированное хранилище данных, управляемое Docker, которое существует независимо от контейнера и позволяет сохранять данные между перезапусками и удалением контейнеров.

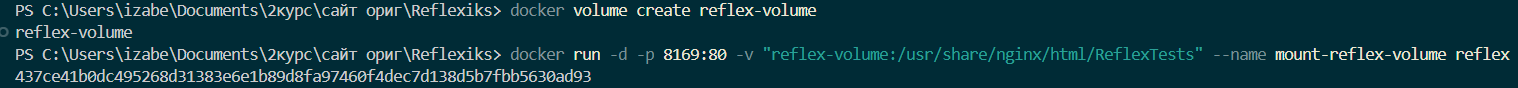


Рисунок 57 - Создание volume

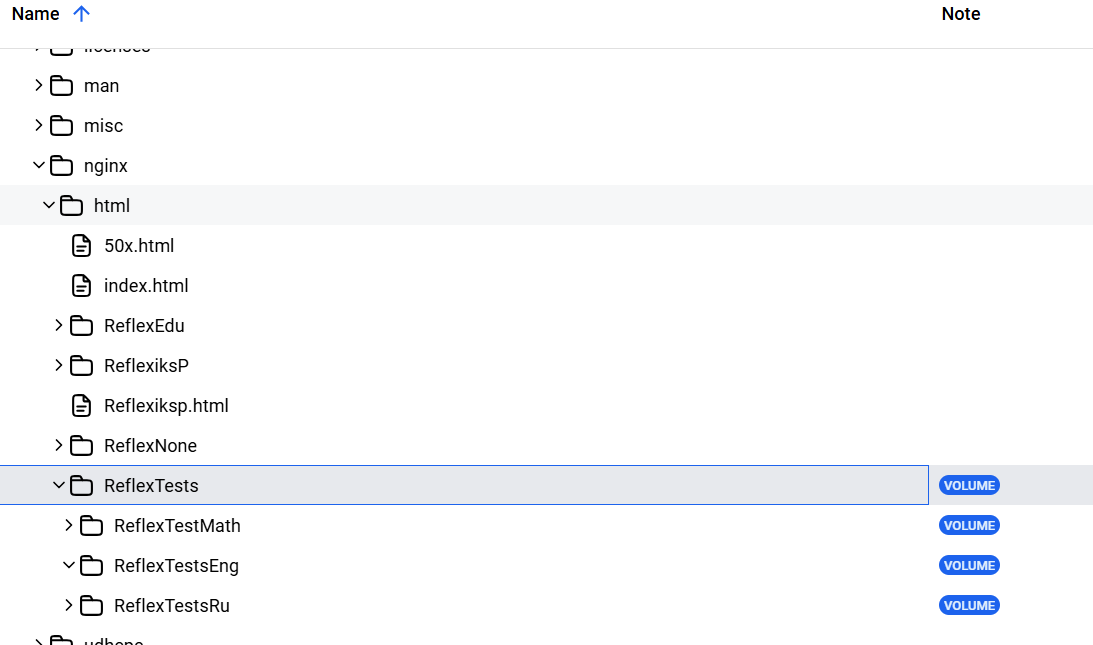


Рисунок 58 - Результат содания volume

Теперь можно копировать обновления внутрь контейнера.

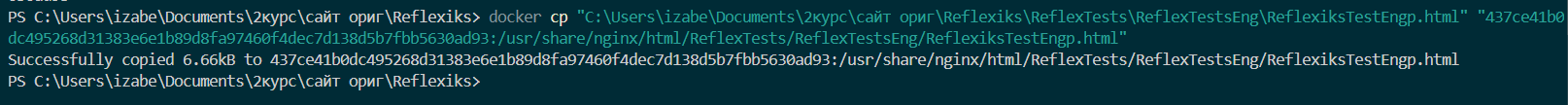


Рисунок 59 - Копирование обновлений

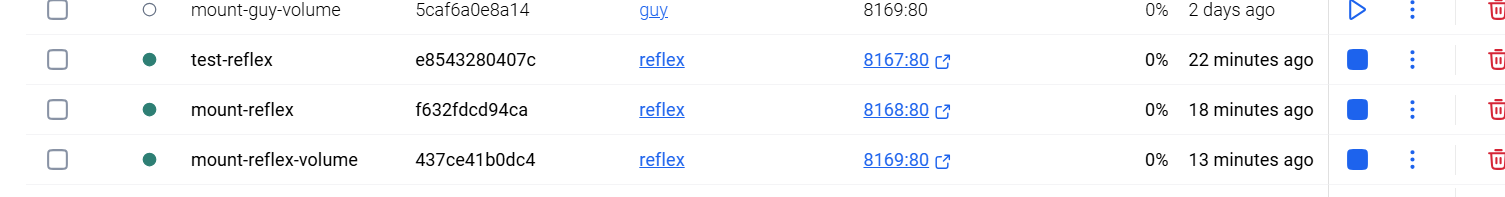


Рисунок 60 – Результаты

Вывод: был развернут сайт, меняющий тесты через Bind Mounts и через Volume. Описана работа приложения, компоненты и разница между ними.