Docker консультация

Наша задача — помочь студентам докеризировать их проекты.
Необходимо убедиться, что проект запускается и работает на компьютере студента.
Докеризировать и “собирать” неработающие или неготовые проекты - бессмысленно!
ТИПИЧНЫЙ ПРОЕКТ для докеризации будет включаться в себя два основных компонента: фронтенд и бэкенд. Опционально - база данных.
В нем реализованы основные функции, включая авторизацию, регистрацию, загрузку изображений, создание и управление постами, лайки, комментарии, а также чаты в реальном времени через WebSocket.

Фронтенд:

- **Технологии:** React, Redux, TypeScript, Tailwind CSS, Vite.

Использование React и Redux позволяет эффективно управлять состоянием приложения, а TypeScript обеспечивает статическую типизацию, повышая надежность кода. Tailwind CSS используется для стилизации, а Vite — для быстрой сборки и разработки.

Бэкенд:

- **Технологии:** Node.js, Express, MongoDB.

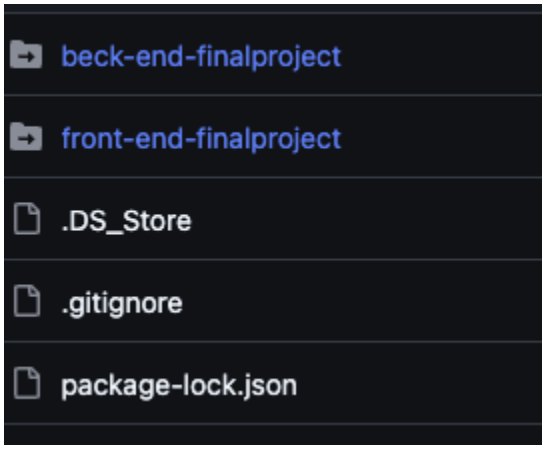
Node.js и Express обеспечивают надежную и масштабируемую серверную часть, а MongoDB служит для хранения данных.

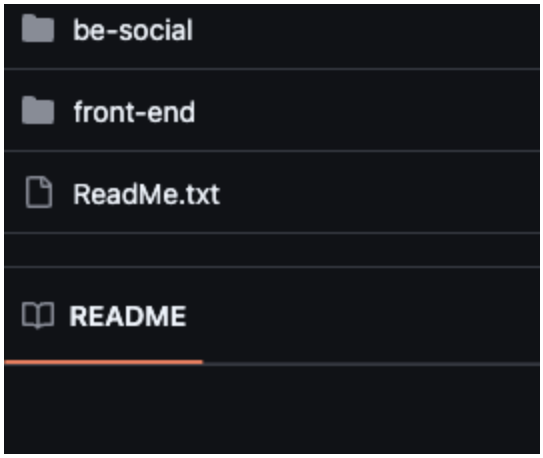
База данных:

В данном проекте используется MongoDB для хранения данных. Возможна интеграция с MongoDB Atlas — облачным сервисом, предоставляющим бесплатные кластеры. В некоторых случаях могут использоваться другие базы данных, такие как MySQL или PostgreSQL.

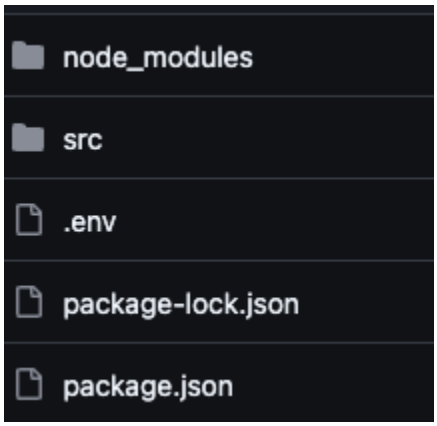
Примеры проектов:

В таком виде чаще всего будут проекты для докеризации:





И совершенно неготовый проект, состоящий из одного компонента (тут только бекенд):



https://github.com/it-career-hub/Final_ICHGRAM - пример проекта

План действий :

1. Пишем Dockerfile для бекенда
2. Добавляем .dockerignore файл для бекенда
3. Пишем Dockerfile для фронтенда
4. Добавляем .dockerignore файл для фронтенда
5. Собираем docker image бекенда
6. Собираем docker image фронтенда

7. Пишем docker-compose
8. Исправляем пути к dockerfile в build context docker-compose
9. Обрабатываем .env файл в docker-compose через его подключение, исключаем при сборке в .dockerignore (Безопасность!)
10. Исправляем конфигурационные файлы проектов при необходимости подключения не к localhost, а к докер контейнерам с сервисами
11. Опционально в docker-compose добавляем сервис с базой данных
12. Запускаем весь docker-compose стек

Пишем Dockerfile для бекенда

Уточняем версию node !

```
Unset
FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5000
EXPOSE 5003
CMD [ "npm", "run", "dev" ]
```

Используется официальный базовый образ Node.js версии 20.
Устанавливает рабочую директорию внутри контейнера как /app.
Копирует файлы package.json и package-lock.json из текущей директории в контейнер.
Устанавливает зависимости проекта, указанные в package.json.
Копирует все файлы проекта в рабочую директорию контейнера.
Делает порты 5000 и 5003 доступными для внешнего мира (обычно для доступа к API или другому сервису приложения).

Определяет команду для запуска приложения. В данном случае, это npm run dev, что указывает на использование режима разработки.

Альтернативно entrypoint контейнера может быть таким (если нужно запускать несколько сервисов)

```
Unset
CMD ["sh", "-c", "npm run dev & npm run chat && wait"]
```

Что именно запускаем при старте можно посмотреть в package.json в разделе "scripts"
Например :

```
Unset
{
  "name": "ichgram",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

Здесь для запуска будет **npm run test**

```
Unset
"scripts": {
    "start": "node ./src/server.js",
    "dev": "nodemon ./src/server.js"
  },
```

Здесь для запуска будет **npm run start** или **npm run dev**

Создание .dockerignore и добавление node_modules в него

важно добавлять в .dockerignore директорию node_modules
При копировании файлов в контейнер в процессе сборки Docker образа, вся директория проекта передается в контекст сборки. Если включить node_modules - это описано в

```
Unset
COPY . .
```

это не только значительно увеличит объем конечного docker image, но и может вызвать конфликт и ошибку сборки!
Пакеты в node_modules установлены для вашей локальной машины и могут содержать двоичные файлы, специфичные для вашей операционной системы.
Пример : пакет bcrypt из node_modules, устанавливаемый на Windows ОС, после копировании в процессе сборке образа не запускается, так как он специфичен для Windows и не работает на Linux!

Проверяем сборку docker image:

Переходим в директорию с бэкэндом, убеждаемся в наличии Dockerfile и собираем:

```
Unset
docker build . -t ichgram
```

Пишем Dockerfile для фронтенда

Dockerfile очень похож:

```
Unset
FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5173
CMD [ "npm", "run", "devdocker" ]
```

Саму команду запуска смотрим в package.json в разделе scripts

Не забываем про соединение фронта и бека!

- **Использование имени контейнера: При работе фронтенда и бэкенда в контейнерах, для подключения используется имя контейнера с бэкендом (например, ichgram), а не localhost. Это возможно благодаря Docker DNS, который автоматически сопоставляет имена контейнеров с их IP-адресами внутри одной сети.**
- **Почему не localhost: localhost указывает на контейнер, где выполняется запрос, а не на хостовую машину или другой контейнер.**

Пример из файла FE/src/api/api.ts:

```
Unset
import axios from "axios";

const base_url = "http://localhost:5005/api";

export const $api = axios.create({ baseURL: base_url });

$api.interceptors.request.use((config) => {
  const token = localStorage.getItem("token"); // ???
  config.headers.Authorization = token ? `Bearer ${token}` : "";
```

```
  return config;
});


export const socketURL = "http://ichgram:5005";
```

важно : http://ichgram:5005 - это имя docker контейнера!
Здесь уже указано имя контейнера (ichgram), что позволяет фронтенду связаться с бэкендом через встроенную DNS-службу Docker.

Проверяем сборку docker image:

Переходим в директорию с фронтендом, убеждаемся в наличии Dockerfile и собираем:

```
Unset
docker build . -t frontichgram
```

VITE - разрешить коннект с 0.0.0.0

При использовании Vite для сборки и разработки приложения важно правильно настроить сервер разработки, чтобы разрешить подключение не только с localhost, но и с любого сетевого интерфейса, например, с IP-адреса контейнера в Docker-среде.

Разрешение подключения по адресу 0.0.0.0**:**

- В конфигурации vite.config.ts параметр host: true в блоке server позволяет серверу разработки слушать подключения на всех сетевых интерфейсах (0.0.0.0). Это необходимо, чтобы фронтенд был доступен не только на localhost, но и для других сервисов (например, бэкенда) внутри Docker.

Пример из файла vite.config.ts**:**

```
Unset


export default defineConfig({
  server: {
    host: true, // Разрешает подключение с любого IP-адреса
    port: 5173  // Устанавливает порт для сервера разработки
  },
  plugins: [react()],
  resolve: {
    alias: {
      'i18next': path.resolve(__dirname, 'node_modules/i18next'),
    },
  },
});
```

- host: true**:** Включает прослушивание на 0.0.0.0.
- port: 5173**:** Устанавливает порт для работы Vite-сервера разработки.

По умолчанию Vite сервер слушает только на localhost, что ограничивает доступ к фронтенду. При работе в Docker контейнере или на удаленной машине это делает фронтенд недоступным извне.

- Указание host: true позволяет фронтенду взаимодействовать с бэкендом в другом контейнере или быть доступным для разработчиков из локальной сети.

Пишем docker-compose

Unset

```
services:
  ichgram:
    build:
      context: ./BE
      dockerfile: dockerfile
    container_name: ichgram
    ports:
      - "5000:5000"
      - "5005:5005"
    # depends_on:
    #   - mongo
    # volumes:
    #   - .env:/app/.env
  frontichgram:
    build:
      context: ./FE
      dockerfile: dockerfile
    container_name: frontichgram
    ports:
      - "5173:5173"
 mongo-database:
    container_name: mongo-database
    image: mongo:7
    restart: always
    ports:
      - 27017:27017
    command: --auth --bind_ip 0.0.0.0
    environment:
       MONGO_INITDB_ROOT_USERNAME: ${MONGO_ROOT_USERNAME}
       MONGO_INITDB_ROOT_PASSWORD: ${MONGO_ROOT_PASSWORD}
       MONGO_INITDB_DATABASE: ${MONGO_INITDB_DATABASE}
    env_file:
      - .env
    volumes:
      - ./data/db:/data/db
volumes:
  mongo-data:
```

В этом блоке описаны два основных сервиса приложения: бэкенд (ichgram) и фронтенд (frontichgram) и база данных, которая может не использоваться, так как выбран mongo atlas.

build**:**

- Указывает путь к исходному коду бэкенда (context: ./BE) и (context: ./FE)

и файл dockerfile, который будет использован для сборки Docker-образа. Имя dockerfile важно, особенно если регистр разный! Dockerfile и dockerfile - критически разные!

container_name: frontichgram**:**
container_name: ichgram:

- Назначает контейнеру имя frontichgram, чтобы другие контейнеры могли обращаться к нему по этому имени.

База данных в docker

1. container_name: mongo-database**:**

   - Устанавливает имя контейнера как mongo-database. Это имя можно использовать для обращения к контейнеру в пределах Docker-сети (например, для подключения от бэкенда).

2. image: mongo:7**:**

   - Указывает, что для контейнера будет использован официальный образ MongoDB версии 7. **Убедитесь, что разработка велась с такой же версией базы данных!**

3. restart: always:

   - Обеспечивает автоматический перезапуск контейнера в случае его остановки и автоматический запуск контейнера при старте docker

4. ports: - 27017:27017**:**

   - Пробрасывает порт MongoDB (27017) из контейнера на хостовую машину. Это позволяет обращаться к базе данных с хостовой машины через этот порт.

5. command: --auth --bind_ip 0.0.0.0**:**

   - Включает аутентификацию (--auth) и разрешает доступ ко всем сетевым интерфейсам (--bind_ip 0.0.0.0), чтобы база данных была доступна из других

контейнеров. Важно, если нужно дать возможность подключаться с любого хоста и через mongo compass или Mysql Workbench (при использовании MySQL)

6. environment**:**

- Переменные окружения задают параметры для инициализации MongoDB:
  - MONGO_INITDB_ROOT_USERNAME: Имя суперпользователя MongoDB.
  - MONGO_INITDB_ROOT_PASSWORD: Пароль суперпользователя.
  - MONGO_INITDB_DATABASE: Имя базы данных, которая создается при запуске.
- Эти значения читаются из .env файла для удобства настройки.

7. env_file: - .env**:**

- Подключает файл .env, откуда считываются переменные окружения. Это позволяет скрыть чувствительную информацию (например, пароли) от самого файла docker-compose.yml. Полезно при отправке в github, сами пароли лежат в .env файле, которые мы не отправляем!

8. volumes: - ./data/db:/data/db**:**

- Подключает локальную директорию ./data/db как volume для хранения данных MongoDB. Это обеспечивает их сохранение между перезапусками контейнера.

9. volumes: mongo-data**:**

- Объявляет volume mongo-data. Хотя он не используется в явной привязке в этом блоке, он может быть задействован для других целей (например, если нужно заменить локальную директорию).

.env файл, безопасное хранение секретов

В начальной версии, при написании Dockerfile мы использовали

```
Unset
COPY . .
```

Существенным недостатком является копирование файла .env, который лежит в директории фронтенда и бэкенда. В этом файле содержаться ключи и пароли, которые будут являться частью финального docker image. Если запушить такой image в dockerhub , то любой сможет его скачать и получить пароли.
Для того, чтобы этого избежать, добавляем **.env файл в файл .dockerignore** и не забываем смонтировать его в docker-compose при запуске!

```
Unset
   volumes:
       - .env:/app/.env
```

Скорректируйте путь к .env файлу, они могут быть разными для проекта в целом, для фронтенда и бэкенда!
Таким образом мы “подкладываем” .env непосредственно перед запуском, а не во время сборки. Сам файл не отправляется ни в github ни в dockerhub

Зависимости при запуске и healthcheck

```
Unset
depends_on: - mongo
```

указывает, что этот контейнер должен запускаться после mongo. Это гарантирует, что контейнер с базой данных будет **запущен раньше** бэкенда.

Для тонкой настройки процесса запуска используется healthcheck:

Unset

```
mongo-database:
  container_name: mongo-database
  image: mongo:7
  restart: always
  ports:
    - 27017:27017
  command: --auth --bind_ip 0.0.0.0
  environment:
    MONGO_INITDB_ROOT_USERNAME: ${MONGO_ROOT_USERNAME}
    MONGO_INITDB_ROOT_PASSWORD: ${MONGO_ROOT_PASSWORD}
    MONGO_INITDB_DATABASE: ${MONGO_INITDB_DATABASE}
  env_file:
    - .env
  volumes:
    - ./data/db:/data/db
  healthcheck:
    test: ["CMD", "mongo", "--eval",
"db.adminCommand('ping')"]
    interval: 30s
    timeout: 10s
    retries: 5
ichgram:
  build:
    context: ./BE
    dockerfile: dockerfile
  container_name: ichgram
  ports:
    - "5000:5000"
    - "5005:5005"
  depends_on:
    mongo-database:
      condition: service_healthy
```

healthcheck для mongo-database:

1. test**:**

   - Проверяет доступность MongoDB

Команда выполняет простой запрос ping, чтобы убедиться, что MongoDB работает и отвечает.

2. interval**:**

   - Интервал между проверками. Здесь каждые **30 секунд**.

3. timeout**:**

   - Максимальное время ожидания ответа от MongoDB. Здесь это **10 секунд**.

4. retries**:**

   - Количество попыток перед признанием сервиса недоступным. Здесь до **5 раз**.

5. depends_on**:**

   - Указывает, что сервис ichgram должен запускаться только после того, как mongo-database будет признан “здоровым”.

6. condition: service_healthy**:**

   - Проверяет статус “здоровья” MongoDB, определенный через healthcheck. Контейнер бэкенда стартует только после успешного прохождения всех проверок.

Healthcheck для MongoDB:

- Проверка с помощью команды ping обеспечивает уверенность, что база данных работает корректно и готова к приему запросов.

- Это важно, так как MongoDB может запускаться дольше других сервисов, особенно при необходимости выполнения инициализации (например, создания пользователя или базы данных).

Depends_on с условием service_healthy**:**

- Обеспечивает правильный порядок запуска сервисов.

- Например, бэкенд не должен пытаться подключиться к MongoDB до тех пор, пока база данных не будет готова.