Poniżej — krok-po-kroku instrukcja uruchomienia frontendu i backendu Task.io + omówienie typowych problemów.

---

1) Co będzie potrzebne

- **VS Code**
- **Docker Desktop**
- **Terminal z narzędziami Unix** (na Windows wystarczy Git Bash, na macOS Terminal, na Linuksie dowolne CLI)
- **Node.js z npm** dla frontendu

Jeśli **npm nie jest zainstalowany**:

- Najszybciej na dowolnym systemie — przez **nvm** (Node Version Manager):

    1. Zainstaluj nvm według instrukcji na jego stronie (zwykle jedno polecenie w terminalu).

    2. Następnie:

    ```
    nvm install --lts
    nvm use --lts
    node -v
    npm -v
    ```

- Alternatywa: zainstaluj **Node.js LTS** z oficjalnej strony, używając instalatora dla Twojego systemu.

---

2) Uruchomienie frontendu

1. Otwórz **VS Code**.

2. Menu **File** → **Open Folder…** i wybierz folder **`frontend`** projektu.

3. Otwórz wbudowany terminal VS Code (**Terminal** → **New Terminal**) i wykonaj:

    ```
    npm install
    ```

    Po instalacji pojawi się folder **`node_modules`**. To znak, że zależności frontu są gotowe.

4. Uruchom dev-serwer frontu z katalogu głównego **`frontend`**:

    ```
    npm run dev
    ```

5. Domyślnie frontend będzie dostępny pod: **http://localhost:3000/ (http://localhost:3000/)**

---

3) Uruchomienie backendu

1. Uruchom **Docker Desktop** i poczekaj, aż będzie w pełni gotowy.

2. W terminalu przejdź do katalogu głównego backendu (folder **`Task.io`**, gdzie leży `docker-compose.yml`).

3. Zbuduj i podnieś wszystkie serwisy:

```
docker-compose up --build -d
```

   Jeśli chcesz obserwować logi na żywo, uruchom bez `-d` i zostaw okno otwarte.

4. Poczekaj, aż:

   - pobiorą się obrazy,
   - zbudują się serwisy,
   - zainstalują się zależności,
   - wystartują kontenery.

5. Gdy wszystko wstanie, Swagger UI backendu będzie tutaj: **http://localhost:8000/docs (http://localhost:8000/docs)**

6. Wejdź do Swagger UI, znajdź endpoint `GET /api/v1/auth/init-db` w rozdziale `auth`, kliknij **Try it out ➜ Execute**. To zainicjalizuje dane słownikowe w bazie (seed).

   Uwaga: operacja jest idempotentna — po jednokrotnym wykonaniu nie musisz jej powtarzać przy każdym starcie środowiska. Jeśli endpointu nie widać, upewnij się, że serwis auth działa poprawnie.

---

4) Szybka weryfikacja

- Frontend: **http://localhost:3000 (http://localhost:3000)**
- Backend (Swagger): **http://localhost:8000/docs (http://localhost:8000/docs)**

---

5) Częste problemy i rozwiązania

5.1 Problem z importami w generowanych plikach gRPC (`*_pb2.py`, `*_pb2_grpc.py`)

Objaw: błędy typu `ModuleNotFoundError: No module named 'pm_pb2'` albo kłopoty z importami względnymi na początku pliku `*_pb2_grpc.py`.

Dlaczego tak bywa: generowane pliki używają albo **importu względnego**

```
from . import pm_pb2 as pm__pb2
```

albo **importu absolutnego**

```
import pm_pb2 as pm__pb2
```

i zależy to od tego, czy Python widzi katalog jako **pakiet** (obecność `__init__.py`) i od sposobu uruchomienia modułu.

Szybkie rozwiązanie:

- Spróbuj zamienić import na górze `*_pb2_grpc.py` na przeciwne rozwiązanie. Na przykład:
  - było `from . import pm_pb2 as pm__pb2`, zrób `import pm_pb2 as pm__pb2`,
  - i odwrotnie.
- Upewnij się, że w katalogu z modułami proto jest pusty plik **`__init__.py`**. To pozwoli używać importów względnych.
- Jeśli problem nie zniknął, przejdź do „pełnej regeneracji” (niżej).

Pełna regeneracja i lokalne sprawdzenie serwisów

Czasem wygodniej otworzyć każdy serwis w osobnym IDE i zbudować środowisko.

Wariant A: przez PyCharm

1. Otwórz serwis jako osobny projekt.
2. Skonfiguruj wirtualne środowisko:
   - **PyCharm → Settings → Project → Python Interpreter → Add → Virtualenv**
   - Utwórz środowisko w folderze projektu, wybierz wersję Pythona.
3. Zainstaluj zależności:

```
pip install -r requirements.txt
```

Wariant B: przez CLI

1. W katalogu głównym serwisu utwórz venv:

```
python -m venv .venv
```

2. Aktywuj venv:
   - Windows (Git Bash): `source .venv/Scripts/activate`
   - macOS/Linux: `source .venv/bin/activate`
3. Zainstaluj zależności:

```
pip install -r requirements.txt
```

Regeneracja modułów gRPC Poniżej „szkic” komend i co oznacza każdy flag.

- **Dla Gateway (przykład: auth w Gateway)**

```
python -m grpc_tools.protoc \
  -I ./proto/auth \
  --python_out=./proto/auth \
  --grpc_python_out=./proto/auth \
  ./proto/auth/auth.proto
```

Gdzie: `-I` — katalog, gdzie szukać *.proto i ich include; `--python_out` — gdzie umieścić wygenerowane `*_pb2.py`; `--grpc_python_out` — gdzie umieścić `*_pb2_grpc.py`; ostatni argument — ścieżka do *.proto.

- **Dla samego Auth Service**

```
python -m grpc_tools.protoc \
  -I app/grpc_app \
  --python_out=app/grpc_app \
  --grpc_python_out=app/grpc_app \
  app/grpc_app/auth.proto
```

- **Analogicznie dla innych serwisów** (podstaw swoje ścieżki i nazwy plików): Project Management Service:

```
python -m grpc_tools.protoc \
  -I app/grpc_app \
  --python_out=app/grpc_app \
  --grpc_python_out=app/grpc_app \
  app/grpc_app/pm.proto
```

Task Service:

```
python -m grpc_tools.protoc \
  -I app/grpc_app \
  --python_out=app/grpc_app \
  --grpc_python_out=app/grpc_app \
  app/grpc_app/task.proto
```

Analytics Service:

```
python -m grpc_tools.protoc \
  -I app/grpc_app \
  --python_out=app/grpc_app \
  --grpc_python_out=app/grpc_app \
  app/grpc_app/analytics.proto
```

Wskazówka: po regeneracji sprawdź, czy w folderach, z których importowane są generowane moduły, istnieje `__init__.py`. W razie potrzeby dodaj.

5.2 Nie wyświetlają się awatary lub obrazki projektów (dostęp do MinIO)

Objaw: frontend wstał, ale awatary użytkowników lub ikony projektów są niewidoczne.

Przyczyna: buckety w MinIO mają ustawioną politykę prywatną. Aby umożliwić publiczne czytanie statycznych plików, trzeba nadać dostęp do odczytu dla bucketów **`avatars`** i **`project-avatars`**.

Wariant 1. Przez MinIO Console (najprościej)

1. Otwórz MinIO Console: zazwyczaj **http://localhost:9001 (http://localhost:9001)**.
2. Zaloguj się danymi z `docker-compose.yml` (standardowo `minio` / `minio_supersecret`, jeśli nie było zmienione).
3. Otwórz **Buckets**, wybierz `avatars`.
4. Znajdź **Access Policy** i ustaw politykę **Public** (lub „Read Only” dla anonimowych). Zapisz.
5. Powtórz to samo dla `project-avatars`.

Wariant 2. Przez `mc` (MinIO Client) w Dockerze

Jeśli wolisz polecenia:

1. Uruchom tymczasowy kontener z `mc` i ustaw alias:

```
docker run --rm -it --network=host minio/mc \
  mc alias set local http://localhost:9000 MINIO_ROOT_USER MINIO_ROOT_PASSWORD
```

2. Zezwól na anonimowe pobieranie dla potrzebnych bucketów:

```
docker run --rm -it --network=host minio/mc mc anonymous set download local/avatars
docker run --rm -it --network=host minio/mc mc anonymous set download local/project-avatars
```

Teraz obiekty z tych bucketów można czytać bez autoryzacji.

Uwaga: na Windows flaga `--network=host` może nie działać. W takim przypadku użyj adresu `http://host.docker.internal:9000` zamiast `http://localhost:9000`, albo zainstaluj `mc` lokalnie i wykonaj te same polecenia.

---

6) Przydatne polecenia na co dzień

Podgląd logów wszystkich kontenerów:

```
docker-compose logs -f
```

Przebudowanie i restart:

```
docker-compose up --build -d
```

Zatrzymanie i usunięcie kontenerów:

```
docker-compose down
```

Pełne czyszczenie (z usunięciem wolumenów danych):

```
docker-compose down -v
```

---

7) Gdzie zaglądać w przeglądarce

- Frontend: **http://localhost:3000 (http://localhost:3000)**
- Swagger backendu: **http://localhost:8000/docs (http://localhost:8000/docs)**