Docker勉強会

- まえがき
  BookStationで学習もとい研修を行う上での問題点（**環境構築の面倒くささ**）を解消するために、**Docker**を使用して簡単に環境を作成できるようにしていきます。
  ※事前にBook Stationがどのようなアプリケーションであるかを理解しておくと内容がすんなり入ってくると思います。
  ※そもそもDockerって何？コンテナって何？という方でも環境構築できる資材は準備していますので、ぜひ読んでください。

成果物

今回環境構築するために作成・編集したファイルは以下の通りです。
以下のファイル群を用意して、Dockerコマンド2回実行するだけで、サーバーの構築、起動までが完了します。

```
Book Station/
├── books/
│   ├── backend
│   ├── front
│   └── :                    #booksリポジトリをCloneしたものであるため省略
├── db/
│   └── init_db/
│       └── test.sql         #初期データ登録用
├── docker/
│   ├── api/
│   │   └── Dockerfile       #apiコンテナ作成用
│   ├── app/
│   │   └── Dockerfile       #appコンテナ作成用
│   └── db/
│       ├── Dockerfile       #dbコンテナ作成用
│       └── my.cnf           #MySQLの文字コード設定用
├── .env                     #MySQLで使用する環境変数を記述
└── docker-compose.yml       #dockercomposeファイル
```

次の章からコンテナ環境の作成手順をファイル毎に説明していきます。

コンテナ作成手順

appコンテナ作成

まずはBook Staitonを構成するWebサーバの部分のコンテナから作成/定義していきます。

docker/app/Dockerfile

```
FROM node:12.13-alpine
```

```
WORKDIR /app

RUN npm install
```

FROM句では、コンテナのベースイメージを決定しています。（今回はalpine上で動作するnode.js v12.13を選んでいます。）

WORKDIR句では、コンテナ内のどこで処理を行うかpathを指定しています。今回の場合、立ち上がったappコンテナ内の`/app`以下でRUN句以降の処理を行っている、ということになります。

RUN句では、`npm install`を行い、front側で必要なライブラリ等を準備していきます。

//TODO
`npm install`を実行してもらう予定でしたが、実行されず。。仕方なく、ローカルで`npm install`を実行してnode_moduleをインストールしました。(backend側も同様)

以下は、`Docker-compose.yml`のappコンテナの定義になります。
毎回起動コマンドを実行するのも面倒なので、ymlファイルに`npm run serve`を記述して、コンテナ起動時に代わりに実行してもらいます。

/BOOK STATION/docker-compose.yml

```
app:
  container_name: app_container   # コンテナ名
  build: ./docker/app             # Dockerfileの配置場所
  ports:
    - 8080:8080                   # ポートフォワーディングの設定
  volumes:
    - ./books/front:/app          # Vueプロジェクトのfrontフォルダとコンテナ内のappをマウント
  stdin_open: true                # ホストマシンの入力をコンテナに伝えるための記述
  tty: true                       # プロセスが終了した際に、コンテナを終了させないための記述
  environment:
    TZ: Asia/Tokyo                # タイムゾーンを変更
  command: npm run serve          # コンテナ起動時に実行するコマンド
  networks:
    - default                     # コンテナ間で通信のために使用するネットワークを明示
```

apiコンテナ作成

次に、Book Staitonを構成するAPIサーバの部分のコンテナを作成/定義していきます。
とは言え、イメージの作成まではappコンテナとほぼ同じものになります。

docker/api/Dockerfile

```
FROM node:12.13-alpine

WORKDIR /api

RUN npm install
```

FROM句では、コンテナのベースイメージを決定しています。（appコンテナ同様に、alpine上で動作するnode.js v12.13を選んでいます。）

WORKDIR句では、コンテナ内のどこで処理を行うかpathを指定しています。今回の場合、立ち上がったappコンテナ内の`/app`以下でRUN句以降の処理を行っている、ということになります。

RUN句では、`npm install`を行い、backend側で必要なライブラリ等を準備していきます。

//TODO
Appコンテナ同様に、`npm install`を実行してもらう予定でしたが、実行されず。。仕方なく、ローカルで`npm install`を実行してnode_moduleをインストールしました。

以下は、`docker-compose.yml`のapiコンテナの定義になります。
こちらもappコンテナ同様、ymlファイルに`npm run start`を記述して、コンテナ起動時に代わりに実行してもらいます。

/BOOK STATION/docker-compose.yml

```
api:
  container_name: api_container   # コンテナ名
  build: ./docker/api             # Dockerfileの配置場所
  ports:
    - 3000:3000                   # ポートフォワーディングの設定
  volumes:
    - ./books/backend:/api        # Vueプロジェクトのfrontフォルダとコンテナ内のappをマウント
  tty: true                       # プロセスが終了した際に、コンテナを終了させないための記述
  environment:
    CHOKIDAR_USEPOLLING: 1        # ホットリロードの有効化
    TZ: Asia/Tokyo                # タイムゾーンを変更
  depends_on:
    - db                          # dbが起動した後にapiを起動するための指定
  command: npm run start          # コンテナ起動時に実行するコマンド
  networks:
    - default                     # コンテナ間で通信のために使用するネットワークを明示
```

上記のコンテナの作成に加えて、AppサーバとAPIサーバ間の通信を実現するために、`vue.config.js`を編集します。
devserver > proxy > /api > target の値を`http://api:3000`とします。
（もともとは`http://localhost:3000`）

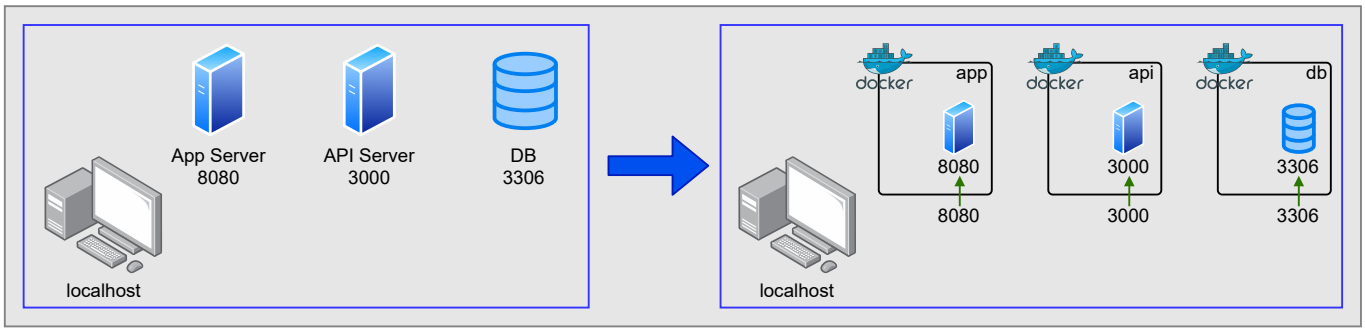
/BOOK STATION/books/front/vue.config.js（一部抜粋）

```
  devServer: {
    https: true,
    port: 8080,
    proxy: {
        '/api': {
            target: "http://api:3000",
```

```
                changeOrigin: true,
            }
        }
    }
```

これは、Book Stationの実行環境の違いによるものが原因となっています。

これまでの実行環境は、ホストPC内でサーバを起動していました。そのため、3000番ポートへのアクセスはlocalhostからそのままアクセス可能でした。
ただし、今回はそれぞれ独立したコンテナが3つ起動しているため、Appサーバがlocalhost:3000を参照しても、Appコンテナ内の3000番ポートを参照してしまう。といった状況になるのです。



では、何を指定することでAPIコンテナと通信することができるのか。という問題になると思います。
これはかなり都合の良い話なのですが、Docker-composeを使用して立ち上げたコンテナはデフォルトで、サービス名を指定するだけでアクセス可能となります。（Docker様々。。！）
今回の場合、`docker-compose.yml`で記載した「`api`」を指定するだけで通信が行えます。

/BOOK STATION/docker-compose.yml

```
  api:                              # サービス名はここ！
    container_name: api_container
    build: ./docker/api
    ports:
      - 3000:3000

# ～中略～
```

dbコンテナ作成

最後にDBサーバのコンテナを作成/定義していきます。

docker/db/Dockerfile

```
FROM mysql:8.0.39-debian

EXPOSE 3306

CMD ["mysqld"]

ADD ./my.cnf /etc/mysql/conf.d/my.cnf
```

```
RUN chmod 644 /etc/mysql/conf.d/my.cnf
```

FROM句では、コンテナのベースイメージを決定しています。（今回はdebian上で動作するmysql:8.0.39を選んでいます。）

EXPOSE句では、コンテナのポートを公開していることをDockerに伝えます。実際には不要とされることが多いコマンドではありますが、明示する意味も込めて記載する場合があるそうです。

CMD句では、`mysqld`を実行します。（デーモンの起動。MySQLを使用するためのおまじない程度に考えています。）

ADD句では、MySQLの文字コードを設定しているファイルを指定のフォルダにコピーします。
DB操作する際に文字化けしないように明示してあげます。

RUN句では、その設定ファイルに対して適切な権限を付与します。
設定ファイルが作成者以外からも書き込み可能になっていると、この設定ファイルが機能しないので要注意！！

以下は、`Docker-compose.yml`のappコンテナの定義になります。
コマンドの登録はありませんが、初期提供データの設定を記述しています。

/BOOK STATION/docker-compose.yml

```
  db:
    container_name: db_container                          # コンテナ名
    build: ./docker/db                                    # Dockerfileの配置場所
    ports:
      - 3306:3306                                         # ポートフォワーディングの設定
    volumes:
      - ./db/init_db:/docker-entrypoint-initdb.d          # 初期データ投入用のsqlファイルを
バインド
      - test_data:/var/lib/mysql                          # 永続化したデータをバインド
    environment:
      - MYSQL_DATABASE=${MYSQL_DATABASE}                  # コンテナの環境変数から各種設
定を読み込み
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - TZ="Asia/Tokyo"
    networks:
      - default                                           # コンテナ間で通信のために使用す
るネットワークを明示
```

上記のdocker-compose.ymlで使用している`.env`は以下の通りです。
DBの名前やアカウントの情報を別ファイルで管理することで、docker-compose.ymlに直書きすることを防いでいます。

BOOK STATION/.env

```
MYSQL_DATABASE=intern
MYSQL_USER=intern
MYSQL_PASSWORD=intern
MYSQL_ROOT_PASSWORD=intern
```

MySQLへの接続を実現するために、vue.config.jsを編集します。
APIサーバへのアクセス時の同様に、localhostではdbコンテナにアクセスできないため、サービス名を指定してDBとのやり取りを実現させます。

/BOOK STATION/books/backend/db/utility.js

```
const Sequelize = require("sequelize");

/**
* DBコネクション取得
* @returns DBコネクション
*/
module.exports.connect = function () {
  return new Sequelize("intern", "intern", "intern", {
    dialect: "mysql",
    host: "db",
    port: 3306,
    pool: {
      max: 5,
      min: 1,
      acquire: 30000,
      idle: 10000
    }
  });
}
```

上記の設定でdbコンテナの作成はほぼ完了となります。
後は、初期提供データを投入するためのクエリと文字コードなどの設定ファイルを用意して完了となります。

初期提供データのほうは割愛しますが、MySQLの設定ファイルは以下のようになっています。

BOOK STATION/docker/db/my.cnf

```
[mysqld]
character-set-server=utf8mb4
collation-server=utf8mb4_0900_ai_ci

[mysql]
default-character-set=utf8mb4

[client]
default-character-set=utf8mb4
```

この設定ファイルを適用するために、DockerfileのADD句以降の処理が必要でした。

コンテナイメージの作成

ここまでの手順で環境構築に必要なDockerfileと、docker-compose.ymlが完成しました。
設計図は出揃ったので、早速コンテナイメージの作成に取り掛かります。

とは言っても、以下のコマンドを実行するだけでコンテナイメージの作成は完了します。

```
docker-compose build
```

実行が完了したら、イメージがちゃんと作成できているか以下のコマンドでチェックします。

```
docker images
```

以下の様に3つのコンテナイメージが確認できていればOKです。

```
PS C:\Book Station> docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
bookstation-db    latest    f50eb46f4078   33 minutes ago   611MB
bookstation-app   latest    fbfea4cd0616   7 weeks ago      80.2MB
bookstation-api   latest    35ac3085f01e   7 weeks ago      80.2MB
PS C:\Book Station>
```

コンテナの作成

最後に以下のコマンドを実行して、コンテナイメージをコンテナ化します。

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

実行が完了したら、コンテナがちゃんと作成できているか以下のコマンドでチェックします。

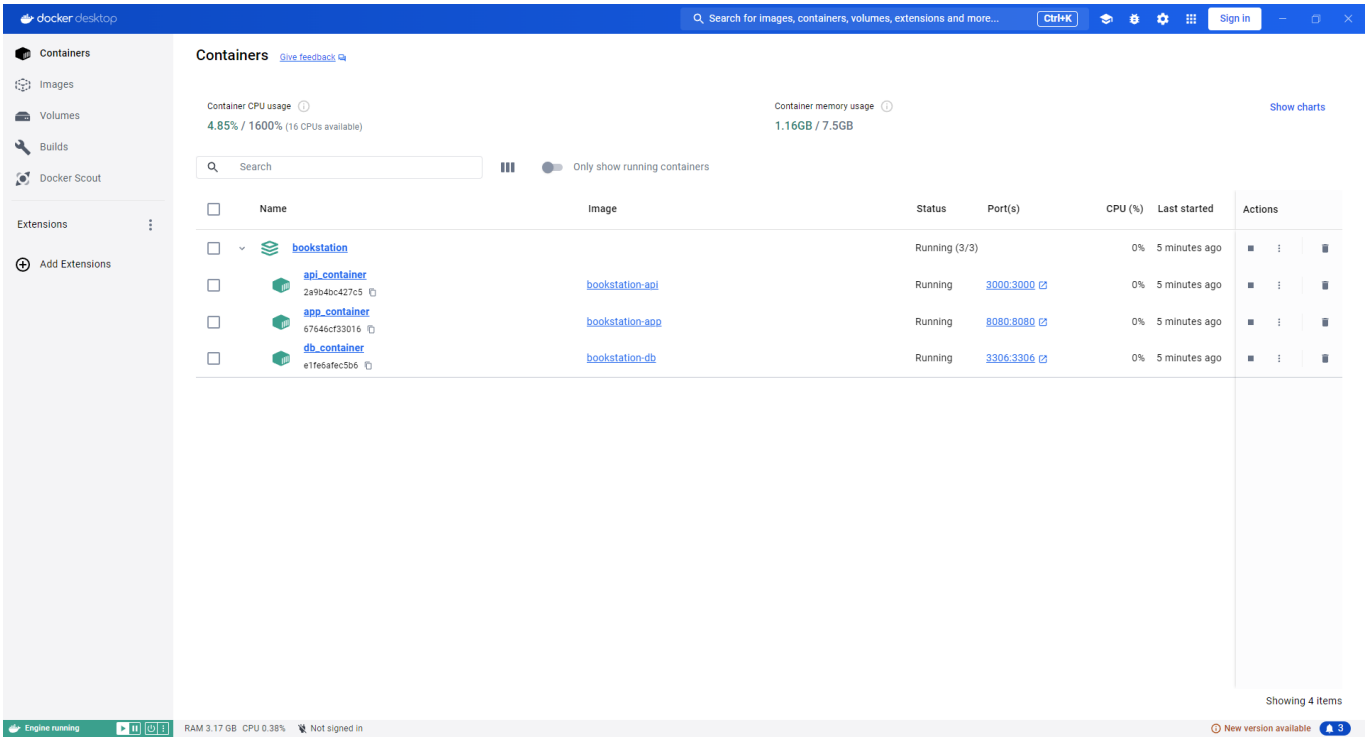
```
docker ps
```

以下の様に3つのコンテナが確認できていればOKです。

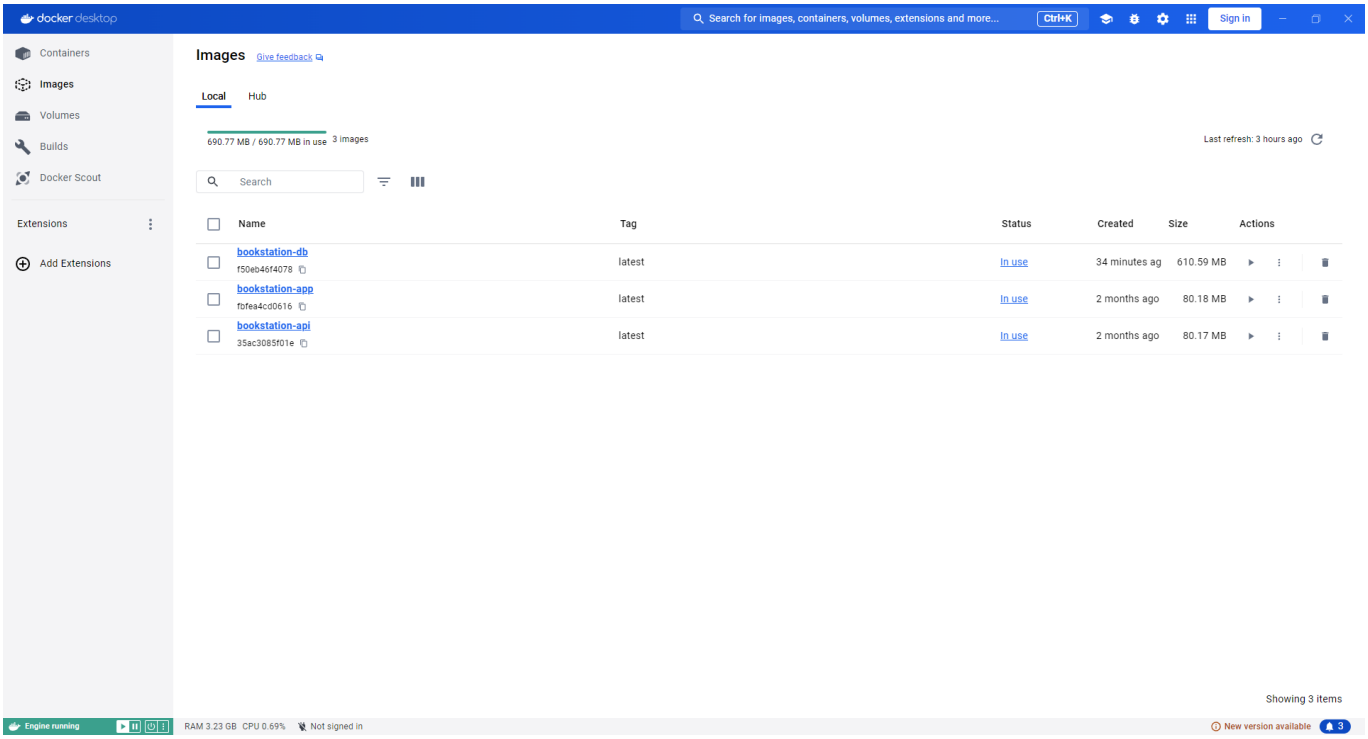
```
PS C:\Book Station> docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORTS                                NAMES
2a9b4bc427c5   bookstation-api   "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:3000->3000/tcp               api_container
67646cf33016   bookstation-app   "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:8080->8080/tcp               app_container
e1fe6afec5b6   bookstation-db    "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:3306->3306/tcp, 33060/tcp    db_container
PS C:\Book Station>
```

ちなみにコンテナイメージ、起動したコンテナについてはDocker Desktopからも確認可能です。

起動したコンテナ



作成したコンテナイメージ

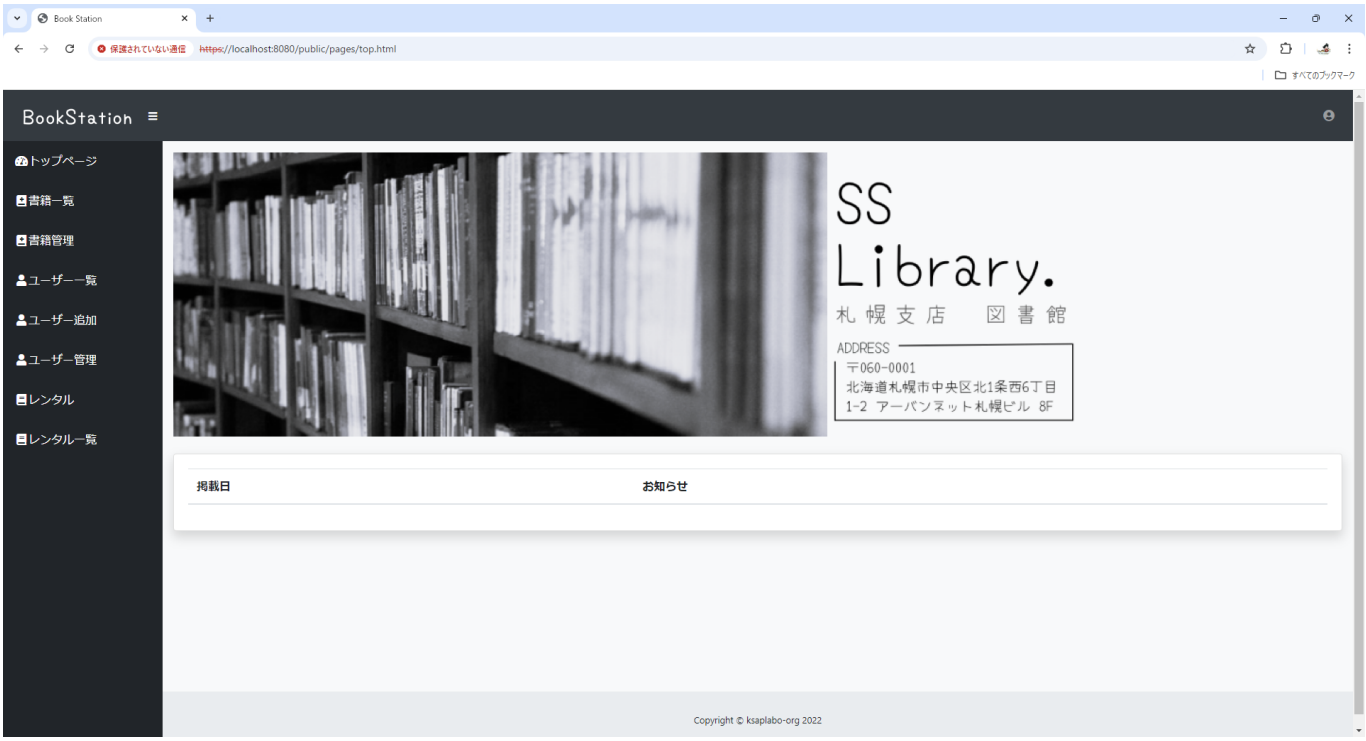


動作確認

後はいつも通りBook Stationにアクセスしてみましょう。アクセスするためのpathは以下の通りです。

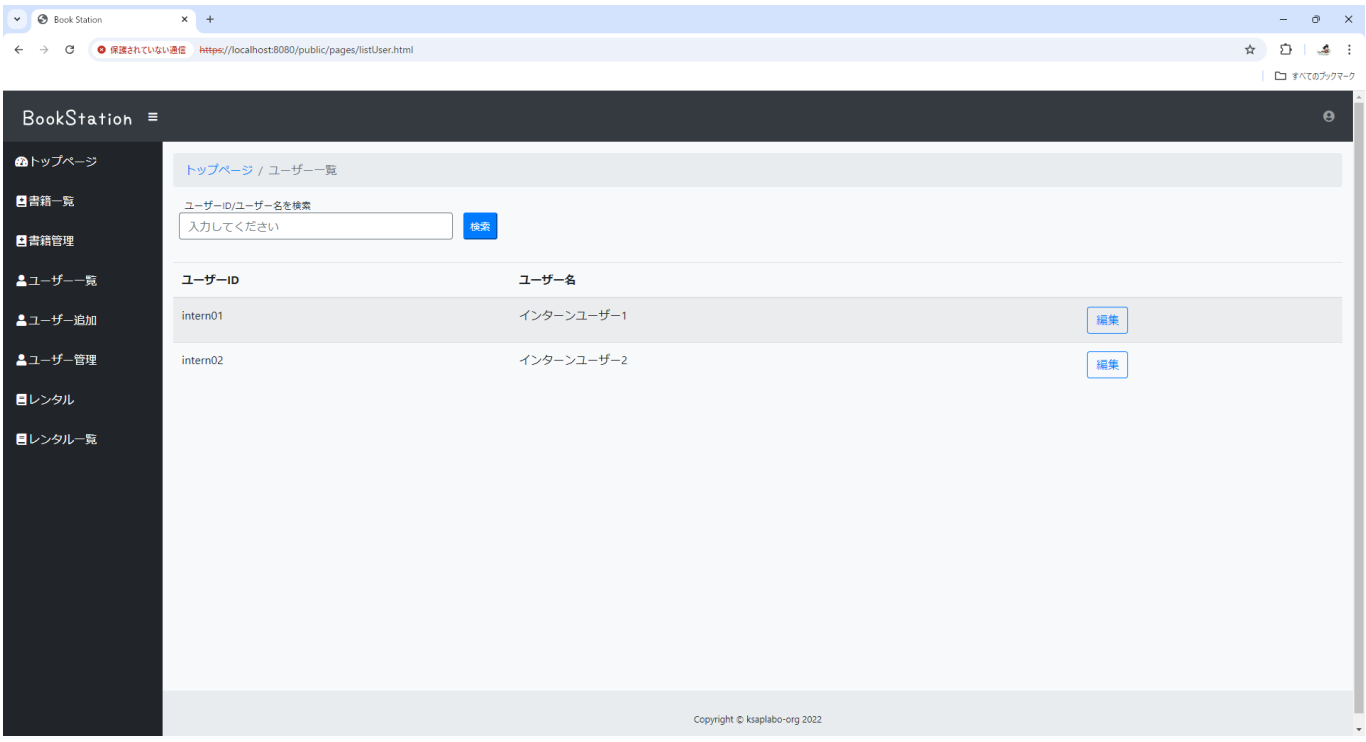
```
https://localhost:8080/public/pages/signin.html
```

ばっちりみたいです！（init.sqlをさぼったのでお知らせ情報が登録されていませんが。。）



念のため、ユーザー一覧を確認してみます。

ちゃんと初期提供データのSQLも実行できているみたいでよかったです！



Dockerコンテナを停止する際には以下のコマンドを実行してください。

```
docker-compose stop
```

まとめ

今回は、仮想化技術（Docker）を使用して環境構築の簡易化を実践してみました。
設計図（Dockerfile,docker-compose.yml）を作成するまではとても大変でしたが、一度作成してしまえば複数台のPCに対して、一瞬で環境構築ができてしまうのがとても大きなメリットとなっています。
また、それらのPC毎に環境差異が全くないことも大きな特徴となっています。（インターンで使えないかな。。）

Dockerでの環境構築に挑戦する方へ

以下に環境構築の手順を簡単に記載しておきますので、やってみたい方は是非！

1. cloneの作成

   ローカルにBooksリポジトリの内容をクローンしてください。
   ブランチは「**docker/docker-compose**」になります。

   ※クローンが作成出来たら、front,backendそれぞれの階層で`npm install`を実行してください。（本来であれば不要な手順ですが、まだそこまで対応できていないため）

2. コンテナ作成に必要なファイル群を適切な場所に配置

   本リポジトリから、環境構築に必要なファイルを**クローンしたブランチと同じ階層**に配置します。
   以下の配置になっていればOKです。（ルートディレクトリの名前は適当です）

```
Book Station/
├── books/
│   ├── backend
│   ├── front
│   └── :                    #booksリポジトリをCloneしたものであるため省略
├── db/
│   └── init_db/
│       └── test.sql         #初期データ登録用
├── docker/
│   ├── api/
│   │   └── Dockerfile       #apiコンテナ作成用
│   ├── app/
│   │   └── Dockerfile       #appコンテナ作成用
│   └── db/
│       ├── Dockerfile       #dbコンテナ作成用
│       └── my.cnf           #MySQLの文字コード設定用
├── .env                     #MySQLで使用する環境変数を記述
└── docker-compose.yml       #dockercomposeファイル
```

3. Docker Desktopのインストール

   以下からインストールしてください。（悪質なサイトを疑う方は自力で調べてください）

```
https://docs.docker.com/desktop/install/windows-install/
```

4. Docker Desktopの起動

インストールが完了したらDocker Desktopは起動したままにしてください。
このアプリが起動していないと、dockerコマンドが実行できず、コンテナが動作しないため注意してください。

5. コンテナの起動

   最後に以下のコマンド実行すればBook Stationにアクセスできるようになります。

```
docker-compose build

docker-compose up -d
```

6. コンテナの停止

   コンテナを停止したい場合は、以下のコマンドを実行してください。

```
docker-compose stop
```