運用メンバー用資料
2024/01/24

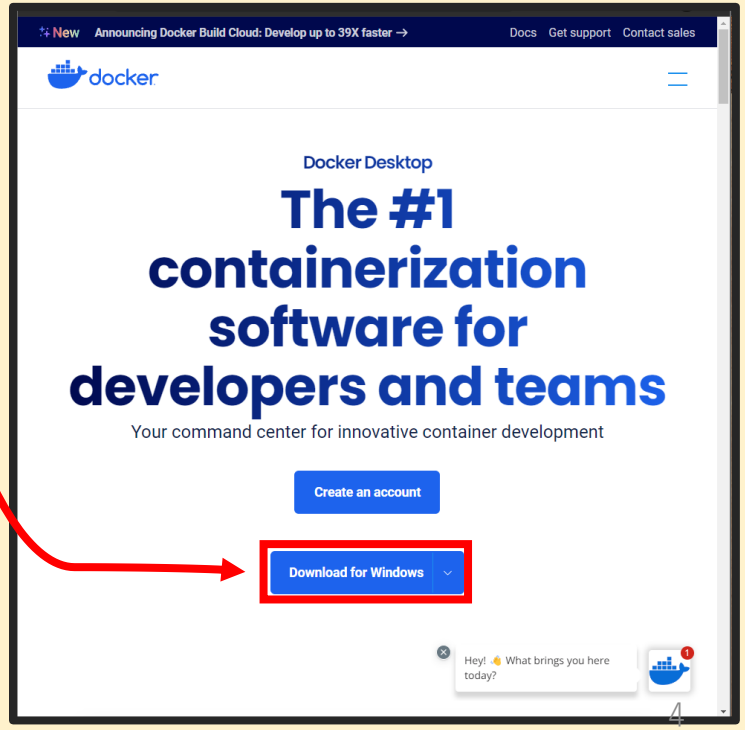
目次

環境構築（Windows）

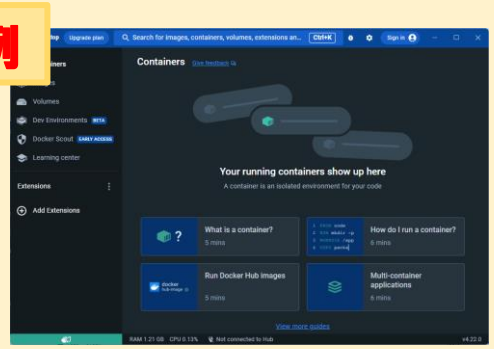
環境構築（Windows）

- Dockerのインストール
  - Docker Desktopのダウンロードサイトから、Docker Desktopをダウンロード
  - このボタンをクリックして、Windows版をダウンロード
  - インストーラをダウンロードできたら、指示に従って再起動
  - 再起動後、Docker Desktopを無事起動できれば成功

Docker Desktopのダウンロードサイト



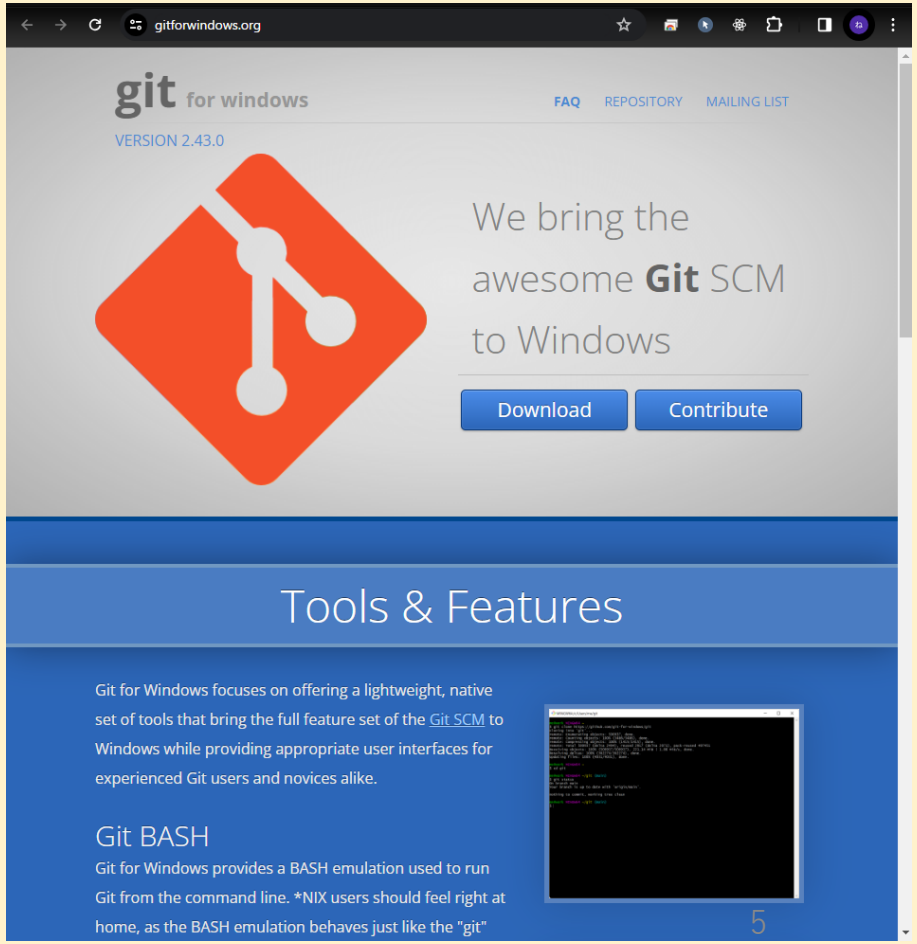
成功例



環境構築（Windows）

- Git・GitHubのインストールと基本設定
  - Gitのダウンロードサイトからインストーラをダウンロード
  - インストーラの実行後、設定用の画面が表示されるが、全て「Next」をクリック
  - 最後もチェックボックスをそのままにして「Finish」をクリック

Gitのダウンロードサイト



環境構築（Windows）

- Git・GitHubのインストールと基本設定
  - 右図1のように「Git Bash」があれば、Gitのインストールに成功
  - 「Git Bash」を起動すると、右図2のような画面が出現
  - git config --global user.name “ユーザー名”
    git config --global user.email “メールアドレス”
    以上の文を1行ずつ入力
    ※ ユーザー名とメールアドレスは任意

図1　Gitのインストールの成功例

図2「Git Bash」の起動画面

環境構築（Windows）

- Git・GitHubのインストールと基本設定

  - 前頁の内容までが成功していれば、以下のコマンドを実施

入力	出力
git config user.name	（設定したユーザー名）
git config user.email	（設定したメールアドレス）

  - GitとGitHubアカウントをSSHキーで連携させるため、SSHキーを生成するが、既にSSHキーがあるかを確かめるため、以下コマンドを実施

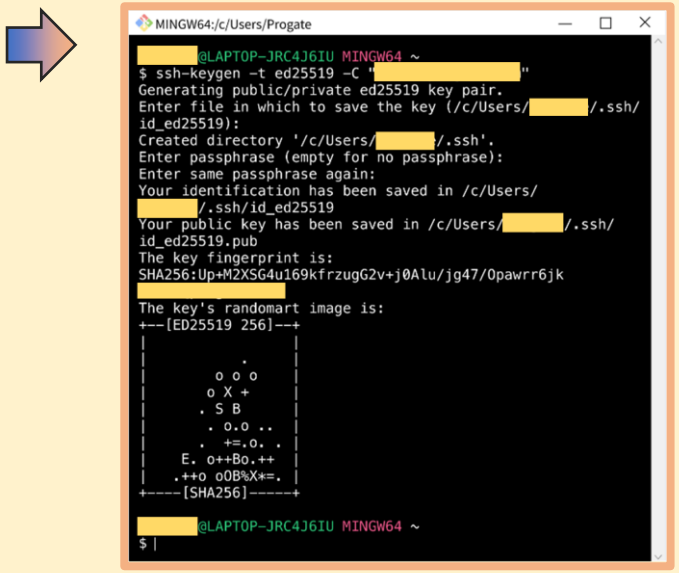
入力	出力
ls ~/.ssh	id_ed25519　id_ed25519.pub
	ls: cannot access '/c/Users/dir/.ssh': No such file or directiory

以上の下側の出力場合は、次の手順を、表示される場合は、本資料9頁に遷移

環境構築（Windows）

- Git・GitHubのインストールと基本設定
  - SSHで接続するために必要なSSHキーの作成

    ssh-keygen -t ed25519 -C “(GitHubに登録したメールアドレス)”



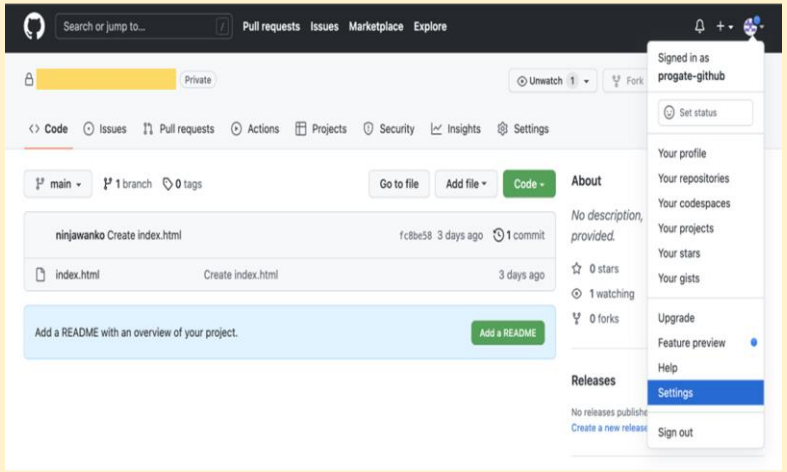
  - SSHキーが追加されたかを確認（本資料7項で実行したコマンドの実行により、上側の出力があれば成功）

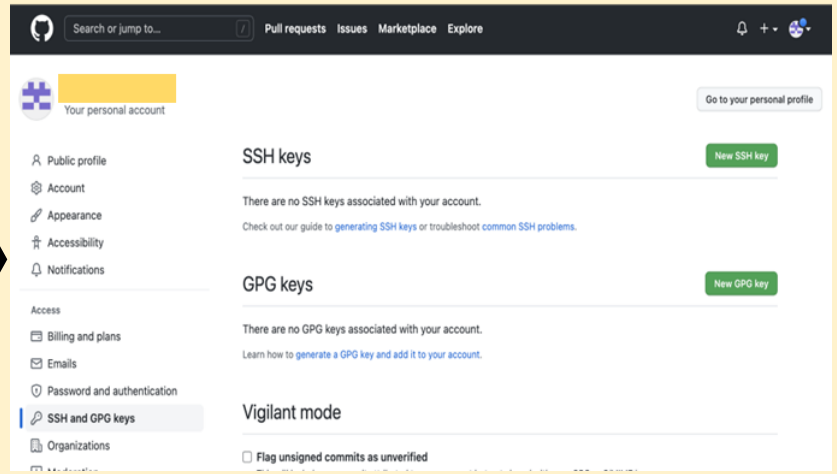
環境構築（Windows）

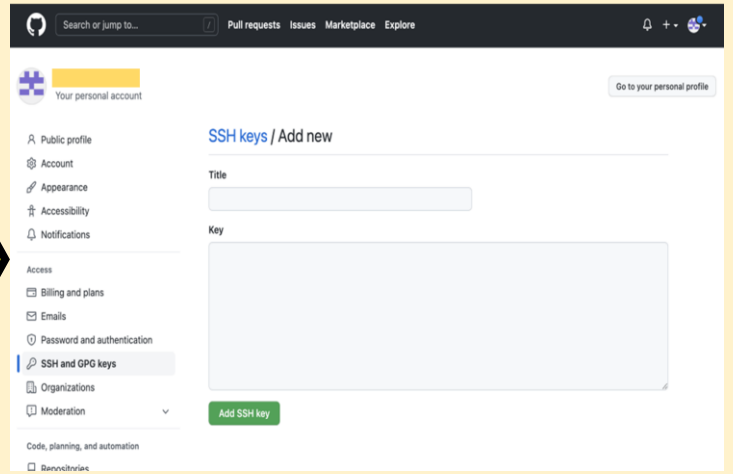
- Git・GitHubのインストールと基本設定
  - 以下のコマンドにより、GitHubアカウントへSSHキーを追加

    `clip < ~/.ssh/id_ed25519.pub` ⇒ （クリップボードにキー内容をコピー）

  - 以下のGitHubの画面より、Settingページに遷移し、「New SSH Key」を押下、押下後は「Title」に任意のタイトルを記入し、「Key」にコピー内容を添付







環境構築（Windows）

- コードの実行方法

  - タブバーの検索欄に「コマンドプロンプト」と入力し、コマンドプロンプトを起動、起動後は、カレントディレクトリを任意の場所に移動し、その場所で以下コマンドを実行

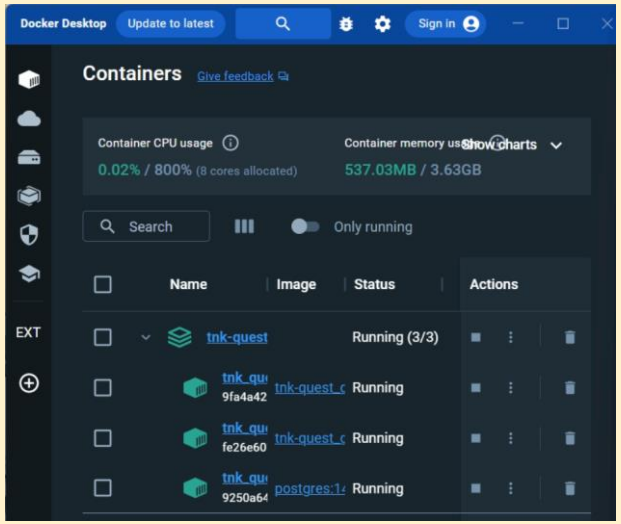
    `git clone git@github.com:ebaoacho/tnk-quest_development_js.git`

    ⇒ （カレントディレクトリにディレクトリのコピーが出現）

  - カレントディレクトリを出現したディレクトリに移動し、以下のコマンドを実行

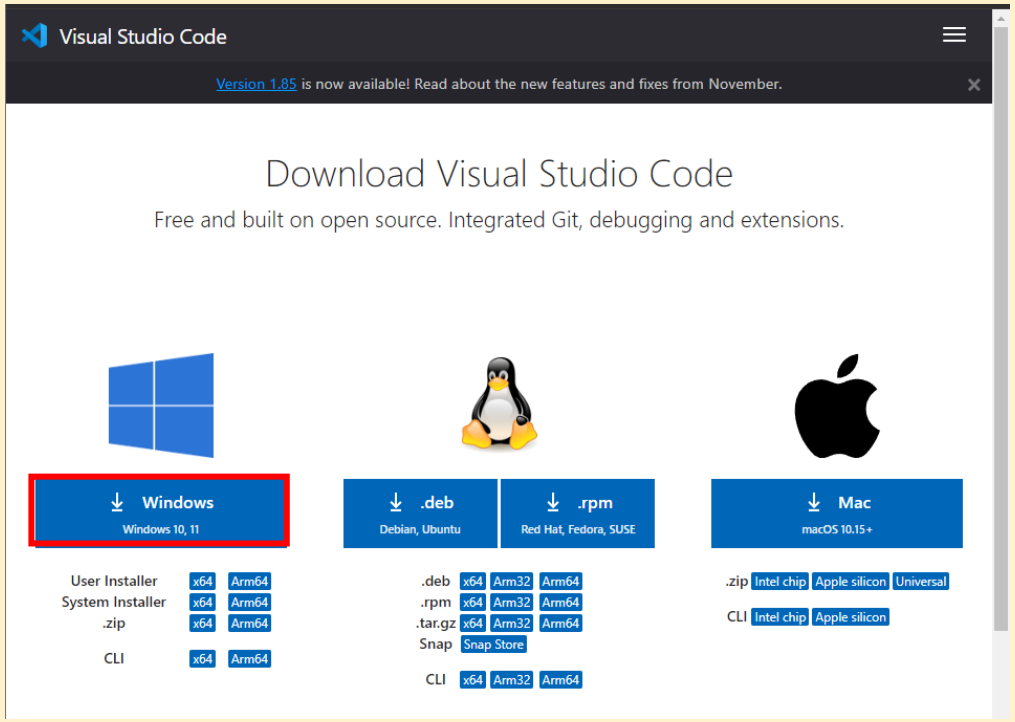
    `docker compose up -d` ⇒ 非常に長いビルドログの発生

    Docker Desktopで右図のように表示されれば成功 →



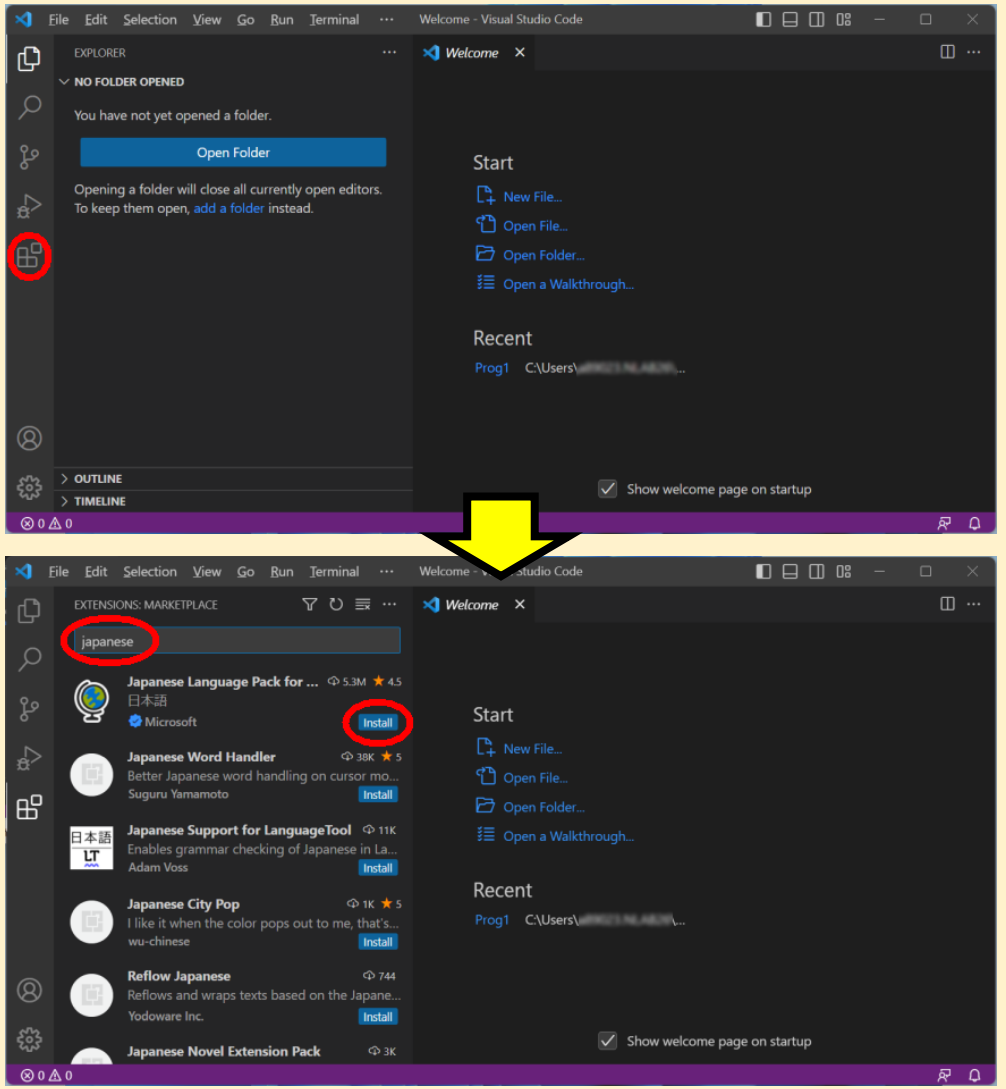
環境構築（Windows）

- コードの実行方法
  - Visual Studio Code をダウンロードページからダウンロード（既にある人は本資料13頁へ）
  - 右図のインストーラをダウンロードし、インストーラを実行
  - インストーラ実行後は、設定項目の選択が必要になるが、全てデフォルトの設定のままで「次へ」または「完了」でも正常に動作が可能



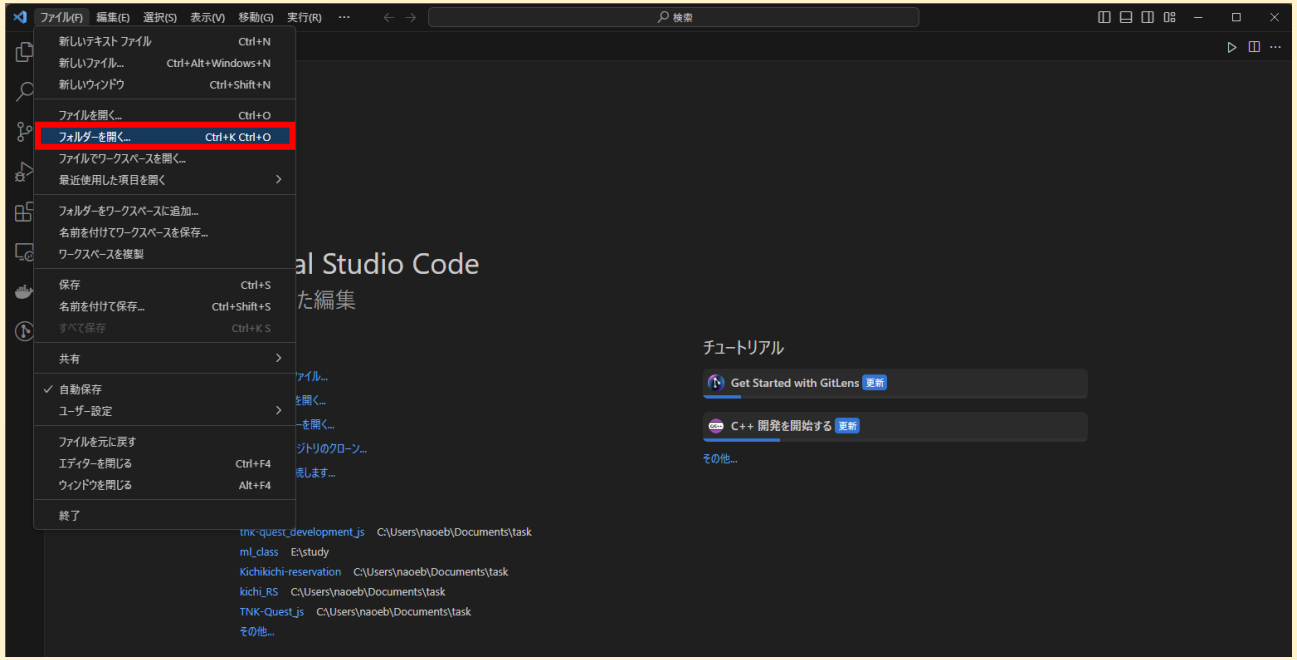
環境構築（Windows）

- コードの実行方法
  - 右図の手順により、Visual Studio Code を日本語に設定
  - 設定後はVisual Studio Code を再起動



環境構築（Windows）

- コードの実行方法
  - 下図の赤枠部分のクリックにより、フォルダーを選択
  - 「tnk-quest_development_js」フォルダの選択後、コードの閲覧が可能になり、F5キーによりコードの実行され、Webアプリが新規ウィンドウとして出力



環境構築（Windows）

- コードの実行方法
  - バックエンド内を操作するため、以下のコマンドを入力

    `docker exec -it tnk_quest_back /bin/sh` → （複数行の文字列）
  - バックエンド内で、アプリとデータベースを接続するため、以下のコマンドを入力

    `npx sequelize-cli db:migrate` → （複数行の文字列）
  - コードの改変後、F5キーを押下しても見た目や動作が変化しないため、コードの改変を環境に認識させるために、以下のコマンドにより、Docker環境を再起動

    `docker restart tnk_quest_front` → （出力無し）

各ファイルの説明

各ファイルの説明

- フロントエンド
  - 「tnk_quest_front/src/components/AccountManagement.jsx」
    ロ　アカウント管理画面を作成するためのコード
  - 「tnk_quest_front/src/components/QuestSelection.jsx」
    ロ　クエストの依頼・閲覧が実施可能であるメイン画面を作成するためのコード
  - 「tnk_quest_front/src/components/fonts」
    ロ　フォントに関するファイルが格納されているディレクトリ
  - 「tnk_quest_front/src/components/image」
    ロ　画像に関するファイルが格納されているディレクトリ

各ファイルの説明

- フロントエンド
  - 「tnk_quest_front/src/index.css」
    ロ Webアプリ全体の色やデザイン情報が含まれるcssファイル
  - 「tnk_quest_front/src/index.js」
    ロ Webアプリのコンポーネントコードを管理するjsファイル（改変不要）
  - 「tnk_quest_front/src/TnkQuestApp.js」
    ロ Webアプリのコンポーネントコードを直接処理するファイル
  - その他のファイルに関しては理解不要

各ファイルの説明

- バックエンド
  - 「tnk_quest_back/rootes」
    ロ　バックエンドの機能を実現するためのコードが含まれるディレクトリ
  - 「tnk_quest_back/models」
    ロ　データベース内のテーブルを定義するコード
  - 「tnk_quest_back/migrations」
    ロ　データベースのテーブル定義を、データベースへ反映させるコード
  - その他のファイルに関しては理解不要

コード

コード

- React.jsとは
  - WebサイトやWebアプリのUI部分を開発する際に使用するJavaScriptライブラリ
  - ダイナミックなUIや高度なユーザーインタラクションの構築に焦点を当てており、基本的なロジックは、使い回しのできるコンポーネントに基づくもので、同じコードを何度も書く手間を削減可能
  - 本Webアプリのフロントエンドの開発では、React.jsのみを使用

コード

- Express.jsとは
  - Webアプリのバックエンドを開発する際に使用するフレームワークであり、Node.js がインストールされている環境で、JavaScriptで動作
  - ユーザーが使い慣れている Node.js の機能をわかりやすくし、基礎的な Web アプリケーション機能をシンプルな階層で提供
  - 本Webアプリのバックエンドの開発では、Node.jsとExpress.jsを使用

コード

- 読み方のコツ（フロントエンド）
  - 右図は、状態変数とその変数を改変する関数の定義をペアで行う部分

```
const [isCreateAccountOpen, setCreateAccountOpen] = useState(false);
const [isLoginOpen, setLoginOpen] = useState(false);
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
const [confirmPassword, setConfirmPassword] = useState('');
const [passwordMismatchError, setPasswordMismatchError] = useState(false);
const [signupErrorDialogOpen, setSinupErrorDialogOpen] = useState(false);
const [loginErrorDialogOpen, setLoginErrorDialogOpen] = useState(false);
```

  - どのオブジェクトでreturnより手前の関数が機能しているかを理解するため、コンポーネント（AccountManagement.jsxファイルなど）を操作する場合、「return」以降を先に理解
  - コード中に登場するURL指定のような部分は、バックエンドのコードと紐づいている箇所であり、その箇所の動作への完璧な理解は不要（バックエンドのコードへの理解が必要であるため）

コード

- 読み方のコツ（バックエンド）
  - 機能的な理解は「tnk_quest_back/rootes」を読み解くことで網羅が可能
  - それ以外の理解は定義などの部分であるため、バックエンド運用メンバーの最低基準ではなく、不要
  - よって、バックエンド運用メンバーは、「tnk_quest_back/rootes」への理解のみが必要
  - コード中の「try」以降が関数の機能の部分であり、それ以外のコードは、理解不要