はじめに

ここでは Web アプリの UI として React アプリを作り、AP サーバへ配備する手順を纏めます。React アプリの開発を行う際は HTML と JavaScript の知識が少し必要になりますので、参考サイト（例えば、下記 URL）等を利用してください。

https://reactjs.org/tutorial/tutorial.html

https://ja.reactjs.org/docs/getting-started.html　（日本語）

1. React アプリの作成

以下の内容は Windows 上での開発を前提にしています。

1.1. 必要なツール／ライブラリ

（1） Node.js

Node.js はブラウザを必要としない JavaScript の実行環境です。

インストーラと zip 形式の資材があり、違いとしてはインストーラは PATH 環境変数への設定を自動でやってくれるのと、「コントロールパネル／プログラム」からアンインストールができるといった程度です。下記 URL よりダウンロード（特に事情がなければ最新版）

https://nodejs.org/ja/download/

【zip 形式を使う場合】

解凍し、環境変数の PATH に解凍先を指定する。

例として、C:¥Tools フォルダに nodejs として解凍した場合は“C:¥Tools¥nodejs”を PATH に追加してください。コマンドプロンプトより which npm と打ってインストール先のフォルダが表示されれば OK です。

（2） その他

●任意

・yarn：パッケージ・マネジャー

Node.js には npm というパッケージ・マネジャーが入っていますが、yarn を使って例示しているサイトが多数あります。yarn の方が後から作られたので高速・簡易に使えると過去には評価されていたようですが、今は npm も大差ないようです。入れるなら下記コマンド。

```
npm install --global yarn
```

●注意：インストール非推奨

・create-react-app：React アプリのセットアップツール

npm install -g[lobal] create-react-app とインストールを前提にしているサイトがありますが、今の推奨は「npx create-react-app *アプリ名*」という方法でセットアップを行う方法です。

npx は npm と一緒にインストールされるコマンドで、最新のパッケージ(create-react-app)をインストールして実行します。

※この後、アプリで使うライブラリやツール（のパッケージ）をインストールする。主なインストール・オプションは以下のとおり。

- **-g/--global** グローバル、Node.js に 1 つ共用としてインストール（指定無：アプリ固有）
- **-D/--save-dev** 開発（デバッグ）環境のみで使用（運用[ビルド]時は組み込まない）

1.2. アプリの初期生成

以下の手順／コマンドで React アプリのテンプレートと、必須ライブラリが入ったアプリ名のフォルダ（プロジェクトと呼んでいる場合もある）が作成されます。

（１） 手順

① フォルダを作成する親フォルダに移動する　cd *親フォルダ*

② Create React App の実行　npx create-react-app *アプリ名*

--以上で動作可能な資材が揃います／以下、アプリの実行--

③ アプリ名のフォルダに移動する　cd *アプリ名*

④ 実行（デフォルトブラウザが自動起動）　npm start

（２） 生成時に出る警告・エラー

手順②でエラーや警告がでます。以下が対処方法です。

・Git commit not created Error: Command failed: **git を使ってないなら問題ありません（無視）**

・[3/4] Linking dependencies... **：この後に依存性に関する問題個所が...（解決策は下記*1）**

warning "react-scripts > @typescript-eslint/eslint-plugin > tsutils@3.20.0" has unmet peer dependency "typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev || >= 3.7.0-beta".

npm WARN optional SKIPPING OPTIONAL DEPENDENCY

*1 依存性に関する解決手段（）

①npm install を再実行

*アプリ名*フォルダに package.json という依存ライブラリ／バージョンを書いたファイルがある。このコマンドはファイルの内容からライブラリの再インストールを試みる

②npm install *typescript@>=2.8.0*

ラインマーカーの先にあるライブラリを手動でインストールする。但し、インストールした先で更に依存性不足が発生する場合があります。因みに、typescript-eslint は TypeScript 用のコードチェッカー（のようなもの）なのでインストールしてなくてもアプリは動作します。

③無視…“OPTIONAL DEPENDENCY”は OS 別等環境固有のライブラリなので無視してよい

※基本、warning や WARN は無視しても動作します。
気になったら本当に必要なパッケージか、パッケージ名から内容を確認してください。

create-react-app の初期はインストールして使う説明になっていましたが、バージョンアップと共に npx コマンド経由で使うことを推奨に変わり、使い方が重なるとエラーが出ます。

こんなのがでたら...下の下線を引いたコマンドをたたく

If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app to ensure that npx always uses the latest version.

npm uninstall -g create-react-app　or　yarn global remove create-react-app

2. 初期生成資材と開発（肉付け）

2.1. 初期生成物（テンプレート）

create-react-app コマンドで*アプリ名*フォルダ（下記例では”new-app”）配下に複数のフォルダ／ファイルが作られます（版数が上がると出力される内容も若干変わる）。

【新たに作られるフォルダ／ファイル】

```
C:¥REPOS¥REACT¥NEW-APP
|   .gitignore
|   2.8.0
|   package-lock.json
|   package.json
|   README.md
|
├──node_modules
|   (略)
|
├──public
|       favicon.ico
|       index.html
|       logo192.png
|       logo512.png
|       manifest.json
|       robots.txt
|
└──src
        App.css
        App.js
        App.test.js
        index.css
        index.js
        logo.svg
        reportWebVitals.js
        setupTests.js
ー 以上 ー
```

＜各フォルダのサイズ＞

```
$ du -h --max-depth=1 new-app

287M    new-app/node_modules
34K     new-app/public
15K     new-app/src
287M    new-app
```

アプリケーションの起動は src フォルダ内の index.js から始まり[1]ます。

[1] 開発サーバ(npm start)の場合（運用サーバ配備後〔npm run build で作られる資材〕では index.html + JavaScript に変換されている）

2.2. 開発（基礎知識）

アプリケーションは前項の初期生成物（テンプレート）を基に作ります。

（以下、React 17.0.1 で作成）

（１） 初期生成物の構成

App.jsの最後でexportしている“App”をindex.jsでimport（”./App”:同一フォルダより探す）し、htmlタグのような“<App />”(JSX記法)で“function App()”を呼び出しています。

呼び出す単位（この場合はApp）が**コンポーネント**で、コンポーネントは階層を持てます。

【index.js】

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

【App.js】

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

（２） コンポーネントの名前

コンポーネントの名前は大文字で始めます（例；App)。小文字から始まるとReactはhtmlタグとして取り扱います。htmlタグと混在させることができ、htmlタグもJSXのコンパイルを経由して出力されます。

（３） コンポーネントの構成と描画

コンポーネントは関数として定義し、UI（ブラウザに描画する内容）をreturnします。
データの加工処理は以下の場所に書くことができます。

① return句の中JSX記法やhtmlタグと混ぜて波括弧”{}”で囲んで記述する
② returnの前に関数を書いておき、return句(JSX記法)の中から参照する
③ returnの前に関数を書いておき、描画(render)サイクルに合わせてReactに呼んでもらう

<例>

・useMemo()　const kariGoukei = useMemo(処理，トリガー)
・useEffect()　useEffect(処理，トリガー)

…描画(render) はブラウザに対するキー入力を契機にReactにより2度実行され、useMemoは1回目、useEffectは1回目と2回目の間に実行される。
キー入力 ⇒ useMemo(render中に実行) ⇒ (render後)useEffect ⇒ 再render

※キー入力はキーボードを1押下毎に発生する。例えば、5桁の入力に対しては5回のrenderとデータ更新を契機とする処理行われる…validateの契機をonChangeにした場合、開発サーバでは引っかかる感じになるがbuild後は気にならない程度に改善する
※useMemoは入力中にリアルタイムで集計する等に、useEffectは入力値でDBを参照し、結果を使って画面を更新する等に使うことができる

（４） 描画／入力に使うデータ(state)の宣言

コンポーネントは次のライフサイクルを経過します。

① 初期データとhtmlタグを合成した結果を作り出し(return)描画する
② 操作者により①に対してデータの入力やクリック等（イベント）の操作が行われる
③ ②操作を受けたDBアクセスや、時刻等をトリガーとしてコンポーネント内でデータを更新し、再描画する

上記②のデータ入力を契機としたDBアクセスや③のデータ更新による再描画を行うためには当該のデータを監視し、制御する必要があります。この監視・制御対象のデータをステート(state)、stateを扱うコンポーネントをステートフル・コンポーネント(stateful component)と呼びます。

本資料で想定してるstateの宣言方法は2種類あります。

① React本来の方法：変数毎にuseState()を使って宣言
② htmlのformタグに類似した機能を持つreact-hook-formライブラリのuseForm()を使って複数の変数を纏めて宣言

（５） react-hook-form を利用した state 管理

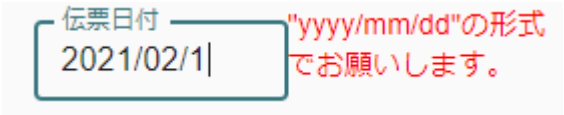
Webアプリはデータ入力にhtmlのformタグを使いますが、React自体にはformのような（複数の項目を纏めて送信する）機能がないのでreact-hook-formを使います。このライブラリは以下の機能も併せて持っています。

① stateの初期値／エラー状態のセットとリセット
② 項目チェック(validation) …必須(required)、形式(pattern)等のチェックができます
③ エラー内容の保存 …errorsオブジェクトにエラーメッセージが保存され利用できます

以下のように記述することで、stateに対する入力中にエラーメッセージの表示が行えるようになります。useForm()で宣言したstateは“register”で画面の項目(html)と関係づけます。

【react-hook-form の validation】

```
<CssTextField
  label='伝票日付'
  type='text'
  name='trDate'
  inputRef={register({
    required: '取引日を入れてください。',
    pattern: {
      value: /^¥d{4}¥/¥d{2}¥/¥d{2}$/, message: '"yyyy/mm/dd"の形式でお願いします。'
    }
  })}
/>
{errors.trDate && (
  <span className={classes.error}>{errors.trDate.message}</span>
)}
```



ReactのuseState()で宣言したstateとreact-hook-formのuseForm()で宣言したstateは別物で、相互作用／干渉しません。画面の表示や制御のみに使う場合はuseState()、ServletへのhttpRequest(JSON)として扱うときはuseForm()を使用します。

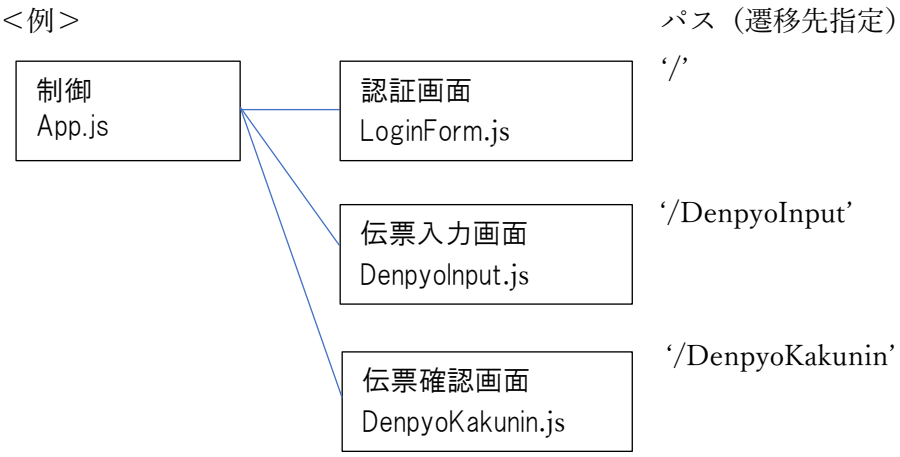
インストール（ローカル）：npm install react-hook-form
使い方の詳細：https://react-hook-form.com/jp/get-started#Quickstart
サンプル集：https://github.com/react-hook-form/react-hook-form/tree/master/examples
（サイトcodesandbox.ioへのリンク先で実際に動作が確認できます）

（６） 画面遷移

Reactは画面の内容を書き換えられますが、画面全体を差し替えるような機能はありません。Webアプリで行う画面遷移はreact-router-domを使い実現します。

react-router-domは、アプリを構成する「１コンポーネント／１画面」と画面の切替を制御するコンポートで画面遷移を実現します。

＜例＞



react-router-domを使って画面遷移行うにあたり、以下の機能を組込む必要があります。

① 認証が通っていない場合（セッション切れ含む）は認証画面に戻す

② 画面間のデータ引継ぎ

---実装例は次ページ---

インストール（ローカル）：npm install react-router-dom

使い方の詳細：https://reactrouter.com/web/guides/quick-start

【App.js …react-router-dom 組込み後】

```
import React from 'react';
import {BrowserRouter as Router ,Switch ,Route ,Link ,Redirect ,useParams} from 'react-router-dom';
import { LoginForm, loginStatus } from './pages/LoginForm';
import DenpyoInput from './pages/DenpyoInput';
import DenpyoKakunin   from './pages/DenpyoKakunin';
export const AppContext = React.createContext();
export const routePath = {curPage : 'Login', prevPage : ''}
//子画面間では直接データの受け渡しができないため、両方の親の App.js に受け渡しデータを定義する。
const DenpyoInputStat = {formData : {}, refData : {}}
function App() {
  //引数オブジェクトの名前"component"を"Component"に付け替え
  const JudgeRoute = ({ component: Component, ...otherParas }) => {
    {console.log('in APP',DenpyoInputStat)}
    return (
      <Route {...otherParas} render={props => (loginStatus.isLogin? //…ログイン済？
          <Component {...props} />
        : <Redirect to='/'/>
      )}/>
    )
  }
  return (
    <AppContext.Provider value={DenpyoInputStat}>
    <div className='App'>
      <Router>
        <div>
          <ul>
            <li><Link to='/'>Login</Link></li>
            <li><Link to='/DenpyoInput'>伝票入力</Link></li>
          </ul>
          <hr/>
          <Route render={() => loginStatus.isLogin? `${loginStatus.Uname} (${loginStatus.Yakucd})`:''} />
          <Switch>
            <Route exact path='/' component={LoginForm}/>
            <JudgeRoute exact path='/DenpyoInput' component={DenpyoInput}/>
            <JudgeRoute exact path='/DenpyoKakunin' component={DenpyoKakunin}/>
          </Switch>
        </div>
      </Router>
    </div>
    </AppContext.Provider>
  );
}
export default App;
```

●App の認証判定[*1] は JudgeRoute コンポーネントで行い、NG ならルート(“/”)にリダイレクト

●子（画面）と親（App）だけのデータ共有(routePath)は親が export、子が import で可能

●子の間で共有が必要な場合は Context(AppContext)にデータ(DenpyoInputStat)を包み export

*1：App はアプリが起動した初期表示と、react-router-dom が表示するメニュー（デフォルトで表示される）から呼ばれる。伝票入力画面画面画面等でサーバとの通信時にセッション切れを検知した場合は、以下のようにリダイレクトする（App に制御を戻し、App で判定する方式も取り得る）。

【App.js 以外で通信時にセッション切れを検知した場合】

```
//伝票照会
const denNo = watch('denNo');
const inqDenpyo = () => {
  if (errors.denNo || denNo==='') return;
  console.log('AccDenpyoInquiry onClick', denNo);
  fetch ('http://localhost:8080/React/AccDenpyoInquiry',
  {
    method: 'POST'
   ,credentials: 'include'
   ,headers: {'Content-Type': 'application/json;charset=utf-8'}
   ,body: JSON.stringify({'denNo': denNo})
  })
  .then(response => {
    if (!response.ok) {
      throw new Error('DenpyoInquiry との通信に失敗しました。');
    }
    return response.json();
  })
  .then(data => {
    console.log('AccDenpyoInqury response', data);
    switch (data.rtncd) {
      case '000' : //正常リターン

        《中略》

        reset(data);
        break;
      case '101' : //未認証
        props.history.push('/');
        break;
      default :
    }
  })
  .catch(error => {
    alert('例外発生: '+error);
    console.error('例外発生: ', error);
  });
}
```

※サーブレットはリクエスト毎にセッションの有効性を確認しリターンコードを設定している

（７） 画面デザイン

Reactはstateを管理しJSXをコンパイルしてhtmlとJavaScriptモジュールを作り出しますが、装飾に関してはスタイルシート(css)を自前で用意する必要があります。Material-UIを組み込むと入出力項目やボタン等の見た目を装飾し、画面全体が統一感のあるものにできます。

Material-UIはReactのコンポーネントライブラリで、react-hook-formと連動するインタフェースを持っています。

【Material-UI のコンポーネント例】

```
//<input type="text"> を生成する
<CssTextField
    label='伝票番号'
    type='text'
    name='denNo'
    variant='outlined'
    margin='dense'
    autoComplete=''
    className={classes.margin}
    inputRef={register({
      pattern: {
        value: /^¥d{5}$/, message: '伝票№は 5 桁です。'
      }
    })}
/>

//<input type="checkbox"> を生成する
<Checkbox
  className={classes.checkBox}
  inputRef={register()}
/>
```

インストール（ローカル）：npm install @material-ui/core

使い方の詳細：https://material-ui.com/ja/getting-started/installation/

Material-UI を使ったテーマ・サンプル

https://material-ui.com/getting-started/templates/

認証画面で参考にしたサイト（React-Hook-Form の DevTools に関する記述もあります）

https://dev.to/sergiosanchezs/ultimate-form-validation-in-react-with-the-awesome-react-hook-form-and-material-ui-libraries-26ii

2.3. React 開発サーバと AP サーバの接続

（1） AP サーバ側の設定

npm start で React の開発サーバが local ホストのポート 3000(http://localhost:3000)で起動しますが、AP サーバ側は React 開発サーバからの通信(fetch/POST,GET 等)を拒否します。

この仕組みは『オリジン間リソース共有（CORS：Cross-Origin Resource Sharing)』と呼ばれるもので、通信を許可するための設定が必要です。

参照先: https://developer.mozilla.org/ja/docs/Web/HTTP/CORS

＜Tomcat v8 の場合＞通信先 AP の web.xml に以下の<filter>～</filter>、<filter-mapping>～</filter-mapping>を追加する

【web.xml ：CORS の許可設定例】

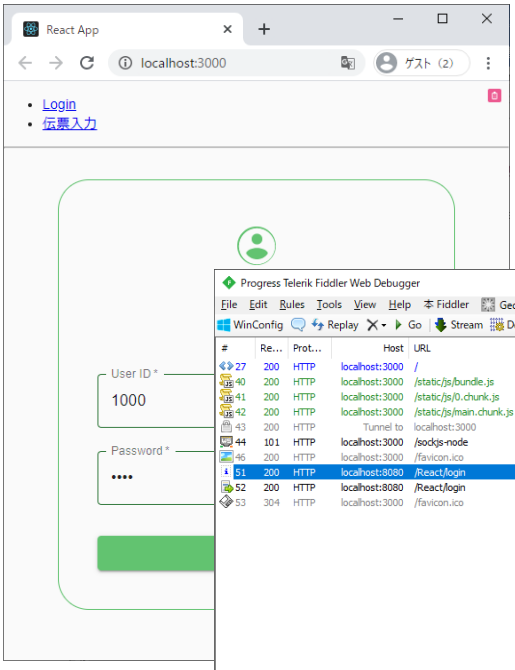
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
        （略 <servlet></servlet><servlet-mapping></servlet-mapping> 略）
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
  <init-param>
    <param-name>cors.allowed.origins</param-name>
    <param-value>http://localhost:3000</param-value> <!-- [*]だと cors.support.credentials=[true]がエラーになる
-->
  </init-param>
  <init-param>
    <param-name>cors.allowed.methods</param-name>
    <param-value>GET,POST,HEAD,OPTIONS,PUT</param-value>
  </init-param>
  <init-param>
    <param-name>cors.allowed.headers</param-name>
    <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-Method,Access-
Control-Request-Headers</param-value>
  </init-param>
  <init-param>
    <param-name>cors.exposed.headers</param-name>
    <param-value>Access-Control-Allow-Origin,Access-Control-Allow-Credentials</param-value>
  </init-param>
  <init-param>
    <param-name>cors.support.credentials</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>cors.preflight.maxage</param-name>
    <param-value>10</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>
```

（2） fetch で起こること

React アプリが AP サーバに向けて fetch/POST を投げると、2 回通信が発生します。

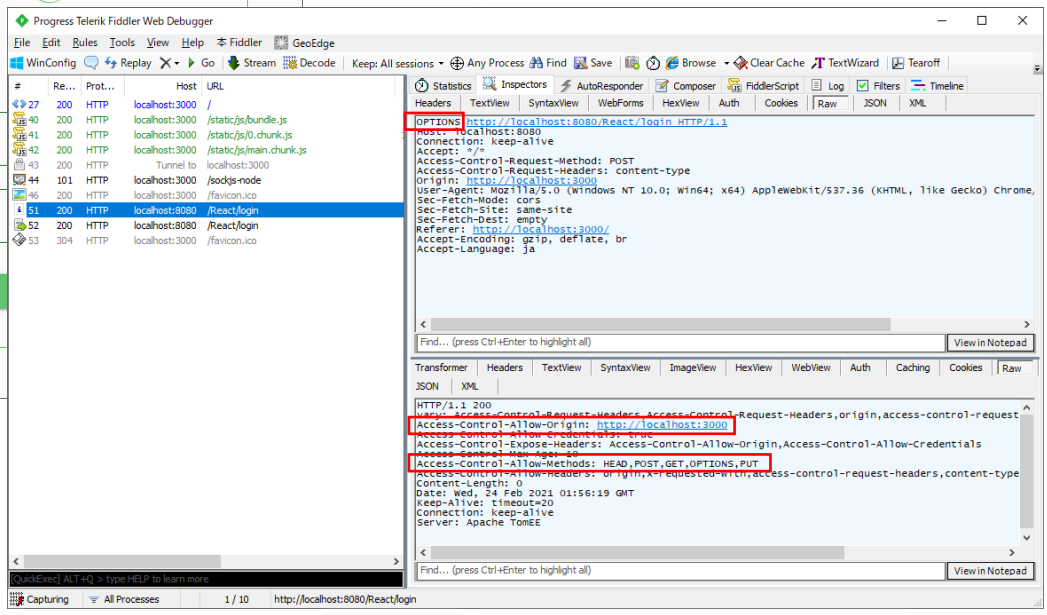
1 回目：OPTIONS　…AP サーバ側の受け入れ確認

2 回目：POST　　…ここでデータが送信される

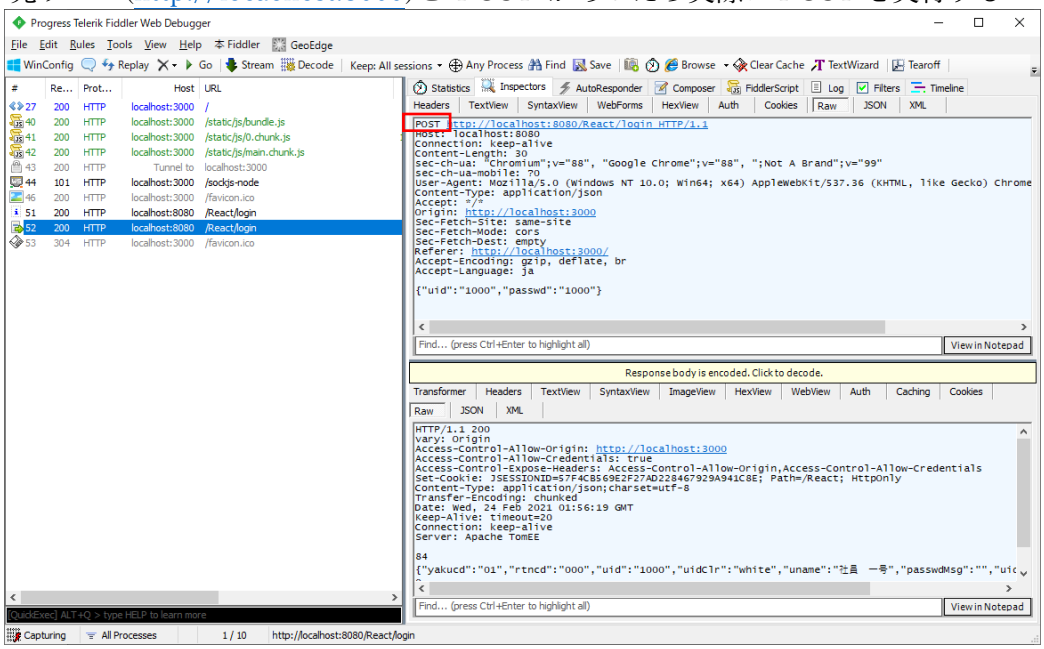


ログインボタン押下

① OPTIONS：CORS 確認



② ①のレスポンス・ヘッダーの「許可オリジン」「許可メソッド」を確認し、開発サーバ(http://locaohost:3000)と”POST”があったら実際に POST を実行する



3. Reactのサーバ配備

3.1. ビルド

開発サーバでの作業が終わったら、ビルドコマンドを使いリリース用の資材を作ります。

（1） 相対パスの構築＃

デフォルトでは、Create React Appは、アプリがサーバールートでホストされていると想定してビルドを生成します。サーブレットと同一サーバに搭載するためには以下のようにします。

詳細＞https://create-react-app.dev/docs/deployment/#step-1-add-homepage-to-packagejson

・プロジェクトフォルダにある package.json に, "homepage": "." を追加する
（他のパラメータとの間に「,」を挟むことに注意してください…JSONフォーマット）。

※package.jsonは cp -a package.json package.json.origin のようにバックアップを作成してから修正してください

```
{
  "name": "acc",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@material-ui/core": "^4.11.0",
    "@material-ui/icons": "^4.9.1",
    "@testing-library/jest-dom": "^5.11.6",
    "@testing-library/react": "^11.2.2",
    "@testing-library/user-event": "^12.2.2",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-hook-form": "^6.11.5",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.0",
    "react-table": "^7.6.2",
    "web-vitals": "^0.2.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@hookform/devtools": "^2.2.1"
  }
, "homepage": "."
}
(END)
```

（２） React の通信(fetch)先

Reactからサーブレットはurlで特定しますが、サーブレットと同一のWebサーバに配置する場合はドメインとポートが同一になるため省略することが可能です。

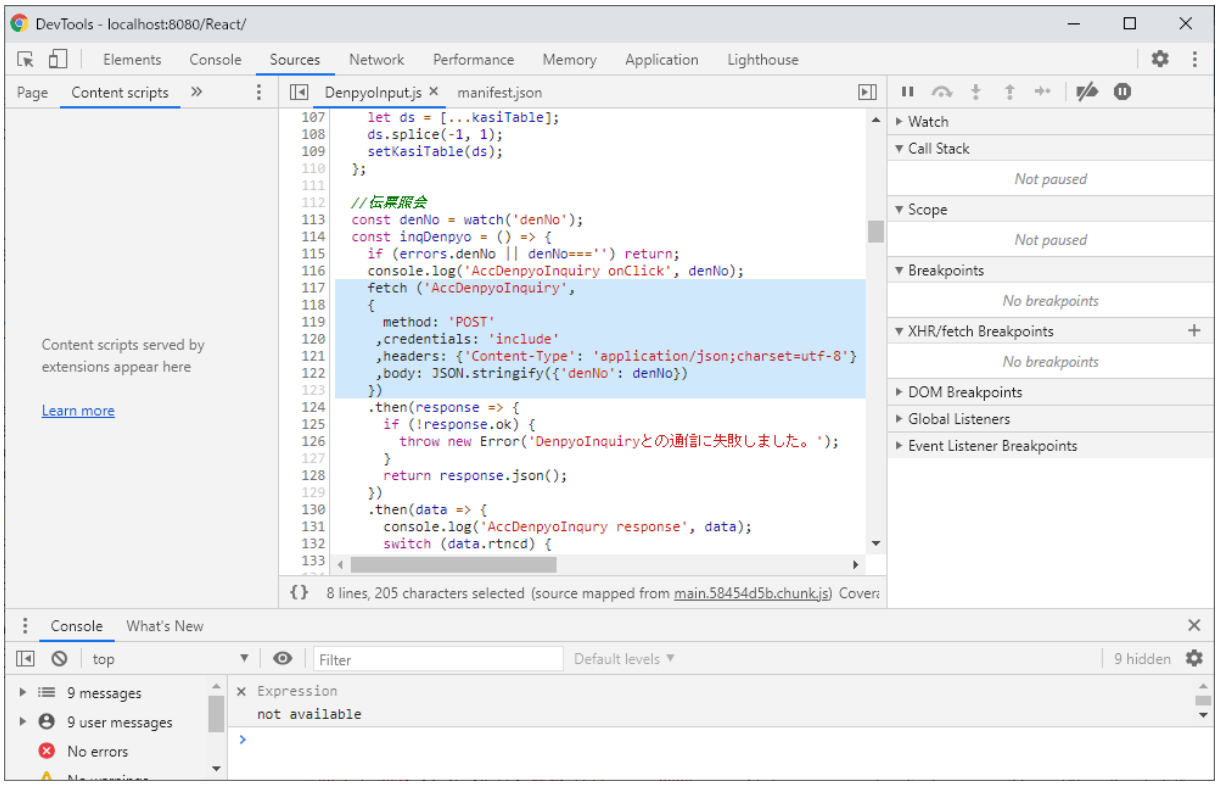
```
fetch ('http://localhost:8080/React/AccDenpyoInquiry',
{
   method: 'POST'
  ,credentials: 'include'
  ,headers: {'Content-Type': 'application/json;charset=utf-8'}
  ,body: JSON.stringify({'denNo': denNo})
})
```

↓

```
fetch('AccDenpyoInquiry',
{
   method: 'POST'
       (以下略)
```

※サーバ配備後にChromeから試したところ、下記のパターンでも動作しました
（ブラウザ、APサーバの差異等、常に動作するかは不明です）
”/React/AccDenpyoInquiry”、“React/AccDenpyoInquiry”

注意）意図したスクリプトが適用されているかは、必ずブラウザの開発ツールで確認のこと
Chromeの場合、F12 > Sourcesタブ > Ctl + P



（３） build コマンド

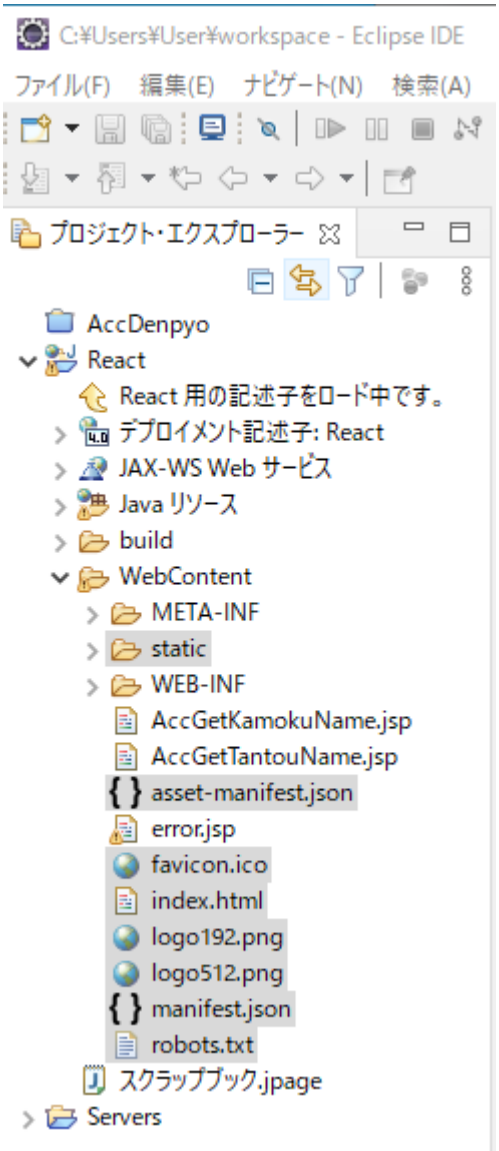
プロジェクトフォルダに移動(cd *プロジェクト・パス*)して、下記のコマンドを実行します。

```
$ npm run build
```

…「buid」フォルダが作られ、実行資材が構成されます

```
build:
total 42
drwxr-xr-x 1 User 197121    0  2月  5 11:55 ./
drwxr-xr-x 1 User 197121    0  2月  5 11:54 ../
-rw-r--r-- 1 User 197121 1031  2月  5 11:55 asset-manifest.json
-rw-r--r-- 1 User 197121 3870 11月  5 10:34 favicon.ico
-rw-r--r-- 1 User 197121 3015  2月  5 11:55 index.html
-rw-r--r-- 1 User 197121 5347 11月  5 10:34 logo192.png
-rw-r--r-- 1 User 197121 9664 11月  5 10:34 logo512.png
-rw-r--r-- 1 User 197121  492 11月  5 10:34 manifest.json
-rw-r--r-- 1 User 197121   67 11月  5 10:34 robots.txt
drwxr-xr-x 1 User 197121    0  2月  5 11:55 static/
```

3.2.　サーバへ配置（サーブレットの開発環境(Eclipse)に入れた例）



buid フォルダ配下にできた資材を WebContent 配下にコピーする。この後「**サーバで実行**」や「**サーバでデバッグ**」で React の画面が表示される。