**Demo Contact List project documentation**

DOCKER – container local hosting for all three instances

MongoDB – storing of the data in a no-sql database

.NET CORE API – uses the mongo db for storage but has methods that the react site sees and utilizes

REACT + REDUX SITE – Visual site the actual UI the user interacts with, that in the backend only talks to the .net core api instance on docker.

**Docker Commands**

<https://app.pluralsight.com/library/courses/docker-web-development/table-of-contents>

Docker pull <image name>

Docker images

Docker ps

Docker ps -a

Docker rm <container>

Docker rmi <image>

Docker stop <container>

Docker run -d -p 8080:80 <tag name>

Docker build -t <tagname> .

Docker system prune 🡪 (THIS COMMAND REMOVES ALL STOPPED CONTAINERS, ALL DANGLING IMAGES, AND ALL UNUSED NETWORKS)

Docker container stop $(docker container ls -aq) 🡪 stops all running containers

Docker container rm $(docker container ls -aq) 🡪 removes all stopped containers

docker image prune -a 🡪removes all unused images images without a container

**Docker MongoDb**

//download the mongo inside the docker container... (latest version else just add the version with mongo:3.3)

docker pull mongo

// deploying an instance of mongo on docker as a container...

docker run --name mongodb mongo:4.0.4

//rather use the below to specify the port also... -d -p detach the container to be accessable from the host pc

docker run -d -p 27017-27019:27017-27019 --name mongodb mongo:4.0.4

//start the mongodb thats hosted in docker... and be able to use the mongoDB shell client

docker exec -it mongodb bash

To launch the MongoDB shell client, execute the following:

mongo

show dbs

use <dbs you want to use>

We’re using the database <dbs you want to use>, but it doesn’t exist until we start creating collections and data. To create a collection with data, we can do something like this:

db.people.save({ firstname: "Nic", lastname: "Raboy" })

db.people.save({ firstname: "Maria", lastname: "Raboy" })

With two documents created in a new people collection in our thepolyglotdeveloper database, we can query for data using something like the following:

db.people.find({ firstname: "Nic" })

There is a lot that you can accomplish with the shell client, but you can get the general idea. What we did prove is that we were able to interact with the container instance.

**Use Robo software to access the mongo db from the host to make sure its accessible from host to docker container.**

**.NET CORE API**

Starting a normal .net core api application on visual studio. Various settings are required such as.

* CORS additional settings that’s missed usually in the Startup.cs file
  + In the ConfigureServices(IServiceCollection services) method before the services.AddMvc() …. method
* services.AddCors(options =>
* {
* options.AddPolicy("CorsSetup",
* builder =>
* {
* builder.AllowAnyOrigin()
* .AllowAnyHeader()
* .AllowAnyMethod();
* });
* });
  + In the Configure method add “ add.UseCors(“CorsSetup”); “ before useSwagger and useMvc method calls… note the in “” same name as the added policy in the above mentioned services.addcors
* Remember to utilize or implement swagger in the startup.cs class
  + In the ConfigureServices method add
* // Register the Swagger generator, defining 1 or more Swagger documents
* services.AddSwaggerGen(c =>
* {
* c.SwaggerDoc("v1", new Info { Title = "Martin's API", Version = "v1" });
* });
  + In the Configure method add the following before the app.UseMvc() method call
* // Enable middleware to serve generated Swagger as a JSON endpoint.
* app.UseSwagger();
* // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
* // specifying the Swagger JSON endpoint.
* app.UseSwaggerUI(c =>
* {
* c.SwaggerEndpoint("/swagger/v1/swagger.json", "My Martin V2");
* });
* In your appsettings.json file add the mongodb connection settings as follows:

"ContactInfoDataBaseSettings": {

"ContactInfoCollectionName": "test",

"ConnectionString": "mongodb://mongodb:27017/test",

"databaseName": "test"

},

* Adding the Database configuration in the startup.cs file in the COnfigureServices method as follow:
* services.Configure<ContactInfoDataBaseSettings>(
* Configuration.GetSection(nameof(ContactInfoDataBaseSettings)));
* services.AddSingleton<IContactInfoDataBaseSettings>(sp =>
* sp.GetRequiredService<IOptions<ContactInfoDataBaseSettings>>().Value);
* General structure used
  + Controller
    - ValuesController.cs 🡪 HTTP Get post put delete and get(id) method declarations.
  + Models
    - ContactInfo.cs 🡪 actual info object that is a model for the db to be stored
    - ContactInfoDataBaseSettings.cs 🡪 Database settings getters and setters in order to get the settings out of the appsettings.json file
  + Services
    - ContactServices.cs 🡪 implementing the mongo db services that’s being called from the controller
  + Root folder files
    - Dockerfile 🡪 docker configuration file to create a docker image file
    - Docker-compose.yml 🡪 docker configuration file to create and run docker images and link up the mongo container and the api container
    - Appsettings.json 🡪 default settings and configuration file such as the mongodb settings
    - Program.cs 🡪 Main method of the project being build and run from
    - Startup.cs 🡪 has the ConfigurationServices, Startup and Configuration methods in that needs to be configured to use things like (swagger and mvc and mongodb settings, as well as the CORS configuration)
    - ContactListApi.csproj 🡪 the main solution project file of the .net core api
* .NET CORE API dockerizing the project
  + Dockerfile
* FROM microsoft/dotnet:sdk AS build-env
* WORKDIR /ContactListApi
* # Copy csproj and restore as distinct layers
* COPY \*.csproj ./
* RUN dotnet restore
* # Copy everything else and build
* COPY . ./
* RUN dotnet publish -c Release -o out
* # Build runtime image
* FROM microsoft/dotnet:aspnetcore-runtime
* WORKDIR /ContactListApi
* COPY --from=build-env /ContactListApi/out .
* ENTRYPOINT ["dotnet", "ContactListApi.dll"]
  + docker-compose.yml
* version: "3"
* services:
* mongodb:
* container\_name: mongodb
* image: mongo:3.2
* ports:
* - "27017:27017"
* networks:
* - default
* contactlistapi:
* container\_name: contactlistapi
* image: contactlistapi
* restart: always
* build: .
* networks:
* - default
* ports:
* - "8080:80"

**REACT + REDUX SITE**

**React** (also known as **React**.**js** or **ReactJS**) is a JavaScript library for building user interfaces. ... However, fetching data is only the beginning of what happens on a **web**page, which is why complex **React** applications usually require the use of additional libraries for state management, routing, and interaction with an API.

**Redux** is an open-source JavaScript library for managing application state. It is most commonly used with libraries such as React or Angular for building user interfaces. Similar to (and inspired by) Facebook's Flux architecture, it was created by Dan Abramov and Andrew Clark.

**Why Redux?**

React — A JS library that helps us to divide up our app into multiple components but doesn’t clearly specify how to keep track of the data(aka State) and how to deal with all the events(aka Actions) properly.

React doesn’t talk about how to manage state object and redux is the only way to fix it.

Redux — A complimentary library to React that provides a way to easily keep the data(State) and the events(Actions).Redux isolate state object from components.

RESOURCE: <https://medium.com/tkssharma/understanding-redux-react-in-easiest-way-part-1-81f3209fc0e5>

**Setting up project.**

One of two ways can be used

1. <https://github.com/coryhouse/react-slingshot> clone this repo that has the essentials already set up. And remove the demo project.
2. Create a new “npm init” project in a folder and add all the required dependencies, files and folder structure manually.

Site to use to get a great base ground understanding of how it works work through the course “Building Applications with React and Redux” - <https://app.pluralsight.com/library/courses/react-redux-react-router-es6/table-of-contents>

**Basic structure**

* Src
  + Actions
    - actionTypes.js
    - addcontactactions.js
    - contactsactions.js
  + Reducers
    - apiStatusReducer.js
    - contactsReducer.js
    - index.js
    - initialState.js
  + Api
    - apiStatusActions.js
    - apiUtils.js
    - contactListApi.js
  + Components
    - Addcontact (page)
      * AddNewContactForm.js
      * ManageContactPage.js
      * newContactModel.js
    - Common
      * Header.js
      * Spinner.css
      * Spinner.js
      * TextInput.js
    - Contacts (page)
      * ContactList.js
      * ContactListPage.js
    - Home (page)
      * HomePage.js
    - Root files
      * App.js
      * PageNotFound.js
      * Root.js
  + Constants 🡪 used for constant values
  + Store
    - configureStore.js
  + Styles 🡪 site css styling files
  + Src Root files
    - Favicon.ico
    - Index.css
    - Index.ejs
    - Index.js
    - Webpack-public-path.js
* Tools
  + srcServer.js
* Project Root files
  + Dockerfile
  + docker-compose.yml
  + packages.json
  + package-lock.json
  + webpack.config.dev.js
  + webpack.config.prod.js

**REACT + REDUX FLOW**

In code example of the flow how the code interprets the different states and actions…

<https://hackernoon.com/https-medium-com-heypb-react-redux-workflow-in-4-steps-beginner-friendly-guide-4aea9d56f5bd>

**Save image below as gif and see the flow of how it works**

