Preparing to Deploy Full Stack APP on AWS Beanstalk as a "Monorepo"

The monorepo approach uses a single repository to host all the code

Server Side:

- First, your final frontend React build pack must be located in the back end (server) side. Create a *"public"* folder on the backend and create a ".env" file on the front end and a ".env.production" file on the frontend as well.
- In that frontend (client side) .env the REACT_APP_SERVER_URL (if used) should be pointed to the correct backend local fetch location for development like this example below. The "REACT_APP_SERVER_URL" is a variable that should be used in all your frontend fetch paths, this way you don't have to change them throughout all your code prior to deploying.
- The front end should also have a .env.production file that will contain the build path redirection to the backend side and a modified "REACT_APP_SERVER_URL" fetch path for production as shown in the example below. That production env file will be automatically used when you run the "npm run build" prior to deploying.

Note that the production version of the "REACT_APP_SERVER_URL" fetch path doesn't include the http://localhost:5000/ see below example.
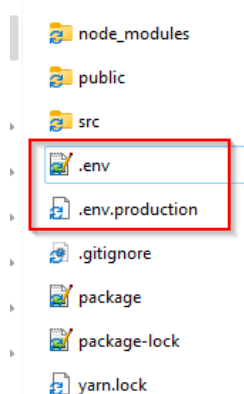
Examples:

```
.env
REACT_APP_SERVER_URL=http://localhost:5000/

.env.production
BUILD_PATH=../server/public/build
REACT_APP_SERVER_URL= /
```



```
await fetch(`${process.env.REACT_APP_SERVER_URL}users/`, {
  method: "POST",
```

- On the server express side, in the main index.js or server.js file of your application you must add this logic below. This will cause the backend build folder to be used by the browser to render the app after deploying. Remember to require or import the "path" up top along with adding the code below.
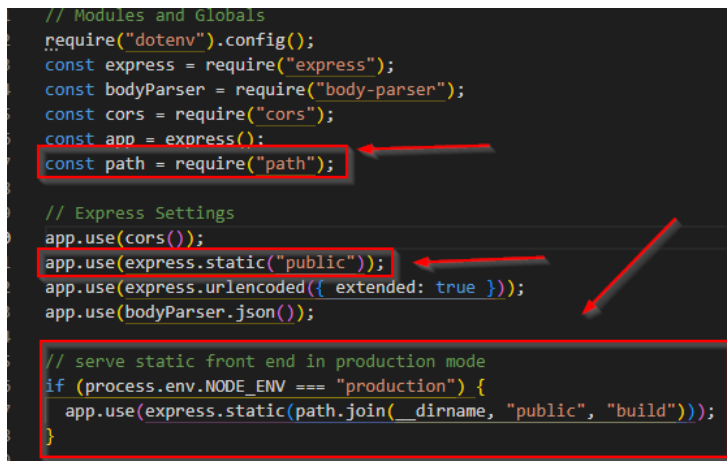
> // require "path
> `const path = require("path")`

```
// serve static front end in production mode
if (process.env.NODE_ENV === "production") {
    app.use(express.static(path.join(__dirname, 'public', 'build')));
}
```
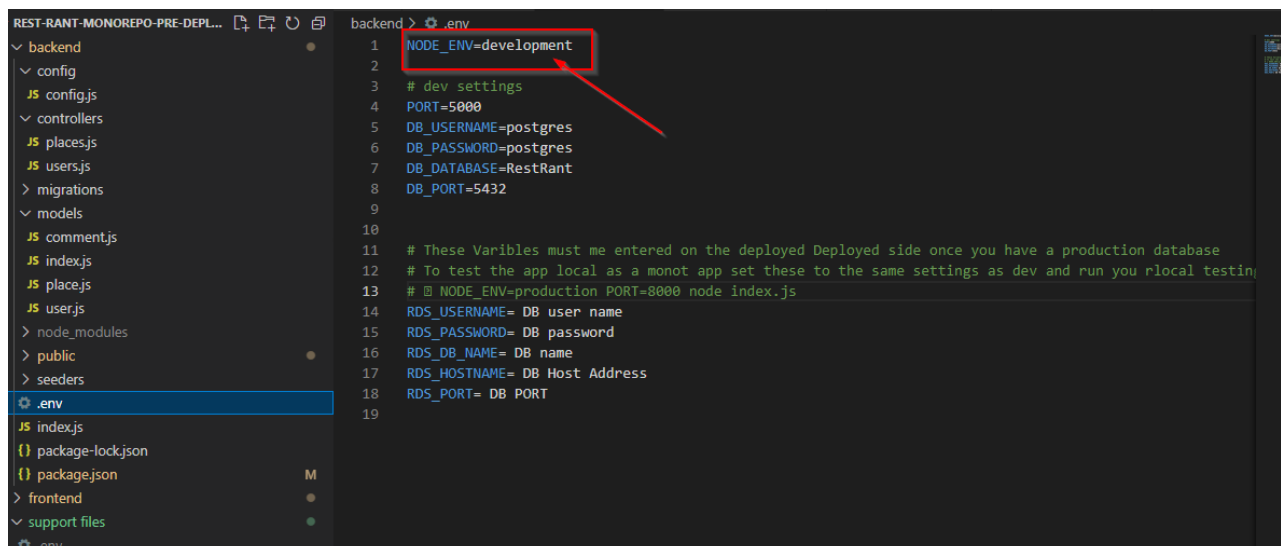
```
// Modules and Globals
require("dotenv").config();
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();
const path = require("path");

// Express Settings
app.use(cors());
app.use(express.static("public"));
app.use(express.urlencoded({ extended: true }));
app.use(bodyParser.json());

// serve static front end in production mode
if (process.env.NODE_ENV === "production") {
    app.use(express.static(path.join(__dirname, "public", "build")));
}
```

Note: *the following code NODE_ENV = production must be located in your server side .env file along with being set on the deployed system as a variable. Only set this to "production when deploying. Have it set to "development" locally prior to deploying. In the .env file there are database connection variables for both production and development. Make sure to use the production options as variables on the deployed system. When "NODE_ENV" is set to development the app will need a local front and back end running. When set to "production" express will render the frontend from the static react build folder located on the backend side.*
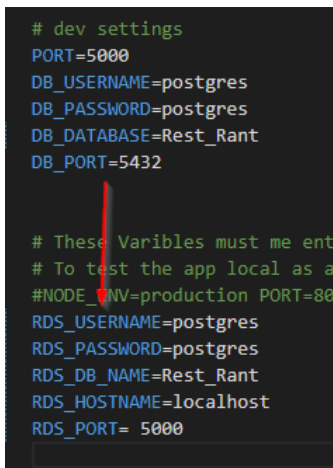
```
REST-RANT-MONOREPO-PRE-DEPL...          backend > .env
∨ backend                          1   NODE_ENV=development
  ∨ config                         2
    JS config.js                   3   # dev settings
  ∨ controllers                    4   PORT=5000
    JS places.js                   5   DB_USERNAME=postgres
    JS users.js                    6   DB_PASSWORD=postgres
  > migrations                     7   DB_DATABASE=RestRant
  ∨ models                         8   DB_PORT=5432
    JS comment.js                  9
    JS index.js                    10
    JS place.js                    11  # These Varibles must me entered on the deployed Deployed side once you have a production database
    JS user.js                     12  # To test the app local as a monot app set these to the same settings as dev and run you rlocal testing
  > node_modules                   13  # ⬜ NODE_ENV=production PORT=8000 node index.js
  > public                         14  RDS_USERNAME= DB user name
  > seeders                        15  RDS_PASSWORD= DB password
  ⚙ .env                          16  RDS_DB_NAME= DB name
  JS index.js                      17  RDS_HOSTNAME= DB Host Address
  {} package-lock.json             18  RDS_PORT= DB PORT
  {} package.json        M         19
  > frontend
  ∨ support files
  ⚙ .env
```

- Modify the start-up scripts on your server side <mark>package,json</mark> file on the backend side. **Deployment will not work** if "start" isn't pointed to "node". Modify index.js or server.js depending on how you built your code

  ```
  "scripts": {

      "dev": "nodemon index.js",        (npm run dev)

      "start": "node index.js",          (npm start)

      "test": "echo "Error: no test specified" && exit 1"},
  ```

- Run this code below in the backend to test the Mono app setup on your local machine but <mark>ONLY after you have the client side ready and the "npm run build" complete</mark>. This command will not work locally if you don't have the server side env production settings pointed to a valid database. Once you deploy, those RDS settings would be set as ENV's on the deployed server pointing to the deployed postgres DB not your local one.

  ```
  # dev settings
  PORT=5000
  DB_USERNAME=postgres
  DB_PASSWORD=postgres
  DB_DATABASE=Rest_Rant
  DB_PORT=5432


  # These Varibles must me ent
  # To test the app local as a
  #NODE_ENV=production PORT=80
  RDS_USERNAME=postgres
  RDS_PASSWORD=postgres
  RDS_DB_NAME=Rest_Rant
  RDS_HOSTNAME=localhost
  RDS_PORT= 5000
  ```

  - ➢ **NODE_ENV=production PORT=8000 node index.js**
    or
    NODE_ENV=production PORT=8000 node server.js

- ➢ Navigate to this path in the browser and the deployed ready code should render
  http://localhost:8000/

Client Side:

- Make sure the front end (client) side has a .env.production with the build path and the modified for deployment React_App_server_URL

  ➤ Example: `BUILD_PATH=../server/public/build`

  ➤ `REACT_APP_SERVER_URL=/`

- Before running the "npm run build" make sure the fetch paths are pointed to the production database path

  ➤ If you didn't use a front end env file for the fetch paths with this REACT_APP_SERVER_URL Make sure to set all the front-end fetch Paths correctly

    Example fetch path for development vs production
    - Dev: http://localhost:5000/api/getData
    - Production: /api/getdata

  ➤ Again, the easiest way to handle this is to use that REACT_APP_SERVER_URL and an .env and .env.production file in your front end.

```
useEffect(() => {
  const fetchData = async () => {
    const response = await fetch(`${process.env.REACT_APP_SERVER_URL}places/${placeId}`);
    const resData = await response.json();
    setPlace(resData);
  };
```

Front end env examples for production and development

```
support files > .env
1    BUILD_PATH=../backend/public/build
2    REACT_APP_SERVER_URL=http://localhost:5000/
3
```

EXPLORER

REST-RANT-MONOREPO...
- backend
- frontend
- support files
  - .env
  - .env.production
  - rest_rant_DB.sql   U
  - .gitignore
  - LICENSE

D:\OneDrive\Documents\ThriveDX\ClassLabs\advanceTopics\rest-rant-monorepo-

REST-RANT-MONOREPO...

> backend
> frontend
∨ support files
  ⚙ .env
  ☰ .env.production
  🗄 rest_rant_DB.sql   U
  .gitignore
  🔑 LICENSE
  ① README.md

support files > ☰ .env.production

1   BUILD_PATH=../backend/public/build
2   REACT_APP_SERVER_URL=/
3