Gulp:: Getting Deployment Ready

Introduction to Production Environment

We have created a huge folder structure wherein different folders are handling different departments of the code.

- Production Environment - A production environment is the setting where the latest working version of a computer program is installed and made available to end-users. Therefore it must always be in working condition, bug-free, and available when the end-user needs it.

Beginning Creating Environment

The purpose of a **development environment** is to have a place for a developer to test anything they want without worrying about it affecting any end-users or content editors working on a live website.

- We need to recall everything that we have done in the project, such as the password that we have chosen, the secret key generators, etc. We need to keep all this in one file.
- We need to create a file inside the { **config** } folder named as **{ Environment.js }**.
- This file will contain two environments - production and development, for that we need to define two constants.
- For now, we will export development and once we fill the production part we choose between what needs to be exported.
- We need to get in different passwords from different files and put them in the development constant inside **{ Environment.js }**.

- We need a session cookie key inside the { Environment.js } file.
- We need a database name inside the development constant.
- We need an SMTP object from the nodemailer file inside the development constant.
- We need client id, client secret, and callback URL from the **{ passport-google-oauth2-startegy.js }** file.

{ Environment.js }

```
const development = {
    name: 'development',
    asset_path: '/assets',
    session_cookie_key: 'blahsomething',
    db: 'codeial_development',
    smtp: {
        service: 'gmail',
        host: 'smtp.gmail.com',
        port: 587,
        secure: false,
        auth: {
            user: 'alchemy.cn18',
            pass: 'codingninjas'
        }
    },
    google_client_id: "313233209747-dnqmail3j800a2jvsuckqhohodhs7i63.apps.googleusercontent.com",
    google_client_secret: "0FXb5EBWa4xRfJ8jR-1HKMd2",
    google_call_back_url: "http://localhost:8000/users/auth/google/callback",
    jwt_secret: 'codeial',
}

const production =  {
    name: 'production'
}


module.exports = development;
```

{ index.js }

```
if (env.name == 'development'){
    app.use(sassMiddleware({
        src: path.join(__dirname, env.asset_path, 'scss'),
        dest: path.join(__dirname, env.asset_path, 'css'),
        debug: true,
        outputStyle: 'extended',
        prefix: '/css'
    }));
}
```

Environment Variables:: Production Environment

- We need to set up the production environment like the way we have set up the development environment. The only difference is the keys that we can see in the development environment that should be hidden. All of these will be stored somewhere in a file on the system and not in the code and that file will be accessed by the code inside the {environment.js } file so that the developers do not get access to the production level keys.
- To generate random keys we can refer to the website

  ***EXTRA*: To generate random keys we can refer to the website - https://randomkeygen.com/**

Logging for Production

- Logging is a very important aspect of development and is super useful for debugging.
- When we are in development mode in the terminal we can see the logs and we can see which error is occurring.

- Whereas when we are in the production model, the server runs in the background as a daemon process { a background process that is not visible as a running process } for which we save the logs in the files.
- To save the logs in the file we will be using a middleware that will put those logs in the file but also that file can grow huge, to prevent in growing huge either we create a backup for weekly logs or we keep on deleting the older logs.
- We will be using Morgan as a middleware so for that we need to install using the command { npm install morgan } in the terminal.
- We need to require the morgan inside the **{index.js}** file

  ***EXTRA*: You can read about morgan from the link below -**
  http://expressjs.com/en/resources/middleware/morgan.html

- We also need to install that rotating file system that will delete the files when they grow large. For now, a rotating file system is going to do the task of creating multiple log files so when one file reaches a specific file size that is defined in the documentation it will move on transferring that set of files and start filling the current logfile from scratch.
- We need to install a rotating file system using the command { **npm install rotating file system** }.

  ***EXTRA*: You can read about the rotating file system from the link below**
  https://www.npmjs.com/package/rotating-file-stream

{index.js}

```
const fs = require('fs');
const rfs = require('rotating-file-stream');
const path = require('path');


const logDirectory = path.join(__dirname, '../production_logs');
fs.existsSync(logDirectory) || fs.mkdirSync(logDirectory);

const accessLogStream = rfs('access.log', {
    interval: '1d',
    path: logDirectory
});
```

{ Environment.js }

```
const production =  {
    name: 'production',
    asset_path: process.env.CODEIAL_ASSET_PATH,
    session_cookie_key: process.env.CODEIAL_SESSION_COOKIE_KEY,
    db: process.env.CODEIAL_DB,
    smtp: {
        service: 'gmail',
        host: 'smtp.gmail.com',
        port: 587,
        secure: false,
        auth: {
            user: process.env.CODEIAL_GMAIL_USERNAME,
            pass: process.env.CODEIAL_GMAIL_PASSWORD
        }
    },
    google_client_id: process.env.CODEIAL_GOOGLE_CLIENT_ID,
    google_client_secret: process.env.CODEIAL_GOOGLE_CLIENT_SECRET,
    google_call_back_url: process.env.CODEIAL_GOOGLE_CALLBACK_RURL,
    jwt_secret: process.env.CODEIAL_JWT_SECRET,
    morgan: {
        mode: 'combined',
        options: {stream: accessLogStream}
    }
}


module.exports = eval(process.env.CODEIAL_ENVIRONMENT) == undefined ? development : eval(process.env.CODEIAL_ENVIRONMENT);
```

Logging for Production

- We need to optimize assets { images, JS,CSS }.
- We need to compress them while sending them to the server to the browser.
- Compress means - any extra space in the files is removed, the variable names are shortened and the code is generally one single line.
- For compressing the files we need to gulp.
- To install gulp we need to use the command { npm install --global gulp-cli} and { npm install gulp -D } in the terminal.
- We need to create a new file { gulp.js }.
- We need to require the gulp inside the file.
- We need to install gulp-sass using the command **{ npm install gulp-sass }** in the terminal.
- The package **gulp-sass** converts the SASS to CSS.
- We need to install a package, cssnano, that will compress the CSS into one line using the command **{ npm install gulp-cssnano }** in the terminal.
- We need to require gulp-sass and **css-nano** inside the file { gulp.js }.
- We need to install gulp rev that will rename the files with a # alongside them.
- To install gulp rev we need to use the command { npm install gulp-rev }.
- We need to require gulp rev inside the file { gulp.js }.
- Gulp contains tasks that need to be created and one of those tasks is minifying CSS.
- A pipe is a function that is calling all the sub middleware that is there in the gulp.
- For production mode we need to change the asset path, we need to put it inside the folder called public, and that public folder will contain assets that further contain the folder like - CSS, JS, images.

  { gulp.js }

```
const gulp = require('gulp');

const sass = require('gulp-sass');
const cssnano = require('gulp-cssnano');
const rev = require('gulp-rev');


gulp.task('css', function(){
    console.log('minifying css...');
    gulp.src('./assets/sass/**/*.scss')
    .pipe(sass())
    .pipe(cssnano())
    .pipe(gulp.dest('./assets.css'));

    return gulp.src('./assets/**/*.css')
    .pipe(rev())
    .pipe(gulp.dest('./public/assets'))
    .pipe(rev.manifest({
        cwd: 'public',
        merge: true
    }))
    .pipe(gulp.dest('./public/assets'));
})
```

***EXTRA*: You can read about Gulp from the link below**
https://gulpjs.com/

Completing Gulp:: JS and CSS Minification

- We have minified the JS and CSS files inside the **{ gulp.js }** file.

{ gulp.js }

```
const gulp = require('gulp');
const sass = require('gulp-sass');
const cssnano = require('gulp-cssnano');
const rev = require('gulp-rev');
const uglify = require('gulp-uglify-es').default;
const imagemin = require('gulp-imagemin');
const del = require('del');
gulp.task('css', function(done){
    console.log('minifying css...');
    gulp.src('./assets/sass/**/*.scss')
    .pipe(sass())
    .pipe(cssnano())
    .pipe(gulp.dest('./assets.css'));
    gulp.src('./assets/**/*.css')
    .pipe(rev())
    .pipe(gulp.dest('./public/assets'))
    .pipe(rev.manifest({
        cwd: 'public',
        merge: true
    }))
    .pipe(gulp.dest('./public/assets'));
    done();});
gulp.task('js', function(done){
    console.log('minifying js...');
    gulp.src('./assets/**/*.js')
    .pipe(uglify())
    .pipe(rev())
    .pipe(gulp.dest('./public/assets'))
    .pipe(rev.manifest({
        cwd: 'public',
        merge: true }))
    .pipe(gulp.dest('./public/assets'));
    done()});
gulp.task('images', function(done){
    console.log('compressing images...');
    gulp.src('./assets/**/*.+(png|jpg|gif|svg|jpeg)')
    .pipe(imagemin())
    .pipe(rev())
    .pipe(gulp.dest('./public/assets'))
    .pipe(rev.manifest({
        cwd: 'public',
        merge: true }))
    .pipe(gulp.dest('./public/assets'));
    done();});
```

- Whenever we are building a project we need to clear the previous build and build from scratch so for that we need a task inside the **{gulp.js}** file.

- We have created one more task build that will cover all the four tasks for JS, CSS, images, and clean asset tasks.

{ gulp.js }

```
// empty the public/assets directory
gulp.task('clean:assets', function(done){
    del.sync('./public/assets');
    done();
});

gulp.task('build', gulp.series('clean:assets', 'css', 'js', 'images'), function(done){
    console.log('Building assets');
    done();
});
```

View Helpers

- Helpers are basically functions that can be used inside the view by passing them to locals.
- We will create a helper function that will be based upon the development or the production environment.
- We will create a file inside the config folder { **view_helper.js**}.

{ **view_helper.js**}

```
const env = require('./environment');
const fs = require('fs');
const path = require('path');

module.exports = (app) => {
    app.locals.assetPath = function(filePath){
        if (env.name == 'development'){
            return filePath;
        }

        return '/' + JSON.parse(fs.readFileSync(path.join(__dirname, '../public/assets/rev-manifest.json')))[filePath];
    }
}
```

- We need to require the file { **view_helper.js**} inside the **{index.js**} file.

{index.js}

```
const express = require('express');
const env = require('./config/environment');
const logger = require('morgan');
```

Summarizing Production Setup

- We set up the development environment and moved all the passwords and keys over there.
- We set up the production environment that was the copy of the production environment. The only difference was that all the keys were put up into the bash profile.
- We have installed a gulp that minimizes, renames, and creates a manifest.
- Gulp is a project builder.
- To support the project builder we created a view helper.
- View helpers can be globally available by parsing them into the locals of the app.