DPR Next Gen Sustainability Dev Guide

Newly Migrated System and DPR Cal Rewrite - Spring 2022

This document focuses on aspects of the project of interest to developers continuing our work, and includes instructions such as when to update the development environment setup and use, or how to make modifications and extensions to the project. It includes all commands required to install development tools and dependencies on a given operating system, run the software from a checkout, and run all tests.

# Database Design

This is the database for the calendar application. The core component is the creation of *Courses*, which can have an unlimited number of *Sections*. Sections relate back to their Course, but also have an *Instructor* and a *Roster* students. The “instructors” table is linked to *users* by its user\_id which will have to, but is composed of its own object with demographic information. A *Roster* has zero or more *Users*, which are students enrolled in the Section of the Course.

The DPR Training Calendar does not have functionality for adding users, as this is a system wide process in the legacy monolith, so currently, users will have to be manually added to the *users* table for use in the training calendar application.



# Frontend

The frontend of the DPRCal application is written in React JS. We use react-router to navigate around the webpage, MaterialUI for styling our components, and FullCalendar for our calendar component. This section will be split into several subsections: App Structure, MaterialUI, API Calls, and testing.

App Structure and Navigation

The root directory for the frontend is dprcal/frontend. It is here that any npm install commands should be run to install new dependencies. Inside this directory is App.js, which is the main file for this application. App.js contains all of the routes for the application, which are defined inside route components. Each route component contains a path, which is the path inside the application, and an element, which is the component that path goes to. Note that the root path (‘/’) is set to the login page, so going to the base path (in this case localhost/new/dprcal) takes you straight to the login page. To navigate to routes, you first define the navigate function inside your component with:

const navigate = useNavigate();

Then, you can simply use:

navigate(‘path’);

to navigate to different paths in the application. Once a user logs in, they are redirected to one of the landing pages depending on the role the backend returns. These landing pages contain buttons, which when clicked call the navigate function to navigate to the correct path in the application.

Aside from App.js and the login page, the components for all other views and pages are in their own folders inside the dprcal/frontend directory.

MaterialUI

Rather than styling our pages with CSS, we are using MaterialUI to provide pre-styled basic components for us to use. These components include Button, Select, TextField, Autocomplete and others. Formatting the location of the components on the page is achieved using Grid and Box. Box is used to provide padding and margins around a group of components, while grid is used to create columns and define spacing across the page.

API Calls

All calls to the backend are handled through the fetch() function. This function is used to make a call to the backend, then wait for the response and do something with the response it gets. The format of the call is as follows:

fetch(‘route\_to\_endpoint’, requestOptions)

.then(response => response.json())

.then(data => do\_something\_with\_data(data));

The request options should contain the HTTP method and the credentials for authentication. If it is a put or post request, it should also include the content type and the body. The fetch() function works by sending a request to the api, waiting for the response, converting the response to json, and then taking the data it receives and doing something with, like printing out a message or setting a variable.

Testing

All of the tests for the frontend are located inside the directory containing the component being tested. For example, the tests for CreateOrEditCourse.js are inside the CreateOrEditCourse folder, and are called CreateOrEditCourse.test.js. To run all the tests, navigate to dprcal/frontend, and then run the following command:

npm test – –coverage

To run just a specific test:

npm test testfile.test.js – –coverage

## Legacy Migration Notes

### Upgrade to PHP 7.4

The primary changes encountered when updating to PHP 7.4 where the following:

* True deprecation of the mysql module in favor of the mysqli module.
* An upgrade of the Apache version packaged in the updated php Docker image.

### Mysql module deprecation

The mysql module was deprecated as of PHP 5.5.0, neccesitating its upgrade. Furthermore, the official PHP Docker image does not support installing this plug with a sufficiently high PHP version specified. Luckily, there is a near drop-in replacement in mysqli. Most of the method signatures are similar and replacements are easy to perform, especially in an IDE with some form of intellisense. Once these methods were updated, no other PHP issues were discovered.

The upgrade to mysqli does not come with any inherent security improvements. To take advantage of it, a manual refactor is required in order to prevent sql injection using query binds.

### Apache version bump

The Apache version bundled with the offical PHP 5.4 Docker image was 2.4.10, while the PHP 7.4 image contains 2.4.52. This version bump came with some minor adjustments to the config files that needed to be accounted for. Ultimately, the legacy application Apache setup is a fairly minimal deviation from the default setup.

# Stack Networking

## NGINX Reverse Proxy

The NGINX reverse proxy redirects requests to the DPR stack to the appropriate container in the stack based on the request’s URL. This allows functionality to be switched between containers (such as when a legacy app is rewritten) with the change of a couple lines in the NGINX config.

The reverse proxy also provides the service of rewriting requests to remove path prefixes such as /api/dprcal/ to make containers ignorant of their position behind a reverse proxy.

It’s important to note that should any change be desired in the routing of a request within the stack, the nginx config will be the primary method of doing so.

# Project Orchestration

## Docker Compose

Docker Compose is the backbone of the new DPR stack. It manages the building, configuration, and running of all of the application containers. A description of containers and Docker Compose is out of the scopt of this document so please follow these links for details:

* <https://docs.docker.com/get-started/>
* <https://docs.docker.com/get-started/08_using_compose/>

## Secrets

Any piece of sensitive configuration information is considered a “secret.” Examples of secrets are the encryption keys for https, database admin passwords, etc. The proper handling of these secrets is imperative for application security. Luckily, Docker Compose provides a handy method for inserting these secrets into our application at runtime. The DPR stack has adopted the convention of application secrets being declared in plain text in files in the secrets directory of the project. On runtime, the path to these secrets are identified by environment variables and loaded in by Compose. Once read in, Compose copies them into a folder in the appropriate containers with appropriate permissions as specified in the docker-compose.yaml. From there, the applications that need them can access them as regular files with minimal permissions.

For more information, see <https://docs.docker.com/compose/compose-file/#secrets>.

## Environment Variables

There are several points in the Docker Compose file where parameters are specified with environment variables. These are intended to be specified primarily in a .env file, filled with key-value pairs, stored in the envs/ folder. That said, any parameter that is also specified in the shell will overwrite the corresponding key in the .env file. Any environment variables not specified in either the shell or the .env specified with –env-file cmd arg will be set to an empty string resulting in unknown behavior.

It is recommended that any parameter that should be specified at runtime is specified using an environment variable with a description in example.env in the envs directory.

# API Documentation

Below is a link to the full public API for viewing hosted on SwaggerHub:

<https://app.swaggerhub.com/apis/joshbroddy/dpr-cal_api/1.0.0>

And below this, is a link to the html documentation for this API:

<https://app.swaggerhub.com/apis-docs/joshbroddy/dpr-cal_api/1.0.0>

This is just the documentation, but to modify, the yaml to create these docs is found in the *src\dprcal\documentation\api\_html* directory of the project.

The html documentation can also be found in *documentation\DPR\_CAL\_API\_HTML* in the project.