Delivery Console Documentation

Mozilla Product Delivery

Mar 25, 2019

Contents

Delivery Console is a unified admin UI for the many delivery pipelines to the browser.

CHAPTER 1

Developer Documentation

These documents describe how to set up Delivery Console and maintain a development environment for it.

1.1 Developer Setup

The following describes how to set up an instance of the site on your computer for development.

1.1.1 Prerequisites

This guide assumes you have already installed and set up the following:

1. Git
2. Node.js 8 and Yarn.
3. Python 2.7 or higher

These docs assume a Unix-like operating system, although the site should, in theory, run on Windows as well. All the example commands given below are intended to be run in a terminal.

1.1.2 Installation

1. Clone this repository or your fork:

```
git clone https://github.com/mozilla/delivery-console.git
cd delivery-console
```

2. Install the dependencies using yarn:

```
yarn install
```

Once you've finished these steps, you should be able to start the site by running:

```
yarn start
```

The site should be available at http://localhost:3000/.

1.1.3 Therapist

If you want to automatically enforce Delivery Console's code style guidelines, you can use the Therapist pre-commit hook.

You could install Therapist in a virtualenv using *pip* but if you are installing it globally we recommend using pipsi:

```
pipsi install therapist
```

After that, you should be able to run the following to set up the git pre-commit hook:

```
therapist install
```

After that, whenever you make a new commit Therapist will check the changed code. This will save time when submitting pull requests.

If you want Therapist to attempt to automatically fix linting issues you can install the hook using:

```
therapist install --fix
```

If you ever need to bypass Therapist, you can do so by passing `--no-verify` to your `git commit` command.

1.2 Local Configuration

There are a number of *configuration variables* that can be overridden.

For local configuration, variables can simply be passed on the command line when starting the development server:

```
$ REACT_APP_OIDC_DOMAIN=my.oidc.domain.example.com yarn start
```

However, it is recommended that you create a `.env` file in the root of the project instead. For example:

```
REACT_APP_OIDC_DOMAIN=my.oidc.domain.example.com
REACT_APP_OIDC_CLIENT_ID=6YRYpJyS5DnDyxLTRVGCQGCWGo2KNQLX
```

1.3 Workflow

The following is a list of things you'll probably need to do at some point while working on Delivery Console.

1.3.1 Running Tests

You can run the automated test suite with the following command:

```
yarn test
```

If you'd prefer that the tests watch for changes and re-run automatically:

```
yarn test:watch
```

Note: If you encounter an error like this, when running `yarn test:watch`:

```
 yarn run test:jest
yarn run v1.7.0
 $ react-app-rewired test --env=jsdom
 2018-06-20 13:55 node[6928] (FSEvents.framework) FSEventStreamStart: register_with_
↪server: ERROR: f2d_register_rpc() => (null) (-22)
 2018-06-20 13:55 node[6928] (FSEvents.framework) FSEventStreamStart: register_with_
↪server: ERROR: f2d_register_rpc() => (null) (-22)
 2018-06-20 13:55 node[6928] (FSEvents.framework) FSEventStreamStart: register_with_
↪server: ERROR: f2d_register_rpc() => (null) (-22)
 events.js:167
       throw er; // Unhandled 'error' event
       ^

 Error: EMFILE: too many open files, watch
     at FSEvent.FSWatcher._handle.onchange (fs.js:1372:28)
 Emitted 'error' event at:
     at FSEvent.FSWatcher._handle.onchange (fs.js:1378:12)
 error Command failed with exit code 1.
 info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
```

A probable cause is that you don't have `watchman` installed. For example, on macOS you can fix this by installing. . . :

```
$ brew update
$ brew install watchman
```

1.3.2 Linting

You will need to *install Therapist* for linting. If you have installed the pre-commit hook linting will take place with every commit, however there may be times you want to run the linters manually.

To run the linters on all files that you have changed or added:

```
therapist use lint
```

To run the linters on all files in the repo:

```
therapist use lint:all
```

To run the linters and attempt to fix issues in files that you have changed or added:

```
therapist use fix
```

To run the linters and attempt to fix issues in all files in the repo:

```
therapist use fix:all
```

1.3.3 Production Builds

If you need a production build to debug locally you can create one using:

```
yarn build
```

1.3.4 Redux DevTools

In development mode we have integrated Redux DevTools to help debug issues. To toggle the DevTools, hit `Ctrl-H`. You can change the side of the screen the tools are docked on using `Ctrl-Q`, and can resize the tools by dragging the edge of the bar.

1.4 Code Style Guidelines

1.4.1 Goals

- **Uniformity in code**
  - If you look at code and can tell who wrote it, that's not good
- Rules should be automatable
- Code should be easy to read / understand

1.4.2 Rules

Prettier formatting (uniformity)

Delivery Console uses Prettier to format code. If you set up the *Therapist* pre-commit hook, it can handle formatting your code automatically.

Commenting (uniformity)

Function documentation (jsdoc/docstrings)

```
/**
 * Given an option and its parent group, update the filter state based on the␣
↪`isEnabled` prop
 *
 * @param  {Object}  group
 * @param  {Object}  option
 * @param  {Boolean} isEnabled
 */
function selectFilter({ group, option, isEnabled }) {
  return {
    type: group.value === 'text' ? SET_TEXT_FILTER : SET_FILTER,
    group,
    option,
    isEnabled,
  };
}
```

Rule: Use full width for wrapping comments (80-100 chars)

1.5 Updating The Documentation

You need to have Python to build the documentation locally.

The documentation is built automatically in our continuous integration every time a commit is pushed.

To build the documentation, create and activate a Python virtualenv. For example:

```
$ virtualenv -p `which python3.6` .venv
$ source .venv/bin/activate
```

Now install the Sphinx packages:

```
(.venv) $ pip install -r docs/requirements.txt
```

Now you should be able to build:

```
(.venv) $ cd docs
(.venv) $ make html
```

Watch out for build errors but if all goes well, you can now open the built HTML files in your browser. E.g.:

```
(.venv) $ open _build/html/index.html
```

1.6 Authentication

By default, authentication takes place on `auth.mozilla.auth0.com` which is hosted by Auth0.com.

While you must use Auth0 for authentication you can *override the configuration* of `REACT_APP_OIDC_CLIENT_ID` and `REACT_APP_OIDC_DOMAIN` to use another instance.

You will need to update the backend services (such as Normandy) to use the same domain as well.

1.6.1 Debugging Silent Authentication

The way the authentication works is that a never-ending loop checks if the access token has expired, or is about to expire. Actually, it only uses `localStorage.expiresAt` to do this. To debug this you can either sit very patiently and wait till the check ticks again, or you can speed it up manually. First, to control how often the check ticks, you can override `REACT_APP_CHECK_AUTH_EXPIRY_INTERVAL_SECONDS` when starting the dev server:

```
$ REACT_APP_CHECK_AUTH_EXPIRY_INTERVAL_SECONDS=10 yarn start
```

That will cause the check to run every 10 seconds.

Secondly, to avoid awaiting for the access token to expire, you can paste this function into the Web Console:

```
window.windExpires = hours => {
  let expires = JSON.parse(localStorage.getItem('expiresAt')) - hours * 1000 * 3600;
  localStorage.setItem('expiresAt', JSON.stringify(expires));
};
```

Now you can type, in the Web Console:

```
windExpires(1.5)
```

That will simulate that 1.5 hours on the `localStorage.expiresAt` has gone past.

CHAPTER 2

Operations Documentation

These documents relate to deploying and maintaining Deliver Console in a server environment.

2.1 Configuration

All configuration happens through environment variables.

2.1.1 Delivery Console settings

An environment variable like `REACT_APP_FOO` controls the setting `FOO`.

REACT_APP_SENTRY_PUBLIC_DSN

Default `null`

Optional. The DSN for Raven to report errors to Sentry.

REACT_APP_NORMANDY_ADMIN_API_ROOT_URL

Default `https://localhost:8000/api/`

The root url for the Normandy API that should be used for users with access to the Normandy admin.

REACT_APP_NORMANDY_READ_ONLY_API_ROOT_URL

Default `null`

The root url for the Normandy API that should be used for users without access to the Normandy admin. It will be used as a fallback in case the admin server is inaccesible.

REACT_APP_OIDC_CLIENT_ID

Default `hU1YpGcL82wL04vTPsaPAQmkilrSE7wr`

The Auth0 client ID to be used when authenticating.

REACT_APP_OIDC_DOMAIN

Default `auth.mozilla.auth0.com`

The Auth0 domain to use for authenticating the user.

REACT_APP_OIDC_CALLBACK_URL

Default The TLD origin for the current URL

The URL to redirect users back to after the Auth0 authentication dance is complete.

REACT_APP_OIDC_AUDIENCE

Default `https://${OIDC_DOMAIN}/userinfo`

The audience for the access token that is generated by Auth0.

REACT_APP_CHECK_AUTH_EXPIRY_INTERVAL_MS

Default 300000

How often to issue a silent authentication refresh. Technically, when you're logged in, an infinite loop is run that refreshes your access token forever. And this number is the number of milliseconds to sleep between each check.

This number is also used to preemptively trigger a refresh. Meaning, if the access token hasn't yet expired but it will *in* `REACT_APP_CHECK_AUTH_EXPIRY_INTERVAL_MS` milliseconds, that triggers an authentication refresh too.

Index

E