Informe

En el siguiente documento se van a explicar la desiciones de diseño tomadas a lo largo del desarrollo de la herramienta, como así también se van a explicar un poco más en detalle los procesos de desarrollo acordes a los frameworks seleccionados, y sus respetivos flujos de despliegue.

Tecnologias

Front-end

El stack de tecnologías usado en el desarrollo de la aplicación font-end consta de las siguientes librerias

React

El desarrollo se encaró usando React 17, iniciado con el comando `create-react-app`. Esto se hizo de esta forma por varias razones, la principal era tener un desarrollo estándar, para facilitar la inclución de cualquier persona al mismo. La segunda razón, es para evitar la configuración de productos el tratamiento de los archivos estáticos, como puede ser Webpack, Babel, entre otros.

Material-UI

Para evitar la escritura de componentes y estilos ya desarrollados previamente por la comunidad, se hizo uso de la librería de componetes visuales Material-UI, la cual cuenta con un gran apoyo de la comunidad, y se encuentra en constante crecimiento.

De esta librería se aprovecharon varias componentes, entre ellas la que le da la estructura base al sistema, el `DataGrid`, que nos permite enfocarnos en preparar el conjunto de datos, y las funcionalidades necesarias para la interacción con el back-end, sin preocuparnos por estilos visuales, ni lógicas de visualización.

use-http

Para gestionar todas las llamadas a la API, y centralizar la configuración y la lógica en un solo lado, se hizo uso de esta librería, que nos permite tanto encapsular la gestión del estado interno (datos, errores, variables de estados útiles para la visualización, etc), como gestionarlo de forma programática, sin preocuparnos por las particularidades de cada error HTTP.

Además, se aprovechó el concepto de `Provider` que nos permite centralizar en un solo archivo (en nuestro caso, el `App.js`, la configuración de la URL de la API del backend, para facilitar futuros refactors. Esto además quedó parametrizado via variablers de ambiente.

Back-end

strapi

Para facilitar la gestión y mantención de los modelos de datos, y endpoints HTTP del back-end, se eligió usar este producto que nos permite definirlo de una forma visual, para terminar generando código que luego permite replicar y desplegar de forma simple un back-end.

En nuestro caso, además de la configuración de base generada por el producto, se hicieron 2 personalizaciones. Una para soportar la carga de una semilla de datos inicial con ingredientes activos, aptitudes y cultivos, mediante un archivo de texto plano. La otra para modificar levemente la respuesta de los endpoints de datasets (listado y edición), para agregar un nivel extra en la hidratación de los objetos, en este caso, para que el listado de residuos límite propio de cada dataset, ya sea retornado con los objetos de ingredientes activos, aptitudes y cultivos con todos los atributos, esto para evitar una carga de lógica en el front-end.

Dichas personalizaciones son posibles gracias al principio de diseño de inversión de control aplicada por el frameworks de strapi, que nos permiten implementar estas prociones de lógica personalizadas entrelazada con la lógica propioa del framework.

Patrones de react

A lo largo del desarrollo del front-end, se aplicó un patron conocido en la comunidad de react como componentes de presentación y contenedor

Para más información sobre estos patrones, se puede consultar el gitbook

https://github.com/krasimir/react-in-patterns

Componentes de presentación y contenedor

Esta patrón tiene que objetivo desacoplar lo más posible la lógica de prestanción (que y como se muestra), de la lógica de negocio (como trabajar con los datos), de esta forma vamos a tener componentes de presentación, los cuales no deberian estar cargados de la lógica de negocio, salvo ciertas excepciones propias de una lógica auto-contenida de la componente, y tendríamos componentes contenedores, las cuales están encargadas de definir gran parte de la lógica para recuperar y tratar los datos, además de renderizar las componentes de presentación con haciendo uso de los datos obtenidos.

Aplicado a nuestro caso, podemos reconocer la componente contenedor `ListContainer.jsx`, que tiene entre sus responsabilidades, recuperar los datos de 4 endpoints (`/datasets`, `/active-ingredients`, `/aptitudes` y `/crops`), preparar los callbacks que se van a utilizar para actualizar los datos via API

(`/crops` y `/residual-limits`), gestionar el estado interno, y configurar componentes como el `DataGrid` y el `Modal`

Por otro lado, podemos ver componentes de presentación como `DatasetComponent.jsx`, `ResidualLimitFormComponent.jsx` y `ResourceAlerts.jsx`.

En el caso de `DatasetComponent` y `ResidualLimitFormComponent`, ambas son componentes encargadas de rendrizar un formulario para crear recursos. Dentro de sus responsabilidades, podemos ver que gestiona el estado interno del formulario, como así también las funciones callbacks usadas para modificar los inputs y arma el callback usado por el botón, haciendo uso de la función enviada desde la componente contenedor.

Por último, la componente `ResourceAlerts` se encarga de mostrar todos los mensajes de error al momento de interactuar con la API de recursos.

Desarrollo

Front-end

Estructura

A contionuación se van a mostrar cual es la estructura de directorios, y como seguirla de forma orgánica con el crecimiento del proyecto

```
.
├── README.md
├── build
│   └── ...
├── node_modules
│   └── ...
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── App.js
│   ├── components
│   │   ├── DatasetComponent.jsx
│   │   ├── ResidualLimitFormComponent.jsx
│   │   └── ResourceAlerts.jsx
│   ├── containers
│   │   └── ListContainer.jsx
│   ├── helpers
│   │   └── httpStatus.js
│   └── index.js
└── yarn.lock
```

- El `package.json` y `yarn.lock` fijan las dependencias y sus versiones dentro del proyecto de front-end
- `build`, es el directorio en donde queda la aplicación transpilada. Esto se crea automáticamente luego de ejecutar `yarn build` (**no debe subirse al repoisitorio**)
- `node_modules`, contiene todas las librerias necesarias parletsencrypt-proda el funcionamiento de la aplicación (**no debe subirse al repoisitorio**)
- `public`, contiene todos los archivos estáticos que van a ser usados para generar el index.html
- `src`
    - `App.js`, componente principal de la aplicación. Normalmente se usa para confogurar providers globales a la aplicación, o para definiciones globales.
    - `components`, contiene todas las componentes de presentación (ver patrones)
    - `contienes`, contiene todas las componentes contenedores (ver patrones)
    - `helpers`, contiene un conjunto de funciones útiles a lo largo del proyecto, pero que no se encuentran directamente relacionadas con ningun tipo de componente, ya sea de presentación o contenedor.
    - `index.js`, tiene la componente React, y como hija, la componente principal del la aplicación (`App.js`), renderizada a partir del elemento con id `root`

Otros posibles directorios:

- `src`
    - `hooks`, contiene todos los hooks personalizados de react. Sirve sobre todo como método para desacoplar las componentes contenedores.
    - `config`, contiene funciones y clases usadas para centralizar configuraciones de la aplicación. Por ejemplo, se podria contar con una personalizada de Fetch ya preconfigurada (esto lo reemplazamos por la librería `use-http`).

Back-end

Estructura

El código del back-end se encuentra bajo el directorio `backend/app`, y tiene la siguiente estructura

```
.
├── README.md
├── api
│   ├── active-ingredient
│   │   ├── config
│   │   │   └── routes.json
│   │   ├── controllers
│   │   │   └── active-ingredient.js
│   │   ├── documentation
│   │   │   └── 1.0.0
│   │   │       ├── active-ingredient.json
```

```
│   │   │       └── overrides
│   │   ├── models
│   │   │   ├── active-ingredient.js
│   │   │   └── active-ingredient.settings.json
│   │   └── services
│   │       └── active-ingredient.js
│   ├── aptitude
│   │   ├── config
│   │   │   └── routes.json
│   │   ├── controllers
│   │   │   └── aptitude.js
│   │   ├── documentation
│   │   │   └── 1.0.0
│   │   │       ├── aptitude.json
│   │   │       └── overrides
│   │   ├── models
│   │   │   ├── aptitude.js
│   │   │   └── aptitude.settings.json
│   │   └── services
│   │       └── aptitude.js
│   ├── crop
│   │   ├── config
│   │   │   └── routes.json
│   │   ├── controllers
│   │   │   └── crop.js
│   │   ├── documentation
│   │   │   └── 1.0.0
│   │   │       ├── crop.json
│   │   │       └── overrides
│   │   ├── models
│   │   │   ├── crop.js
│   │   │   └── crop.settings.json
│   │   └── services
│   │       └── crop.js
│   ├── dataset
│   │   ├── config
│   │   │   └── routes.json
│   │   ├── controllers
│   │   │   └── dataset.js
│   │   ├── documentation
│   │   │   └── 1.0.0
│   │   │       ├── dataset.json
│   │   │       └── overrides
│   │   ├── models
│   │   │   ├── dataset.js
│   │   │   └── dataset.settings.json
│   │   └── services
│   │       └── dataset.js
│   └── residual-limit
│       ├── config
│       │   └── routes.json
│       ├── controllers
│       │   └── residual-limit.js
│       ├── documentation
```

```
│       │   └── 1.0.0
│       │       ├── overrides
│       │       └── residual-limit.json
│       ├── models
│       │   ├── residual-limit.js
│       │   └── residual-limit.settings.json
│       └── services
│           └── residual-limit.js
├── config
│   ├── database.js
│   ├── env
│   │   └── production
│   │       ├── database.js
│   │       └── server.js
│   ├── functions
│   │   ├── bootstrap.js
│   │   ├── cron.js
│   │   └── responses
│   │       └── 404.js
│   └── server.js
├── data
│   ├── active-ingredient.csv
│   ├── aptitude.csv
│   └── crop.csv
├── extensions
│   ├── documentation
│   │   ├── documentation
│   │   │   └── 1.0.0
│   │   │       └── full_documentation.json
│   │   └── public
│   │       └── index.html
│   ├── email
│   │   └── documentation
│   │       └── 1.0.0
│   │           ├── email-Email.json
│   │           └── overrides
│   ├── upload
│   │   └── documentation
│   │       └── 1.0.0
│   │           ├── overrides
│   │           └── upload-File.json
│   └── users-permissions
│       ├── config
│       │   └── jwt.js
│       └── documentation
│           └── 1.0.0
│               ├── overrides
│               ├── users-permissions-Role.json
│               └── users-permissions-User.json
├── favicon.ico
├── package.json
├── plugins
├── public
│   ├── robots.txt
```

```
|   └── uploads
└── yarn.lock
```

- El `package.json` y `yarn.lock` fijan las dependencias y sus versiones dentro del proyecto de front-end
- `build`, es el directorio en donde queda la aplicación transpilada. Esto se crea automáticamente luego de ejecutar `yarn build` (**no debe subirse al repoisitorio**)
- `node_modules`, contiene todas las librerias necesarias para el funcionamiento de la aplicación (**no debe subirse al repoisitorio**)
- `public`, contiene todos los archivos que posteriormente van a ser de público acceso desde el servidor
- `api`, tiene toda la configuración propia de cada uno de los recursos generados por strapi
    - `<recurso>`
        - `config`
            - `routes.js`, contiene la configuración de los endpoints
        - `controllers`
            - `<recurso>.js`, contiene los handlers usados por las rutas, en este archivo se agregó la configuración personalizada para los datasets
        - `documentation`
            - `<version>`
                - `<recurso>.json`, contiene la definición de swagger de los endpoints del recurso
                - `overrides`, contiene todas las sobreescrituras necesarias
        - `models`, define los modelos y sus hooks
            - `<recurso>.js`, define todos los hooks del modelo
            - `<recurso>.settings.json`, define el modelo en formato json
        - `services`
            - `<recurso>.js`
- `config`
- `data`
- `extensions`
- `plugins`

Para más información de todos los directorios posibles, se puede consultar la documetación oficial

https://strapi.io/documentation/developer-docs/latest/setup-deployment-guides/file-structure.html#project-structure

Despliegue

El despliegue se encuentra automatizado usando Github Actions, para más información se puede consutlar los README del backend y frontend, donde se explica como configurar e interactuar con los despliegues, ya sea de forma manual como automática bajo commits al repositorio.