TECHNICAL DOCUMENTATION

Dashboard - EPITECH 2024

Alexandre Sauner - Axel Biehler

1. API

The API is a NodeJS REST API using express at its core. Routers such as authentication, services or instances can be found in the `routes` folder. These routers are used in the `main.js` file.

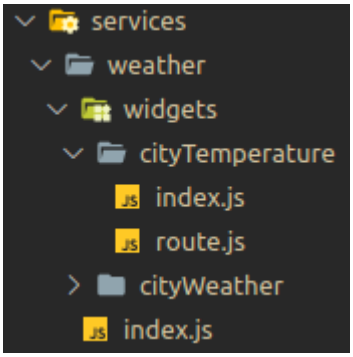
The API handles CORS.

The database used is Mongo. It is queried with Mongoose. Models can be found in the `database` folder.

Authentication is done using Json Web Tokens. Passwords are hashed using BCrypt. An `authMiddleware` is available in the `authentication` folder. This middleware makes sure a token is passed in the `Authorization` header. It verifies the token and adds the user identifier to the Request object under the `userId` key.

Services and widgets are automatically imported with dynamic requirements.

Services and widgets must be created in the `services` folder. Here is the file structure of a service `weather` with a `cityTemperature` widget:



`index.js` files contain metadata for either a service or a widget. route.js files export Express route function which are linked to widgets.

Each service must export a simple metadata object:

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

const metadata = {
  name: path.basename(__dirname),
  displayName: 'Weather',
  description: 'Use the OpenWeather API to get weather data.',
  needsOAuth: false,
};

module.exports = metadata;
```

Each widget must do the same:

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

const metadata = {
  name: path.basename(__dirname),
  displayName: 'City temperature',
  description: 'Display temperature for a city',
  params: [
    {
      name: 'city',
      type: 'string',
      placeholder: 'City name',
    },
  ],
};

module.exports = metadata;
```

A. Routes

a. Authentication

```
POST /auth/login
{
   "username": "alexandre",
   "password:" "password"
}

{
   "status": true/false,
   "error": "wrong password"
}

POST /auth/register
{
   "username": "alexandre",
   "password": "password"
}

{
   "status": true/false,
   "error": "password too short"
}
```

b. Services

```
GET /services

[
   {
       "name": "weather",
       ...
   },
   ...
]
```

c. Instances

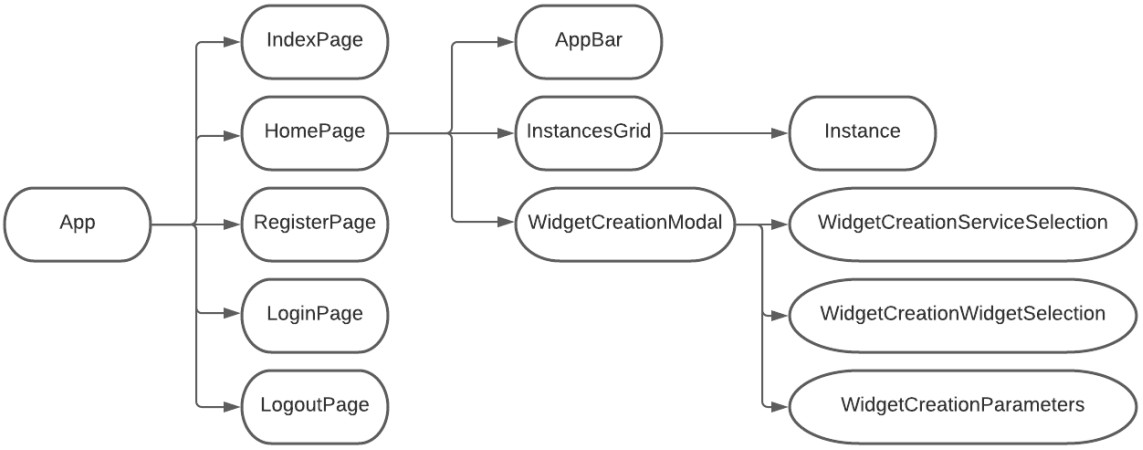
```
GET /instances

{
   "status": true/false,
   "error": "example error"
   "instances": [
       {
           "_id": "618d7ddbbda85c8fa70e9e99",
           "service": "weather",
           "widget": "cityTemperature",
           "params": {
               "city": "Strasbourg",
               "refreshRate": 30
           }
       },
       ...
   ]
}

POST /instances

{
   "service": "weather",
   "widget": "cityTemperature",
   "params": {
       "city": "Paris",
       "refreshRate": 10
   }
}

{
   "status": true/false,
   "error": "example error"
}
```

```
GET
/instances/weather/cityTemperature/618d7ddbbda85c8fa
70e9e99

{
   "status": true,
   "temp": 7.76,
   "city": "Strasbourg",
   "country": "FR"
}

DELETE /instances/618d7ddbbda85c8fa70e9e99
{
   "status": true/false,
   "error": "example error"
}
```

2. Front-End

The front-end framework used is React. Useful npm packages used are:

- @mui/material
- react-grid-layout



3. Create a service and a widget

A. Back-end

Create a folder in the services directory; the name of the folder is the name of the service. In this folder, create an index.js with the following content, update the values as needed.

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

const metadata = {
  name: path.basename(__dirname),
  displayName: 'Weather',
  description: 'Use the OpenWeather API to get weather data.',
  needsOAuth: false,
};

module.exports = metadata;
```

Create a folder in your new service folder, the name of the folder is the name of the widget. In this folder, create an index.js with the following content, update the values as needed.

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

const metadata = {
  name: path.basename(__dirname),
  displayName: 'City temperature',
  description: 'Display temperature for a city',
  params: [
    {
      name: 'city',
      type: 'string',
      placeholder: 'City name',
    },
  ],
};

module.exports = metadata;
```

In this folder, create a route.js file. This file exports an expressjs route which will be called when the widget instance fetches its data.

```
const fetch = require('node-fetch');

const route = async (req, res) => {
  try {
    const r = await
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${req.instance.
params.city}&appid=${process.env.OPENWEATHER_API_KEY}&units=metric`);
    const body = await r.json();
    res.json({
      status: true,
      temp: body.main.temp,
      city: body.name,
      country: body.sys.country,
    });
  } catch (e) {
    console.error(e);
    res.status(500).json({
      status: false,
      error: 'internal error',
    });
  }
};

module.exports = route;
```

B. Front-end

Create a folder in the `/src/components/services` folder for your service. In this folder, create a component for your widget. This component receives a `data` prop which is the output of your widget's route. The component must then be registered in the `Instance.jsx` file, in the `InstanceSwitch` component.