# Front-end Application for Authentication API

Proof of Concept

*Version 0.0*

**André São Pedro**

**E-mail**: andrefbsaopedro@gmail.com

**GitHub**: /afbsaopedro

16 July 2024

# Introduction

## Overview

This is a simple front-end application that includes server-side components to communicate with an authentication API to authenticate and validate users. Upon successful validation, at every step, the API responds with a token and a path to redirect the user.

## Features

This application has the following features:

- User validation and authentication
- Token-based authorization and validation
- Redirection based on API response

## Technology Stack

- Vue.js - v3.4.31
- Node.js - v20.12.0
- Nuxt.js - v3.12.3
- NuxtUI - v2.17.0
- Authentication API

  Nuxt components serve as way to bring together the server-side components with the front-end components in a easy way.

# Architecture

This application uses a server-side component to handle user authentication by communicating with an external API.

## Important Modules/Components

All the components can be found in the root directory

- **Layouts**: The `.vue` component is what provides the navigation bar
- **Pages**: Directory where all the pages can be found
- **Server**: Directory where the endpoints are found
- **app.vue**: Component that dictates how each component and each page are rendered. Note that the `NuxtLayout` encapsulates `NuxtPage`. This structure tells us where each of these components is to be rendered

# API Documentation

The application makes use of server-side components so it's able to communicate with the external Authentication API.
The basic operation of the application is as follows:

1. Front-end makes a request to the application's internal API
2. Internal API endpoint is mapped to an endpoint in the external authorization API
3. Response trickles down all the way to the front-end and action is taken accordingly, usually a redirect to the correct page

## Endpoints

The internal API is mapped through the directory structure. The name of the file also identifies the request verb. As such all endpoints are prefixed as `/api/auth/(end_point_here)`:

1. `GET /users` : Fetches all users and their data. **Used for debug purposes only**.
2. `POST /new-user` : Posts the input from the page's form, constructs a validator and creates the corresponding user. **Used for debug purposes only**.
3. `POST /id` :
    - **File:**: id.post.ts
    - **Destination**: `dummy/auth/id`

- **Request**: `{ "userId": "string" }`
- **Response**: `{ "token": "string", "redirect": "string" }`

4. `POST /login` :
    - **File:**: login.post.ts
    - **Destination**: `dummy/auth/login`
    - **Request**: `{ "placeholder": "json", "placeholder": "json" }`
    - **Response**: `{ "placeholder": "json", "placeholder": "json" }`
5. `POST /signup` :
    - **File:**: signup.post.ts
    - **Destination**: `dummy/auth/signup`
    - **Request**: `{ "placeholder": "json", "placeholder": "json" }`
    - **Response**: `{ "placeholder": "json", "placeholder": "json" }`
6. `POST /validate` :
    - **File:**: validate.post.ts
    - **Destination**: `dummy/auth/validate`
    - **Request**: `{ "placeholder": "json", "placeholder": "json" }`
    - **Response**: `{ "placeholder": "json", "placeholder": "json" }`

## Error Handling

Error Handling is done in only one way in the entire application.
Since the response structure differs depending on the status of the request (whether it failed or was successful), after we receive the response we check for a `redirect` property like so:

```
if('redirect' in response) {
            // succeful code
        } else {
            // failure code
        }
```

Everywhere in the application `succeful code` and `failure code` are always a notification that is shown to the user, which will be explained later on in **`Other components > Notifications`** .

# Other components

## Token Storage

The application needs a way to store the tokens it receives so it can send them again. The way this is done is through `const token = useState('token)` which is declared in the `app.vue` component so it is accessible globally in the entirety of the application. After we receive a `response` and store it, we can assign the token value like so:

```
const token = useState('token)
token.value = response.token
```

From here it is then sent in the POST request (check Swagger UI for reference).

## Notifications

In every page you have `const toast = useToast()` and `toast.clear()` declared. The first allows you to use the notification component provided by Nuxt and clear it respectively.
The way this is used is when you receive a response from a request, you build the notification as follows:

```
toast.add({
    title: 'Bad Validator',
    description: 'Validator is invalid.\nIf the problem persists, please contact us.',
    icon: 'i-heroicons-x-circle',
    color: "red",
    timeout: 6000,
    actions: [{
        label: 'Contact',
        click: () => {
            navigateTo('/contact')
        }
    }]
})
```

The `actions` property of the notification is optional.

Icons use can be found here.

## Redirects and Navigation

The `navigateTo()` method is provided to redirect the user to a page. It takes a `string` as argument and you feed the value of `redirect` that you get from a request response to redirect the user to the correct page

# Additional Resources

## References

- Vue.js Documentation
- Node.js Documentation
- Nuxt.js Documentation
- NuxtUI Documentation