Documentation AREA

Description:

This project is built using Docker, ensuring a streamlined development and deployment process. The project structure is organized into distinct components: the backend (including database and API access), the website, and the mobile application. Each component has its own Dockerfile located within its respective directory, and there is a docker-compose.yaml file at the root of the project.

The docker-compose.yaml file orchestrates the build process by referencing the Dockerfiles in the following directories: /Backend, /FrontendAPP, and /FrontendWeb. It builds the entire project and sets up the server. You can launch it via the executable file launch_docker.sh.

The individual Dockerfiles are responsible for constructing each component (/Backend, /FrontendAPP, and /FrontendWeb) to enable the docker-compose.yaml to successfully launch the entire project.

Web Frontend:

Description:

This part is the frontend web of the Area project. It provides a web user interface to define and manage automation rules (actions and reactions), while ensuring a smooth user experience for viewing and tracking ongoing automations.

Installation:

To install and run the frontend web locally:

1. Prerequisites

   Before getting started, make sure you have the following installed on your machine:

2. Node.js (version 14.18+ or 16+ recommended)
3. npm (comes with Node.js) or yarn.

4. Clone the repository

   ```
   git clone git@github.com:EpitechPromo2027/B-DEV-500-TLS-5-1-area-anastasia.bouby.git
   ```

   ```
   cd B-DEV-500-TLS-5-1-area-anastasia.bouby.git/FrontendWeb
   ```

5. Install the dependencies

   ```
   npm install
   ```

   or

   ```
   npm yarn
   ```

Starting the project:

The server will run by default on port 8081.

6. To start the project, use the following command

```
npm run dev
```

or

```
yarn dev
```

7. If you want to build the project and after start it independently

```
npm run build
```

```
npm run preview
```

or

```
yarn build
```

```
yarn preview
```

Content of an Area (Action-Reaction)

When an "area" is created, it starts with default values for actions and reactions. These values are automatically updated when you modify the input parameters.

Login

File: Login.jsx

8. **Purpose:**

- In the login process, the list of services is used to verify each service's token. If a token exists in the database, it updates the local storage to mark the service as connected.

9. IP Address Request:

When the login page opens, the application requests the server IP address. By default, the web version points to localhost unless otherwise specified.

Register

File: Register.jsx

10. Purpose:

- When registering, a user can create an account using the same username or email as another user. This feature does not restrict duplicate usernames or emails across users, so careful handling of user identity is advised if needed.

11. IP Address Request:

Similar to Login.jsx, this page initiates a request for the server IP address upon opening, defaulting to localhost on the web.

Menu Side Panel Options

Options: Dyslexic Font and Logout

File: App.jsx

12. Dyslexic Font Toggle:

- The menu includes an option for "Dyslexic Font," which toggles the font between "OpenDyslexic" and "Arial." When selected, it updates the database variable for the user to reflect this preference (true for OpenDyslexic, false otherwise).

13. Implementation:

  - Upon loading, App.jsx checks if the "Dyslexic Font" setting is enabled in the database. If true, the app applies "OpenDyslexic" instead of the default "Arial."

14. **Logout:**

The logout option removes all items from local storage and redirects the user back to the login page.

Routes

File: App.jsx, Title.jsx

15. **Purpose:**

- Defines and manages all routes/pages in the web application, controlling navigation and accessibility based on user login status and permissions.

Adding an Action

To add an action, follow these steps:

16. Open the Rectangle_Action.jsx file in your project directory.
17. Find menuItemsAction and create your action.

    This list records the action possible in our project. The action field to write the name of the action, the label field to write a rapid description of your action, icon for the icon of the service you use for this action and connected to set if you are connected to your service.

```
const [menuItemsAction, setMenuItemsAction] = useState([
    {
        action: 'Title',
        label: 'Description',
        icon: your_icon,
        connected: false,
    },
```

18. Write your new action logic in this file, following the existing structure to ensure compatibility.

Adding a Reaction

To add a reaction, follow these steps:

19. Open the Rectangle_Action.jsx file in your project directory.
20. Find menuItemsReaction and create your action.
    This list records the reaction possible in our project. The reaction field to write the name of the reaction, the label field to write a rapid description of your reaction, icon for the icon of the service you use for this reaction and connected to set if you are connected to your service.

```
const [menuItemsReaction, setMenuItemsReaction] = useState([
    {
        reaction: 'Title',
        label: 'Description',
        icon: your_icon,
        connected: false,
    },
```

21. Write your new reaction logic in this file, following the existing structure to ensure compatibility.

Input for Actions and Reactions:

The input system allows users to provide information necessary for each action or reaction to function correctly. Depending on the chosen action or reaction, inputs

might not be required, or multiple inputs may be necessary. The logic for handling inputs is centralized in the Rectangle_action.jsx file.

Basic input handling:

If an action or reaction requires only one input, use the handleInput function. This function provides a shared array of inputs, where each action or reaction can access a single common input field.

Adding New Inputs or Additional Information:

22. If an action or reaction requires multiple inputs or a more complex setup, you can add custom conditions and functions to handle these additional needs:
23.
24. *Single input storage:*

    When only one piece of information is required, use inputContentReact to store the value.

25. *Multiple Inputs Storage:*

    When multiple inputs are necessary (currently only reactions require multiple inputs):

26. Create additional variables to manage these inputs.
27. Use one variable for display purposes and another to store the final formatted data for storage.

Input Validation

When new inputs are added, ensure that each input is verified for completeness to prevent errors when applying actions or reactions.

Adding a Service:

To integrate a new service into the project, the configuration is primarily handled in the ServiceConnection.jsx and Rectangle_Action.jsx files. Each service has specific properties (text, status, and connection state) that define how it appears and behaves in the interface.

28. In Rectangle_Action.jsx

Find checkServicesConnexion and write this:

```
const checkServicesConnexion = async (area) => {
    switch (area) {
        case 'YourService':
            if (localStorage.getItem('yourservice_token') === 'false')
                return false;
            }
            break;
```

29. In ServiceConnection.jsx

30. Add an entry for the new service in the list of services.
31. Define the following properties for the new service in definieStatusService:
32. **text**: The text displayed on the button
33. **connect**: A boolean indicating the initial connection status (false if the service is not yet connected).
34. **status**: The color of the button, which will change based on the connection state.

```
const defineStatusService = async () => {
    if (localStorage.getItem('yourservice_token') === 'true') {
        setYourServiceText('Disconnection of YourService');
        setYourServiceStatus('#3AB700');
        setYourServiceConnect(true);
    } else {
        setYourServiceText('Connect to YourService');
        setYourServiceStatus('#33478f');
        setYourServiceConnect(false);
    }
```

Mobile Frontend

Description

This part represents the mobile frontend of the Area project, offering a responsive user interface optimized for mobile devices. It allows users to manage automation rules and view the status of ongoing automations on the go, with a design tailored for touch interaction.

Installation:

To install and run the mobile frontend locally:

35. Prerequisites

36. Before getting started, make sure you have the following installed on your machine:
37. Node.js (version 14.18+ or 16+ recommended)
38. npm (comes with Node.js) or yarn.

39. Clone the repository

```
git clone git@github.com:EpitechPromo2027/B-DEV-500-TLS-5-1-area-anastasia.bouby.git

cd B-DEV-500-TLS-5-1-area-anastasia.bouby.git/FrontendApp
```

40. Install the dependencies

```
npm install
```

Starting the project:

The server will run by default on port 8100.

41. To start the project and open it in android studio, use the following command

```
ionic capacitor build android
```

42. To start the project and open it in the web browser, use the following command

```
ionic serve
```

Content of an Area (Action-Reaction)

When an "area" is created, it starts with default values for actions and reactions. These values are automatically updated when you modify the input parameters.

Login

File: login.page.ts

43. Purpose:

- In the login process, the list of services is used to verify each service's token. If a token exists in the database, it updates the local storage to mark the service as connected.

44. IP Address Request:

When the login page opens, the application requests the server IP address. By default, the mobileversion points to the environment api.

Register

File: register.page.ts

45. **Purpose:**

- When registering, a user can create an account using the same username or email as another user. This feature does not restrict duplicate usernames or emails across users, so careful handling of user identity is advised if needed.

46. **IP Address Request:**

Similar to login.page.ts, this page initiates a request for the server IP address upon opening, defaulting to localhost on the web.

Menu options

File: app.component.ts

47. **Dyslexic Font Toggle:**
48. When selected, this option switches the font to "OpenDyslexic."
49. The setting is saved in the database as true (enabled) or false (disabled).
50. On page load, app.component.ts checks this setting and applies the "OpenDyslexic" font if enabled.
51. **Logout:**
52. Removes all items from localStorage.
53. Redirects the user to the login page.

File: utils.service.ts

54. The logout function handles the clearing of localStorage and manages redirection.

Routes

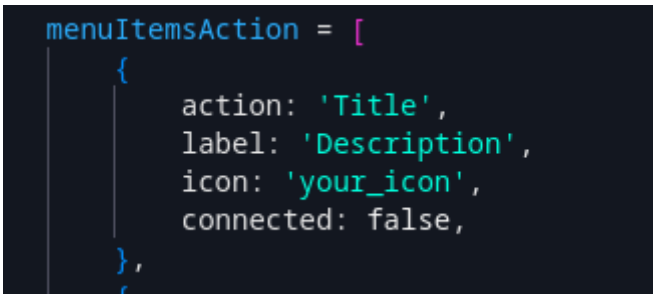
File: app-routing.module.ts

55. Manages all application routes and page navigation.

Adding an Action

To add an action, follow these steps:

56. Open the dashboard.page.ts file in your project directory.
57. Find menuItemsAction and create your action.
    This list records the action possible in our project. The action field to write the name of the action, the label field to write a rapid description of your action, icon for the icon of the service you use for this action and connected to set if you are connected to your service.



58. Write your new action logic in this file, following the existing structure to ensure compatibility.

Adding a Reaction

To add a reaction, follow these steps:

59. Open the dashboard.page.ts file in your project directory.
60. Find menuItemsReaction and create your action.
    This list records the reaction possible in our project. The reaction field to write the name of the reaction, the label field to write a rapid description of your reaction, icon for the icon of the service you use for this reaction and connected to set if you are connected to your service.

61. Write your new reaction logic in this file, following the existing structure to ensure compatibility.

Input for Actions and Reactions

The input system enables users to provide necessary information for each action or reaction to function correctly. Depending on the selected action or reaction, inputs might not be required, or multiple inputs may be needed. The logic for handling these inputs is primarily located in dashboard.page.ts and dashboard.page.html.

Basic input handling:

If an action or reaction requires only one input, use the handleInput function. This function provides a shared array of inputs, where each action or reaction can access a single common input field.

Adding New Inputs or Additional Information:

62. If an action or reaction requires multiple inputs or a more complex setup, you can add custom conditions and functions to handle these additional needs:

63. Single input storage:

When only one piece of information is required, use inputContentReact to store the value.

64. Multiple Inputs Storage:

When multiple inputs are necessary (currently only reactions require multiple inputs):

65. Create additional variables to manage these inputs.
66. Use one variable for display purposes and another to store the final formatted data for storage.

Input Validation

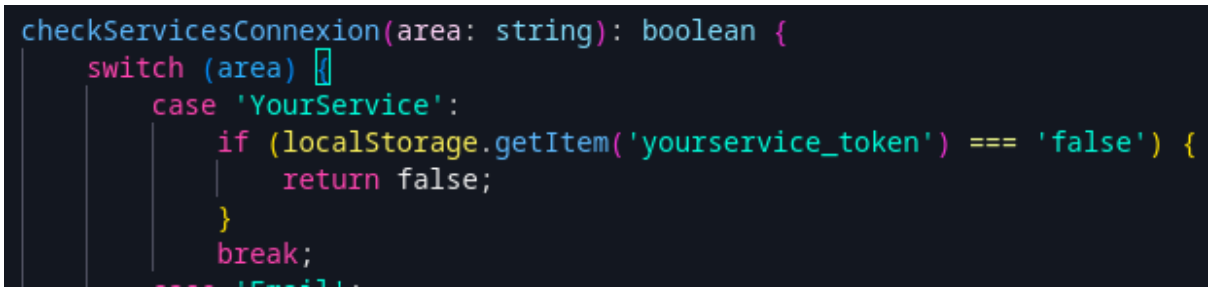
When new inputs are added, ensure that each input is verified for completeness to prevent errors when applying actions or reactions.

Adding a Service:

To integrate a new service into the project, the configuration is primarily handled in the ServiceConnection.jsx and Rectangle_Action.jsx files. Each service has specific properties (text, status, and connection state) that define how it appears and behaves in the interface.

67. In dashboard.page.ts

Find checkServicesConnexion and write this:



```
checkServicesConnexion(area: string): boolean {
    switch (area) {
        case 'YourService':
            if (localStorage.getItem('yourservice_token') === 'false') {
                return false;
            }
            break;
```

68. **In service.page.ts**

Add an entry for the new service in the list of services and add this entry in ngAfterViewInit.

69. In service.page.html

Create a new button.

This button redirects to the service login page and includes:

70. **Text:** Service name.

71. **Color:** Green when connected, blue when not.
72. **Logo:** Displays the service logo.

The button's class name changes based on the connection status to adjust the color accordingly.

```
<div class="flex-rectangle">
  <div class="main-rectangle">
    <img src="assets/yourservice.png" class="icon-container" />
    <ion-text class="connection-rect {{yourservice_connect ? 'connected' : 'notConnected'}}"
    (click)="ManageService('yourservice', yourservice_connect)">
    <h3>{{ yourservice_connect ? 'Disconnect of' : 'Connect to' }} yourservice</h3>
  </ion-text>
  <div class="main-rectangle">
```

Service:

Description:

The Service part of the project is independent from the rest. It allows the area logic to function for all users, even if the backend or the visual interface is not running.

Installation:

To install and run Service locally:

73. Prerequisites

Before getting started, make sure you have the following installed on your machine:

74. Node.js (version 14.18+ or 16+ recommended)
75. npm (comes with Node.js) or yarn.

76. Clone the repository

git clone git@github.com:EpitechPromo2027/B-DEV-500-TLS-5-1-area-anastasia.bouby.git

```
cd B-DEV-500-TLS-5-1-area-anastasia.bouby.git/Service
```

77. Install the dependencies

```
npm install
```

Starting the project:

78. To start the project, use the following command

```
npm run dev
```

Folder and file structure

API directory

This folder contains a subfolder for each API, which themselves contain:

79. {Service.ts} : The general functions related to the "area" (action or reaction) of this service.
80. {Service.query.ts} :The functions containing queries, also related to the "area" (actions or reactions) of this service.

Area directory

81. area.service.ts : The code that links actions and reactions together. The file retrieves the list of all existing AREAs (for all users), and if the actions have been executed, the associated reactions are triggered.
82. service.action.ts : For each action, we call the corresponding functions.
83. service.action.ts : For each reactions, we call the corresponding functions.

Config directory

84. db.ts : Database configuration file. Contains connection parameters and options.

DB directory

85. area directory : Contains the function that retrieves the list of all areas from the Supabase database. This function is called in area.service.ts
86. token directory :Contains the function that retrieves the list of service tokens for a given user. The function is called in the service.action.ts and service.reaction.ts files

ManageFS directory

87. manageFile.ts :Contains the functions that generate a file that stores information related to the "Weather" and "Alert" actions. The content of this file indicates whether the actions have already been triggered to prevent them from triggering in a loop (for example, if it is raining, the action would trigger continuously until it stops raining, which is not acceptable). Thanks to this file, the action only triggers once each time it rains.

Index.ts

88. Index.ts : File where the main service function is called, which initiates the area logic. This function is called at a defined interval that can be modified (currently called every minute).

Backend:

Description:

This part is the backend of the Area project, it manages users, makes calls to third-party APIs such as Google, Spotify, OpenWeather, Twitch, Github and NewsAPI, and serves data to the frontend. Authentication is handled via JSON Web Tokens (JWT) and Supabase. Swagger is used for API documentation.

The database used is Supabase

Installation:

To install and run the backend locally:

89. Prerequisites

    Before getting started, make sure you have the following installed on your machine:
90. Node.js (version 14.18+ or 16+ recommended)

91. npm (comes with Node.js) or yarn.

92. Clone the repository

```
git clone git@github.com:EpitechPromo2027/B-DEV-500-TLS-5-1-area-anastasia.bouby.git

cd B-DEV-500-TLS-5-1-area-anastasia.bouby.git/Backend
```

93. Install the dependencies

```
npm install
```

Starting the project:

The server will run by default on port 8080.

94. To start the project, use the following command

```
npm run dev
```

API Documentation

The complete API documentation with each route can be accessed via Swagger. Once the server is running, navigate to http://localhost:8080/api-docs to explore the available endpoints and their usage.

Folder and file structure

Root

95. Index.ts : : Main entry point of the application. This file initializes the server, sets up global middleware, and configures the main routes.

API directory

This directory holds integrations with external services (Discord, GitHub, Google, Spotify, Twitch). Each service has its own subfolder containing:

96. {Service}.ts :Contains the routes for interacting with the relevant API. The route called by the frontend is always the “{service}/login” route.
97. {Service}.query.ts :This file exists only if there is a need to save information in the database. It contains all requests sent to Supabase.

Test directory

This directory contains unit and integration tests for each major module in the application. The files here validate the core functionalities within the **api**, **auth**, **service**, and **users** modules.

98. api.test.ts : Tests API calls.
99. auth.test.ts : Tests authentication and security features.
100. service.test.ts : Tests business logic in the services module.
101. user.test.ts : Tests user functionalities.

Config directory

102. db.ts : Database configuration file. Contains connection parameters and options.

Middleware directory

Contains global middleware for the application.

103. auth.ts : Authentication middleware. This module verifies access tokens to protect routes requiring authentication.

Routes directory

Contains the various routes of the application, organized by feature (about, area, login, register, services, users). Each subdirectory has the following files:

104. {Route}.ts : Defines the main route logic, handling endpoint and route configuration. For example, to log in, the route “/api/login” is called. This route returns the user’s access token.
105. {Route}.query.ts : As with the APIs, these files contain queries to the Supabase database.

The various routes and their responses can be tested via Swagger: http://localhost:8080/api-docs/

Utils directory

106. test.server.ts :Utility file for initializing a test server. Useful for simulating a test environment and running integration tests.