

AREA Project
Year 2024-2025



Nexus Documentation

Web Service

The goal of this project is to create an automation platform, using actions and reactions with various APIs.

For this project, we needed to create a website and a mobile app.

In this documentation, we will tell you about the Web Service that we created to handle actions and reactions.

Behaviors

Actions

This folder contains all the behaviors of the actions for our APIs.

Reactions

This folder contains all the behaviors of the reactions for our APIs.

Managers

Select the appropriate action / reaction depending on the name of the entry.

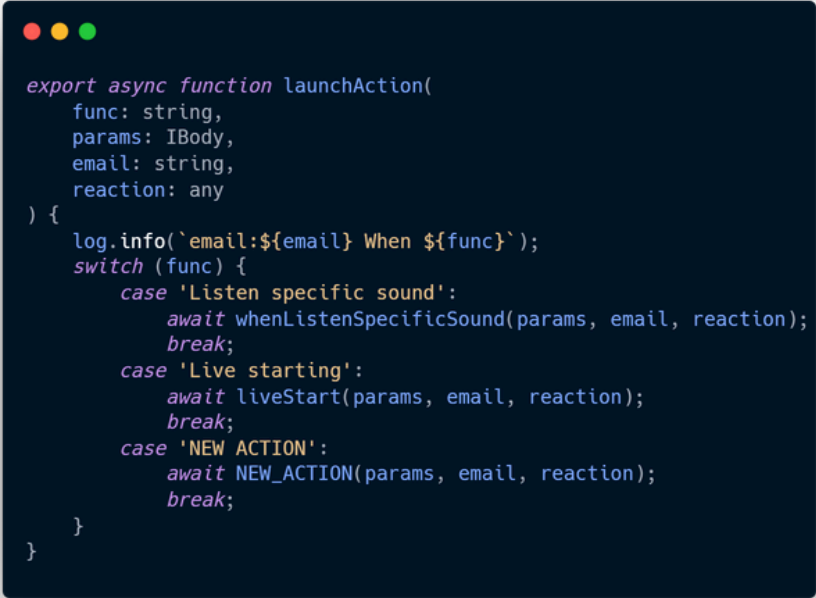
Web Service

Retrieve the user configurations and launch the actions / reactions.

Add new actions

“/ws/src/core/action.manager.ts”

To add new actions, modify the **launchAction** function



```
export async function launchAction(  
  func: string,  
  params: IBody,  
  email: string,  
  reaction: any  
) {  
  log.info(`email:${email} When ${func}`);  
  switch (func) {  
    case 'Listen specific sound':  
      await whenListenSpecificSound(params, email, reaction);  
      break;  
    case 'Live starting':  
      await liveStart(params, email, reaction);  
      break;  
    case 'NEW ACTION':  
      await NEW_ACTION(params, email, reaction);  
      break;  
  }  
}
```

The functions are inside “/ws/src/core/actions”, in the different API files.

Add new reactions

“/ws/src/core/reaction.manager.ts”

To add new reactions, modify the **launchReaction** function

A code editor window with a dark blue background and light-colored text. The code is written in TypeScript and defines an asynchronous function named `launchReaction`. The function takes four parameters: `func` (string), `params` (`IBody`), `actionParam` (`IBodySpecific[]`), and `email` (string). It checks if both `actionParam` and `params.reaction` are non-empty. If so, it iterates over each reaction and action, replacing the reaction's value with a new label that includes the action's name and value. It then logs the debug value. After the loop, it logs an info message and uses a `switch` statement to handle the `func` parameter. The cases are 'Skip to next' (calls `skipToNextMusic`), 'Skip to previous' (calls `skipToPreviousMusic`), and 'NEW REACTION' (calls `NEW_REACTION`). Each case includes a `break` statement.

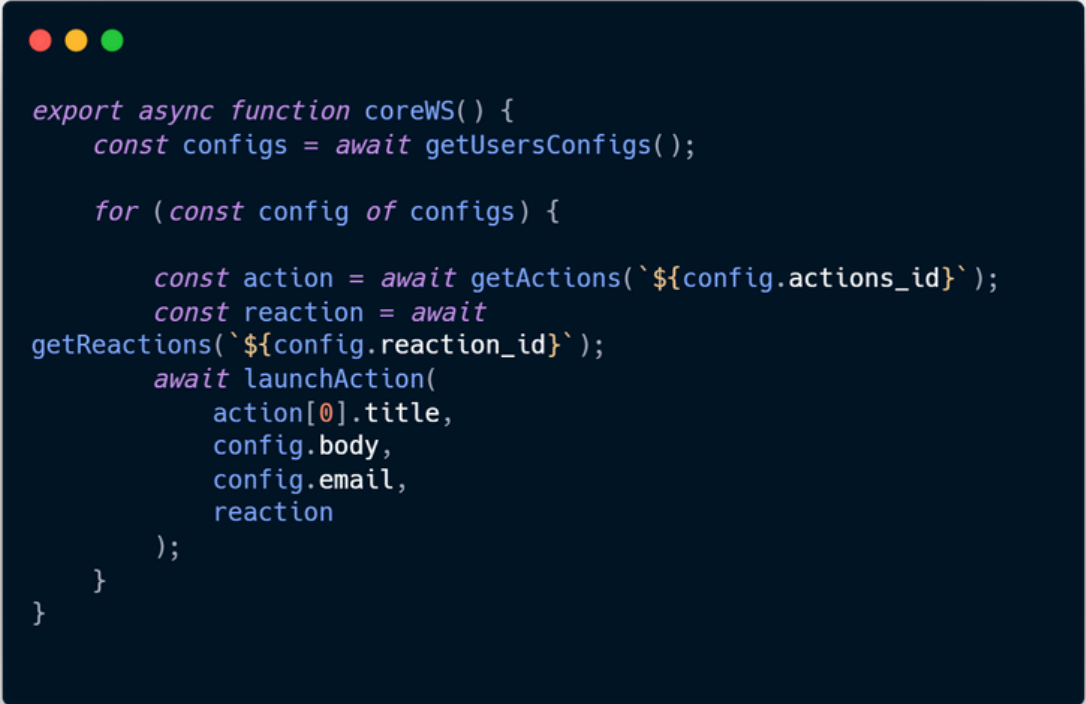
```
export async function launchReaction(  
  func: string,  
  params: IBody,  
  actionParam: IBodySpecific[],  
  email: string  
) {  
  if (actionParam.length > 0 && params.reaction.length > 0) {  
    for (const reaction of params.reaction) {  
      for (const action of actionParam) {  
        reaction.value = replaceLabel(  
          reaction.value,  
          action.name,  
          action.value  
        );  
        log.debug(reaction.value);  
      }  
    }  
  }  
  log.info(`email:${email} Reaction: ${func}`);  
  switch (func) {  
    case 'Skip to next':  
      await skipToNextMusic(email);  
      break;  
    case 'Skip to previous':  
      await skipToPreviousMusic(email);  
      break;  
    case 'NEW REACTION':  
      await NEW_REACTION(email);  
      break;  
  }  
}
```

The functions are inside “/ws/src/core/reactions”, in the different API files.

Web service

“/ws/src/core/webservice.ts”

Here's how the web service works, to launch the actions and reactions

A code editor window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in TypeScript and defines an asynchronous function named `coreWS`. It uses `await` to call `getUsersConfigs`, then iterates over the returned `configs` array. For each `config`, it calls `getActions` and `getReactions` with specific IDs, and then `launchAction` with the action details and reaction.

```
export async function coreWS() {  
  const configs = await getUsersConfigs();  
  
  for (const config of configs) {  
  
    const action = await getActions(`${config.actions_id}`);  
    const reaction = await  
getReactions(`${config.reaction_id}`);  
    await launchAction(  
      action[0].title,  
      config.body,  
      config.email,  
      reaction  
    );  
  }  
}
```