How the architecture of this project works?

This project uses the logic that gives the opportunity to keep expanding the particles emitter types without problem.

**Simple workflow example**

-You put a valid archive in the input of index.html, the system will handle and digest your json to a class in memory (fireworks.ts) | this happen in the class listener.ts

-The invalid parts of the JSON will be ignore, while the valids will keep in memory... (the system will display too if you fill a field in a invalid type). | You can check about these validation in the classes: base-type-fireworks.ts, rocket-fireworks.ts, fountain-fireworks.ts.

-Before this you can your display, the listener.ts will iterate through the list in firework.ts and run every firework who inherits base-type-firework.ts.

-Inside of each class specific of firework we have the run function, who handle specific information of each firework (rocket and fountain).

**Dictionary about each class**

**animation-holder-ts**: holds an example of animation in **phaser 3**.

**base-type-firework.ts**: hold the common information between the fireworks types.

**builder-helper.ts:** now does a if hell to check the **instace** of the firework to instantiate in memory | WIP: does a reflection who takes care about this.

**button-future-implementation.ts:** have an example how to use button in **phaser 3** and how to add listeners to this button.

**config-ts:** holds the information about the canvas (size, framerate, events).

**converter-json.ts:** convert the json in a smart object in memory.

**emitter-example.ts:** Show an example in **phaser 3** how to use the emitter.

**fireworks.ts:** Communicate between: the UI listeners, each firework and the game engine, works similar as a controller in MVC.

**fountain-fireworks.ts:** inherits the **base-type-firework.ts** and says how to run properly the animation of a fountain firework.

**index.ts:** assign our game engine controller to the view **index.html**.

**listener.ts:** handle the **oninput** of the user to show what the game engine can do, and att the screen with warnings if necessary.

**position-class.ts:** class to teach about position of the emitters if necessary.

**rocket-fireworks.ts:** inherits **base-type-fireworks.ts** and says how to run in the game engine a firework of type rocket.

**ui-handler.ts:** wraps **sweetalert2** in an easy way to use and show alerts and warning to the final user.

**velocity-class.ts:** Store information about how some classes will run through the seconds in the game enine.

**why-is-not-valid-type.ts:** Wraps some erros in the class **firework.ts**

**How to add new types of rocket**

* Add a new class who inherits the base-type-fireworks.
* Populate it with the new iterations in the function run.
* Go to game.ts and add new functions if necessary.
* Add new item into the dictionary in builder-helper.ts.
* Add a new if with the new instance in builder-helper.ts in the function buildAndReturn.