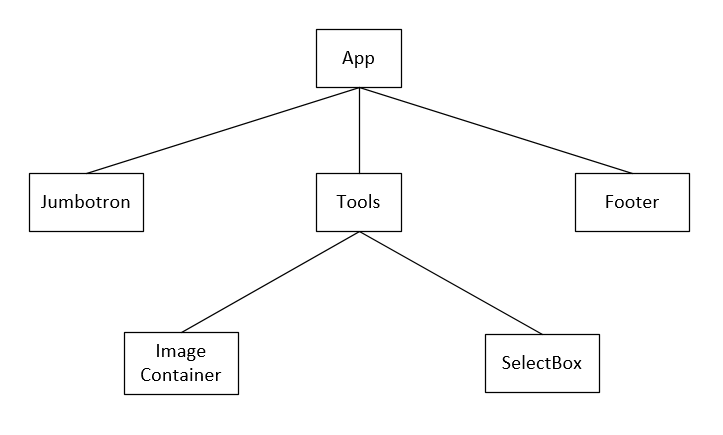
**Comp2Dust Programmers Guide**

“Comp2Dust Image Compressor” uses “compress-images” npm module to achieve its main task and which is designed around.

To explain the system, let’s start with front-end.

Comp2Dust is a single page web application that has been designed with ReactJS. Thus it is highly modifiable and good looking.

Components tree of Comp2Dust is given below:



**App:** Covers all content

**Jumbotron:** Contains Header part of the application

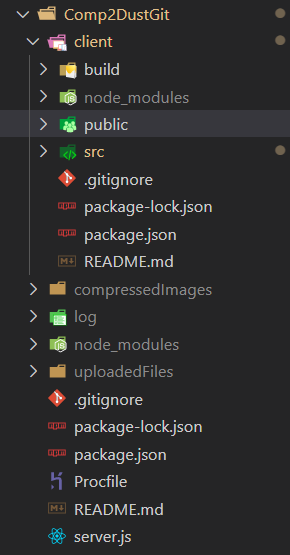
**Footer:** This is the bottom part of the application, contains nav-bar links.

**Tools:** This is the part where compression task is handled. You should focus here.

**Image Container:** The Main image displayer with comparison feature is here as an image gallery.

**SelectBox:** At this part of the application user enters commands, chooses compression engines.

**Project’s file hierarchy is given below:**

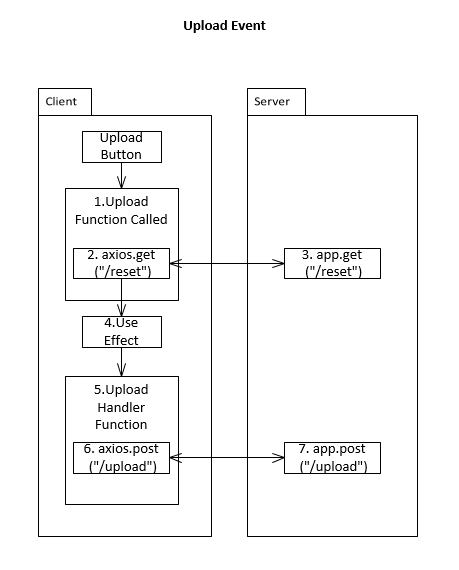


* Server and client files are separated
* Uploaded and compressed files are being kept on server because any change in client side file system will force react to refresh the page. For that reason these folders will be served as static files with express.
* “jsx” files being kept inside src folder
* “build” folder contains optimized client files for production environment

Server and client files are running on different ports, to make them work together, we define a proxy in the client package.json.

We will inspect the application’s life cycle in four event,

* On “Upload Button” clicked
* On “Compress Button” clicked
* On “Configuration File Upload Button” clicked
* On “Download Button” clicked



1. **Upload Function**

This function contains axios.get(“/reset”) function

1. **axios.get(“/reset”)**

This function send a get request to the server. Deletes any previous uploaded or processed file to start a fresh run.

1. **app.get(“/reset”)**

Server responds with a *Session ID* that has been created with **“express-session”** middleware. It is unique to every visitor of the website.

On axios promise resolve, some state hooks will be updated that will affect the react elements. But most important two change here are uploaded files assigned to a state as a **FileList** and **sessionID** of the current user. We will talk about more on how we will use sessionIDs.

1. **Use Effect**

Use effect runs when a state hooks is updated. On uploaded files assigned to a hook, use effect calls the function called “uploadHander”.

1. **Upload Handler**

This function contains axios.post(“upload”) function

1. **axios.post(“/upload”)**

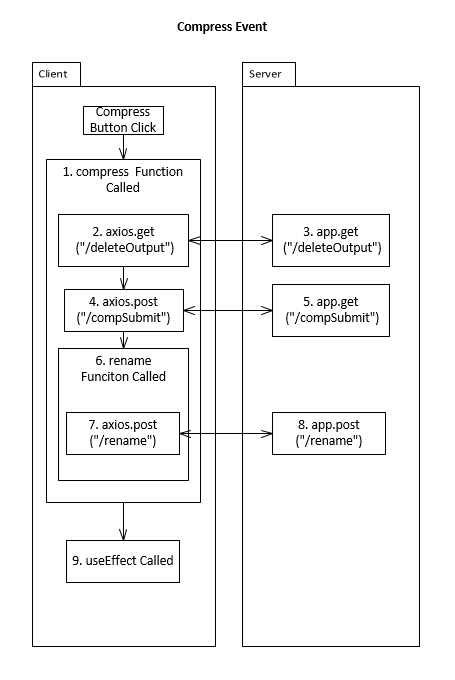
Sends uploaded files with a form data to the multer middleware. While files being uploaded. Upload progression is assigned to a state, which will animate the progress bar.

1. **app.post(“/upload”)**

Saves files to pre-determined location on the server to a folder with name of that user’s **sessionID**. That is how the application keeps files separate for users.

On axios promise resolve, input file names are assigned to a state hook. Which will be used to update image displayers.

After these steps are completed. React elements will be updated after a brief moment.



1. **Compress Function**

At the start of this function, form data taken from option box (serialized). Contains fallowing functions.

1. **axios.get(“/deleteOutput”)**

Sends a get request to server, to clean space

1. **app.get(“/deleteOutput”)**

When request being sent to server. Previously compressed files will be deleted, to free the space to that particular sessionID.

1. **axiosPost(“/compSubmit”)**

Sends **the form data** as post request to the server.

1. **app.get(“/compSubmit”)**

This is where compression function lies. Form data that has been taken from client is passed to the **processImages** function as parameters. When this function completed, compression statistics of the image will be sent back to client.

On axios promise resolve, compression statistics that has been taken will be assigned to a state.

1. **rename Function**

This function contains axios.post(“/rename)

1. **axios.post(“/rename)**

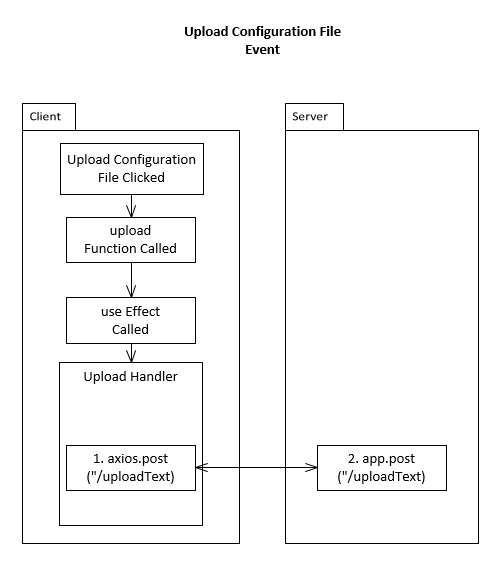
Sends comp index as a post request to the server

1. **app.post(“/rename”)**

This function renames the compressed files, attaching their name an index which will increment by one each time. Reason for this, React only updates its components when files are changed. Even the file itself changes, but its name stays same, this will not update the elements.

1. **useEffect Called**

When compression statistics assigned to a state, a useEffect is called. It changes input and output sizes to kilobytes from megabytes and calculates total space saved and assigns it to a state.



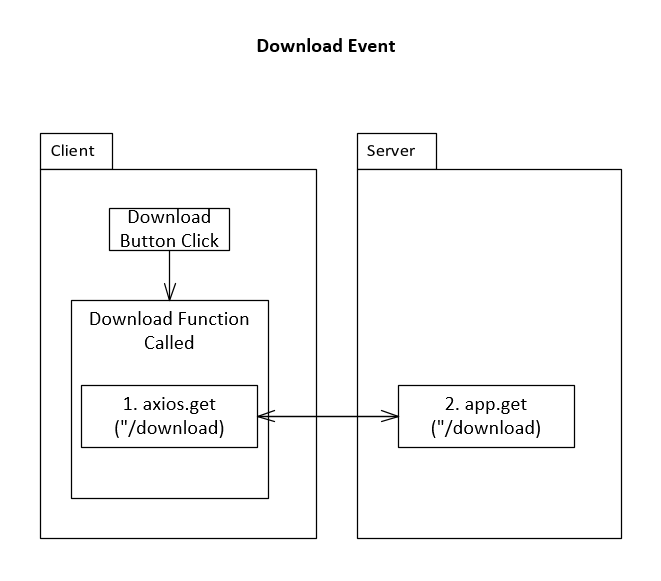
Process are the same with the previous upload section, so the basic steps are fast forwarded.

1. **axios.post(“/uploadText”)**

Sends the text input as a post request to server

1. **app.post(“/uploadText”)**

Saves the file to server



1. **axios.get (“/download”)**

Sends a request to server

1. **app.get (“/download”)**

Reads the particular folder with the according sessionID.

Zips the folder contents. Responds with a download command. However this will not

trigger the browser download.

On promise resolved, it forces browser to download by window.open(“/url”) function.

**But how do you clear files when user leaves your application?**

Application will be hosted on Heroku servers, therefore any static files will not be kept. This is the way Heroku works.