Fairchain

Handbook

Version 1.0, 06.06.2021

Blendi Shala
Jethro Warnett
Korab Bejta
Lawrence Chiang
Marco De Liso

Developed during the ‘Software Engineering Lab FS2021’ of University of Bern on behalf of Abilium GmbH.

Table of Contents

Introduction

Fairchain is a web application which allows users to interactively create and explore Production Chains and visually communicate the ethicality or sustainability of the individual inks in the chain.

The end goal of this project is to offer the application as a plugin to shopping websites and companies, so that they can ensure their customers that their products are ethical and sustainable.

The application was developed during one semester in the software engineering lab of the university of Bern on behalf of the company Abilium.

This handbook provides a manual on how to use the application and explains how the underlying code works and what its dependencies are.

Installation

Clone the repository into a folder on your computer.

Open the command prompt in the cloned repository and run the following command

```
npm install --save-dev @angular-devkit/build-angular
```

Finally run

```
npm install html-to-png
```

To start the application, run in the command prompt the command

```
ng serve
```

Next open your browser and go to the URL

```
http://localhost:4200/editor
```

You are now able to use the application and start creating your own Fairchains.

To run test cases, open a command prompt in the cloned repository and run

```
ng test
```

Manual

Layout

Upon opening the web application, the user is greeted with the homescreen (Figure 1.1). The home-screen consists of the following regions (Figure 1.2)

- Menubar: options for the application
- Toolbar: options to edit the graph
- Canvas: region to draw the graph
- Option Bar: options for nodes consisting of
    - Flags
    - Groups
    - Metadata

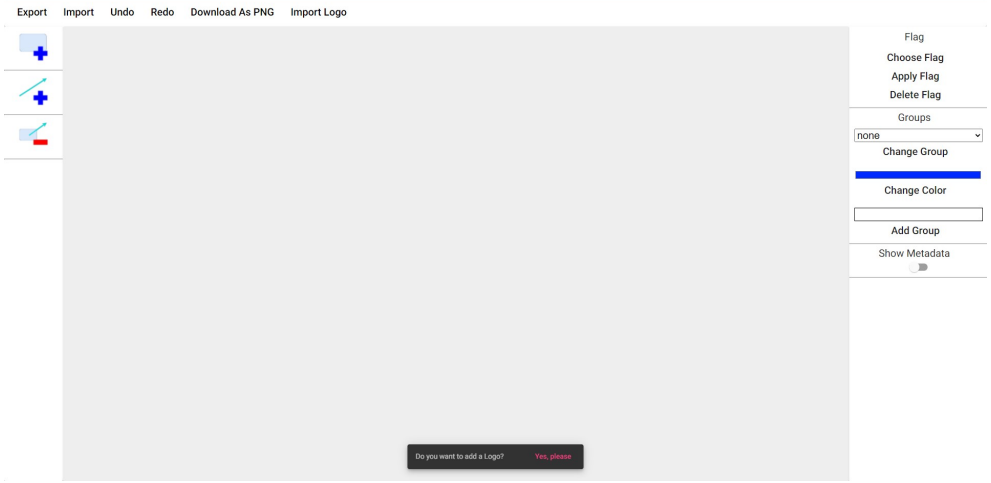

Figure 1.1: The homescreen of the application.

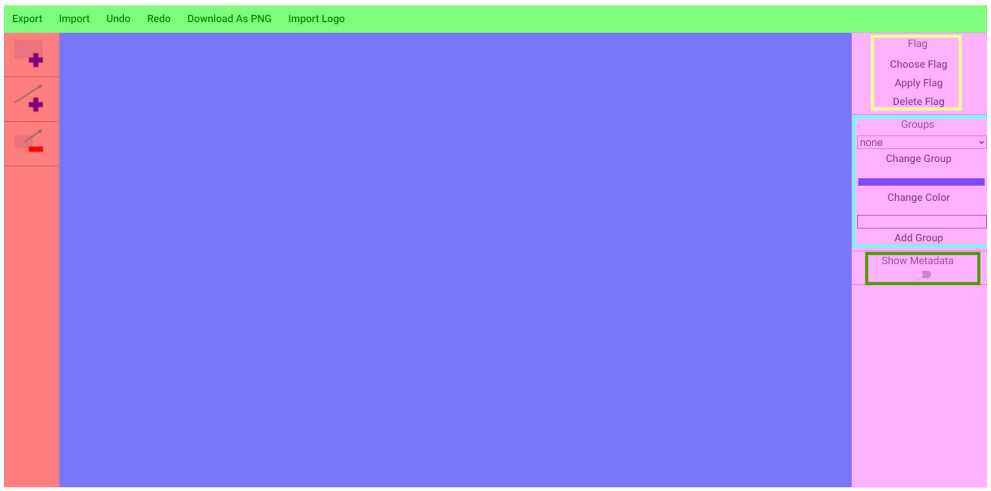

Figure 1.2: Regions of the application. Menubar (light green), Toolbar (red), Canvas (dark blue), Option Bar (pink), Flag Region (yellow), Group Region (light blue), Metadata region (dark green).

Production Chains

Production Chains (or supply chains) are a method of visually explaining how a product is created and what its ingredients are. They are usually a diagram consisting of boxes and arrows called nodes and edges. The nodes indicate the components or manufacturing processes of the item and the edges show how the components are combined.

Graph Manipulation

To add new nodes, click the add node button (Figure 1.3). You are now in the add node mode indicated by the button lighting up blue. By clicking an empty spot on the canvas, a new node will appear. Clicking the add node button toggles the add node mode.

You can add a new edge, press the add edge button (Figure 1.3). The button lights up blue to indicate that you're in the add edge mode. By holding the left mouse button down on one node and dragging it to another node, will connect both nodes by an arrow. Clicking the add edge button toggles the add edge mode.

Child nodes can be added to a node, by hovering the node and pressing the pop up that appears over the hovered node (Figure 1.4).

Elements can be deleted by first selecting the elements and then pressing the delete element button (Figure 1.3).

Holding down shift and the left mouse button creates a selection box. All edges and nodes contained in the selection box will be selected.

You can zoom in and out of the graph, by scrolling the mouse wheel on the canvas.

Whilst not in any edit mode, you can move the nodes around by holding down left click on a node and dragging the mouse.

The labels of a node or an edge can be changed by double clicking the element and writing your label in the pop up that appears over the clicked node (Figure 1.5).

Global Options

In order to export a graph, go to the menu bar and click the export option. Upon clicking the button, the browser will download a file with the format *.json*.

To import a graph, go to the menu bar and click the import button. The system navigator will open and you can select the graph you wish to import. Once the graph is selected, the system navigator will close and the specified graph is loaded onto the webpage.

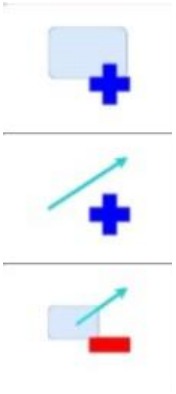

Figure 1.3: Toolbar buttons. Add Node (top), Add Edge (middle), Delete Element (bottom)

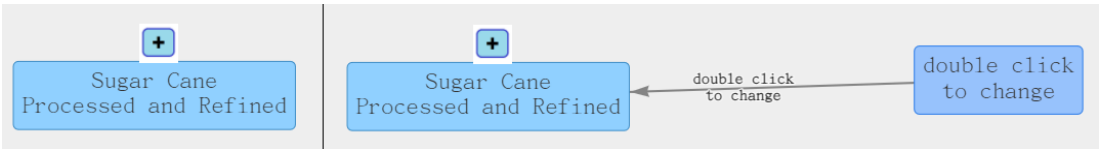

Figure 1.4: Add Node Hover Option. Before pressing left clicking (left), after left clicking (right).

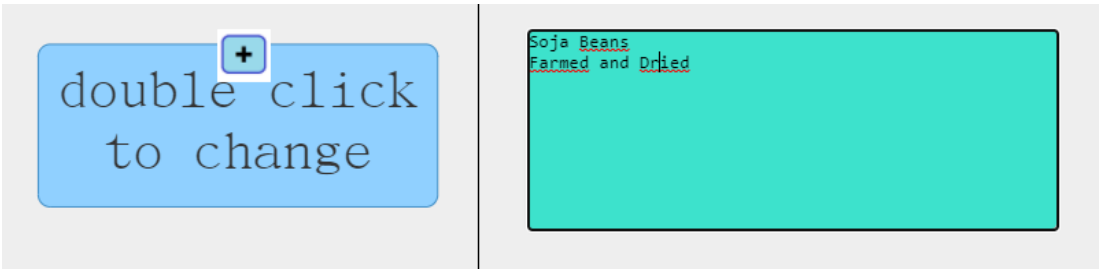

Figure 1.5: Relabel Pop Up. Before double clicking (left), after double clicking (right).

A mistake can be undone by clicking on the undo button, and also be redone by clicking redo. Both undo and redo are found in the menu bar.

The graph can be downloaded as a png image, by pressing the download as png button in the menu bar.

To import a Logo from your computer click import logo. The system navigator will open where you can select a logo, which then will appear in the canvas.

Figure 1.6: Menu bar, contains buttons with global options.

Flags

Nodes can display national flags in order to indicate where in the world a process takes place or where an element is extracted.

To select a flag to apply to your nodes, press the select flag button and choose your desired flag from the displayed flags (Figure 1.7).

To attach the selected flag to any node, press the apply flag button. You are now in the apply flag mode. When left clicking any node, the flag will be attached to the top of the node (Figure 1.8). Pressing the apply flag button, toggles the apply flag mode.

If you wish to remove a flag from a node, press the delete flag button. This activates the delete flag mode. When left clicking any node with a flag, the flag will be removed. Pressing the apply delete flag button, toggles the delete flag mode.

Figure 1.7: Selecting a flag.

Figure 1.8: A flag applied to a node.

Groups

Groups are a collection of nodes that share a common colour. These groups are used to indicate the different states of sustainability and morality in the production chain. As an example see Figure 1.9.

Figure 1.9: Nodes of different groups.

All nodes are initially not in a group and thus have a default blue colour.

To add a node to a group or change its group, first go to the dropdown and select the group (Figure 1.10). Next click the change group button. You are now in the change group mode. Left clicking any node will change the colour of the node to that of the selected group. Pressing the change group button toggles the change group mode.

When selecting a group, its colour is indicated by the colour tab. To change the group colour, press the colour tab and select the colour you want. Finally press the change colour button. The colour of the group and all nodes in the group, will change to that of the selected colour.

You can add your own custom groups, by writing the group name in the text area above the add group button and then clicking the add group button.

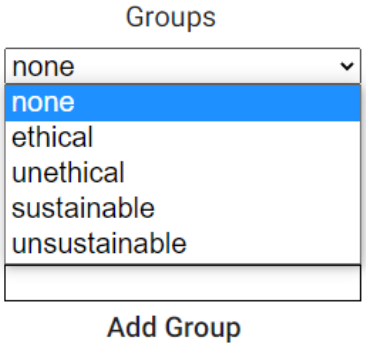

Figure 1.10: Groups that a node can be added to.

Metadata

The metadata of the graph holds additional information of the graph such as the name of the item, the company, the date etc. To display and edit the metadata, press the metadata toggle in the option bar. The information is shown as a box in the top left of the screen (Figure 1.11).

Figure 1.11: Metadata of the graph.

Code Structure and Dependencies

Code Structure

The file that drives the application is fairchain.component.ts. The component is mostly responsible for displaying the html document and to make sure that the application runs as intended. Finer details such as flags or pop up geometry are delegated to services. This is done to make the application easier to test, more cohesive and decoupled. See Figure 2.1 to see all the services used in the application and to see what their responsibilities are.

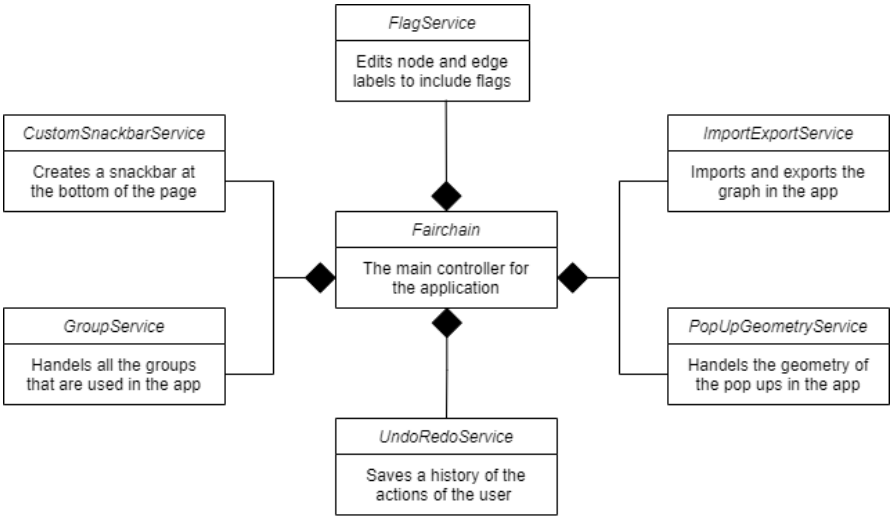

Figure 2.1: The structure of the application and the services it depends on.

The other two components that are used in the application are the NodeHoverOption component and the RelabelPopUp component. The former is displayed over a hovered node and calls a function when pressed. The latter is activated when an element is double clicked and provides a textarea in which one can relabel the element.

Important to note is that not all services and components currently have tests, such as PopUpGeometry service. This is because in a Zühlke Consultation the team was advised to not try and test the GUI aspects of the code, but rather focus on the feature implementation.

Dependencies

The application is built and driven with the powerful web framework angular.

For constructing and viewing the graph, the application is entirely dependent on the library vis.js. More specifically it uses the network aspect of the library. This library offers a powerful dynamic graph building component. Unfortunately the documentation is somewhat out of date. The team was only able to understand how the code functions, by analysing the relevant examples that the library offers.

The application also depends on the library html to png so that it can download the graph in the application to a png image.

Export Format

Overview of JSON format

The exported graph of Fairchain uses the JSON format.

A valid *.json* file must contain these four key and value pairs (Figure 3.1):

- nodes: an array of node objects
- edges: an array of edge objects
- metadata: a string containing the metadata
- groups: an array of group objects

```
{
  "nodes": [],
  "edges": [],
  "metadata": "",
  "groups": []
}
```

Figure 3.1: This json object imports an empty graph.

Json objects that do not have the keys “nodes”, “edges”,“metadata” and “groups” will not be accepted by the program. Each json object should be contained in it’s own *.json* file.
The size of the *.json* file should not exceed 10MB.

Node Objects as JSON

In this section, we will take a closer look at how node objects are structured. A node object must contain the following four attributes (Figure 3.2)

- id: a string which represents the id of the node object.
- label: a string which represents the label of the node.
- x: a number which represents the x coordinate of the node in the canvas space.
- y: a number which represents the y coordinate of the node in the canvas space.

```
{
  "id": "ad6b3a86-7ebf-48a1-52ed-b6ee4fa33cf2",
  "label": "double click\nto change",
  "x": 556,
  "y": -64
}
```

Figure 3.2: A standard node.

The app will only accept the format if each node contains these four attributes.

The nodes can be provided with additional options. Each node can be customised with the attributes (Figure 3.3)

- color: a string of the hex value of a colour.
- fixed: a boolean indicates if the node is fixed to the x or y axis that it's on.
- font: a string that of the font to be used for the label in the node.
- icon: options that are activated when the node is an icon.
- margin: a number which indicates the space that should surround the label in the node.
- group: a string of the group to which the node belongs to.
- shadow: a boolean if the node casts a shadow onto the canvas.


```
{
  "id": "841552e5-8199-4bb4-ba99-0fa06ef919b1",
  "x": 156,
  "y": -64,
  "label": "%uD83C%uDDE8%uD83C%uDDED\ndouble click\nto change",
  "color": {
    "background": "#6370ff",
    "border": "#2f3ccc",
    "highlight": {
      "background": "#828cff",
      "border": "#2f3ccc"
    },
    "hover": {
      "border": "#2f3ccc",
      "background": "#828cff"
    }
  },
  "fixed": {
    "x": false,
    "y": false
  },
  "font": {
    "bold": {},
    "boldital": {},
    "ital": {},
    "mono": {}
  },
  "icon": {},
  "imagePadding": {},
  "margin": {},
  "scaling": {
    "label": {
      "enabled": false
    }
  },
  "shadow": {
    "enabled": false
  },
  "shapeProperties": {},
  "group": "group0"
},
```


Figure 3.3: A node object that overrides global options.

The documentation of the nodes in the vis.js library displays other options that can be used.

To add a flag to the label of a node, find out the Alpha-2 Code in ISO 3166-1 of the country you want to add. Then find out the code of these two letters as emojis with escaped Unicode in

percent-encoding. After you find this code, write it at the start of the label followed by a “\n”. This way the app recognises and renders the flag (Figure 3.4).

```
"label": "%uD83C%uDDE8%uD83C%uDDED\nseee",
```

Figure 3.4: The Swiss flag in the label of a node.

If the user adds a logo to the graph, then this logo is a node in the graph. You can only use images in the logo. The node object of this logo node will have the following format (Figure 3.5)

- id: a string that represents the id of the logo node.
- label: the string of the label, default is an empty string.
- x: a number which represents the x coordinate of the node in the canvas space.
- y: a number which represents the y coordinate of the node in the canvas space.
- size: a number that represents the size of the logo node.
- shape: a string “image”. This is a necessary requirement by the vis.js library.
- image: a string that denoted the image path Base64 form.


```
{
  "id": "Logo",
  "label": "",
  "x": 112,
  "y": 145,
  "size": 50,
  "shape": "image",
  "image": ""
}
```


Figure 3.5: A logo node in json format.

Edge Objects as JSON

In this section, we will take a closer look at how edge objects are structured. These edge objects contain the following keys with string values (Figure 3.6)

- id: the id of the edge object.
- from: the id of the node object that the edge should connect from.
- to: the id of the node object that the edge should connect to.


```
{
  "from": "a63675af-bf20-45be-9f99-06a1cb359af8",
  "to": "e2acda85-4d86-451c-9b85-9dd7e8f709d0",
  "id": "8b92a0c8-e011-42b8-96aa-a6cb956d355b"
}
```


Figure 3.6: A basic edge.

In order for the app to accept the import file, all ids listed in the attributes from and to, must be contained in the array of nodes. In addition every edge must possess these three attributes for the graph to be imported.

Metadata as JSON

The metadata section of the json object contains the metadata of the project as a string. When importing the graph, the metadata is parsed and added to the graph. An empty metadata is represented by an empty string “” (Figure 3.7).

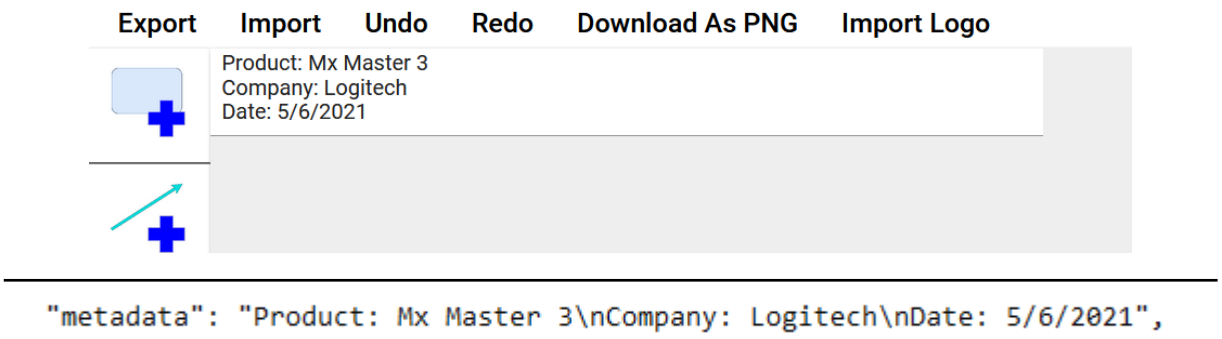

Figure 3.7: Metadata described in the JSON file. New lines are signified with a “\n”.

The metadata in the application (top), the JSON representation (bottom).

Group Objects as JSON

In this section, we will take a closer look at how group objects are structured. These group objects have the following keys with string values (Figure 3.8)

- name: the name of the group that will be shown in the drop down menu.
- color: the color of the nodes of the group.
- visJsName: the name of the group that is used internally in the app.

```
{
  "name": "ethical",
  "color": "#6370ff",
  "visJsName": "group0"
},
```

Figure 3.8: A simple group object in the JSON file.

The value of visJsName should not include special characters and should start with an alphabetical character. If you have exported from the app, the json will include the following values for “names”: ethical, unethical, sustainable, unsustainable.