

4BCT1 – 2018

**Computer Science & Information Technology**

**Final Year Project**

**Supervisor**

**Dr. Owen Molloy**

[owen.molloy@nuigalway.ie](mailto:owen.molloy@nuigalway.ie)

**Author**

**Gary McMahon**

[g.mcmahon5@nuigalway.ie](mailto:g.mcmahon5@nuigalway.ie)

**14534653**

Table of Contents

[Final Year Project 3](#_Toc510729521)

[Acknowledgements 3](#_Toc510729522)

[Introduction 4](#_Toc510729523)

[Project Overview 4](#_Toc510729524)

[Problem Statement 4](#_Toc510729525)

[Solution 4](#_Toc510729526)

[Technical Review 5](#_Toc510729527)

# Final Year Project

Welcome to my report for my final year project. The report will be split up into 5 parts with the final conclusions and results as the closing words.

* **Introduction** Introduce the project overview and how it will apply to the real world.
* **Research chapter** This chapter looks at some of the research undertook as part of the project and how it contributed to the end result.
* **Technical Review/ Research chapter** Discuss the underlying design decisions for the project UI that stemmed from examining the technologies and methods previously used in the project. Examine the current method used to display the graphs in the ResponSEAble codebase.
* **Implementation** chapter examines the code used in the creation of the project. Will look at descriptions of the technical issues in the project. Discuss features that were added to the project along with features that didn’t quite make the cut.
* **Results** Discuss how the project development, if final deliverables were met and evaluate the project.
* **Final conclusion**

# Acknowledgements

* Dr. Owen Molloy – Project supervisor.
* Caroline Brennan – A constant source of help and guidance for the project.
* Conor McCrossan – A constant source of help and guidance for the project.

# Introduction

I choose my final year project because it used technologies I was personally interested in and I wanted to gain more experience with. I also wanted the opportunity to work as part of a project of the scale of ResponSEAble.

## Project Overview

“ResponSEAble is an interactive website that supports the development of cost-effective ocean literacy in Europe. It is an ambitious, 15-partner project and is funded by Horizon 2020. ResponSEAble is mapping European marine research and knowledge to further our understanding of complex human-ocean relationships and the economic benefits that we derive from our seas and the ecosystems they support” [1]. The interactive website currently enables users to select and view a story. If the user has admin rights they can also edit existing stories or create new ones. Due to the complexity of the stories they are presented in a very complicated manner. My goal is to develop an interactive, intelligent user interface for exploring the knowledge in the graph database, both visually and using queries.

## Problem Statement

Version Control Visualisation

Querying and Displaying Knowledge from a Graph Database. Working with a PhD student here, help develop a querying and graph visualisation tool for the OrientDB graph database. The database is being used in an EU project (http://www.responseable.eu/), and will contain a large amount of information which needs to be queried and displayed in an engaging way. You will use browser-based technologies (e.g. js, jquery, d3.js, HTML5, CSS, ajax, REST) to develop an interactive, intelligent user interface for exploring the knowledge in the graph database, both visually and using queries.

## Solution

The purpose of this project to create a develop an interactive, intelligent user interface for exploring the knowledge in the graph database, both visually and using queries. The UI should have a logical flow to its structure that makes it easy to read. Users will be able to visually explore and examine specific stories. The interface should inform the user of how much evidence is supporting each node and the causal links between nodes. This allows users to learn from the stories. Users can select specific start and end nodes on the graph and retrieve the causal evidence that supports them being linked.

# Research

## Tools

The decision on what tools to use as part of the project was extremely important. It was critical to ensure that the correct tools were used to increase the speed and ease of development but at the same time preserve the developer’s knowledge of the inner workings of the application. This section gives a better understanding of the tools used in the development process.

**NetBeans Ide**

Used for the coding/developing the UI. Used NetBeans in conjunction with GlassFish server to run application. This editor supports all the necessary technologies and languages required for the project. It also allows for extensions to the editor though the use of plugins. NetBeans allows for easy and readable presentation of code and the decision to use it over other IDE’s was down to personal preference.

**GitHub**

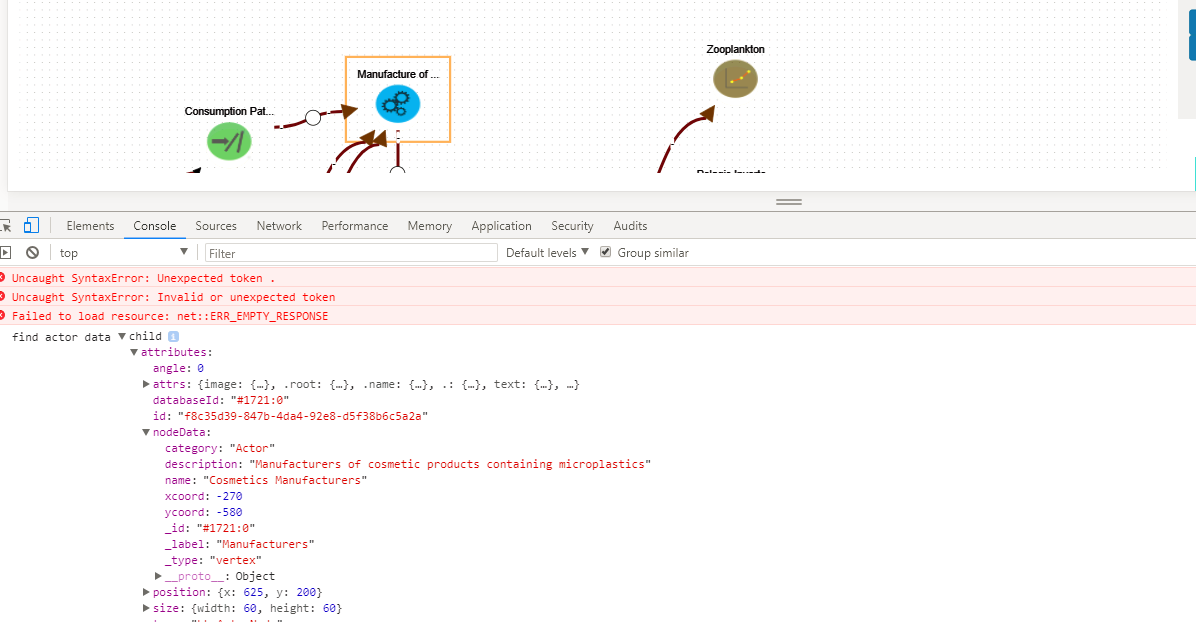
Used for version control and source management.

**Google Drive**

Secondary source of code backup in the possible event of breakdown of device that could result in the loss of code.

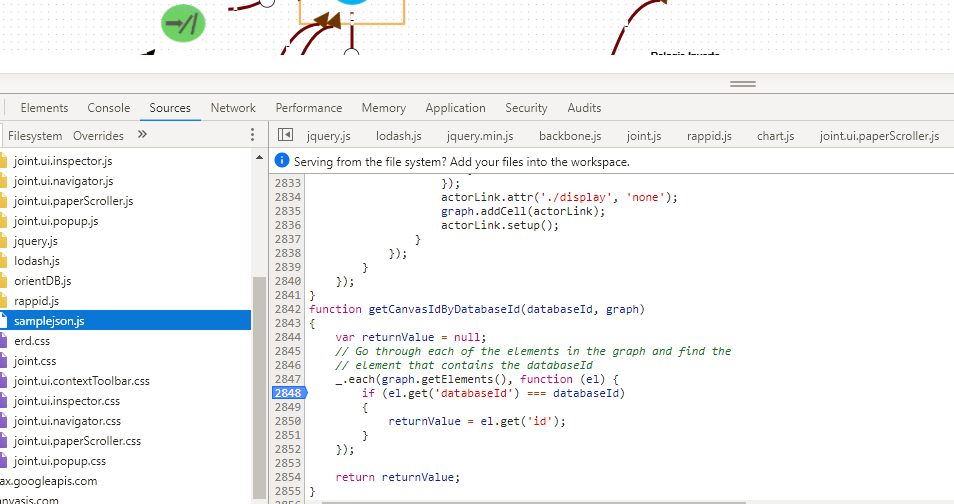
**Chrome and Dev Tools**

One of the most important tools used in the project development was Chrome and its Dev Tools. Their performance, feature set and usability is unapparelled. It’s the first point of call when an error occurs and it makes the sometimes-difficult process of debugging JavaScript that much easier. The time and effort saved by using the console tab to view simple things like the current structure of JSON data was invaluable.



*Example of examining the data contained in a node when it is rendered on the graph. Helped determine the path to access specific data points e.g. to access nodeData*

Another key feature of Chrome and its developer Tools was the ability to use the source tab to debug code in real time as you performed click events on the application.



*Example of debugging a component of the application within the projects source code with Chrome Dev Tools.*

## Languages

**HTML[https://www.w3schools.com/html/default.asp]**

HTML or Hypertext Markup Language is used to tell the browser how to display a web pages’ data to a user. HTML is supported across all major browsers which makes it the ideal language to display our content with. HTML was created to describe the contents of a page.

**JavaScript[https://www.w3schools.com/js/default.asp]**

Due to the non-static data that will be displayed on our web pages we need to use JavaScript in conjunction with HTML pages which allows for the creation of a dynamic and interactive application. JavaScript is stored in separate JavaScript files to make code readability easier. JavaScript is supported across all modern browsers.

**CSS[https://www.w3schools.com/css/default.asp]**

Cascading Style Sheets (CSS) describes how HTML elements are to be displayed on screen, paper, or in other media. CSS can be used to control the layout of multiple web pages all at once. CSS was introduced when tags such as <font> and colour attributes were added to HTML as a means to create a single source of styling instead of having to adding styling to every single page.

## Libraries/Frameworks

**JointJS[https://www.jointjs.com/]**

JointJS is JavaScript library that can be used to dynamically create static or fully interactive graphs and diagrams from JSON data retrieved using Ajax. JointJS was the main language used in the creation of the user interface. JointJS allowed for the creation of custom shapes via SVG and the rendering of potentially hundreds of elements and links with instant interaction. Zooming, animations and touch support are just some of the features of JointJS.

**SVG[https://www.w3schools.com/html/html5\_svg.asp]**

Scalable Vector Graphics (SVG) allows graphics objects, images and text to be rendered. SVG is XML based which means that every element is available within the SVG DOM and can attach JavaScript handlers to each element. In SVG, each drawn shape is remembered as an object and as a result if the attributes of an object is changed, the browser can automatically re-render the shape. SVG makes use of CSS for styling and JavaScript for scripting. The SVG library is included as standard in the JointJS Library.

**Chart.js**[http://tobiasahlin.com/blog/introduction-to-chartjs/]

Chart.js is a JavaScript open-source library that helps you easily visualize data using JavaScript. It supports 8 different types of chart and they’re all responsive. Chart.js is used to summarise the causal data between two nodes in the form of a bar chart.

**JQuery[https://www.w3schools.com/jquery/jquery\_intro.asp]**

JQuery is a free open-source JavaScript library whose purpose is to simplify the use of JavaScript on your website. The Jquery libray contains the following features.

* HTML/DOM manipulation
* CSS manipulation
* HTML event methods
* Effects and animations
* AJAX

# Implementation

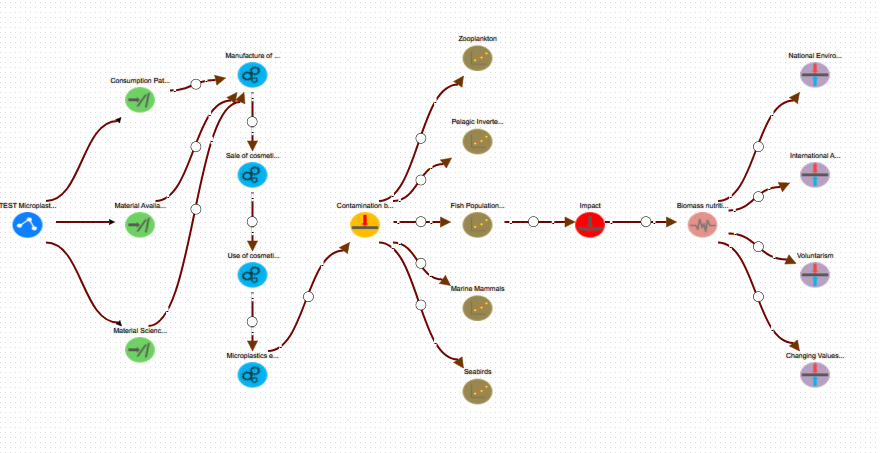
## Display nodes and edges

The first problem to be address was to create a method of displaying all nodes in a pre-determined order. In the current method of displaying stories the layout of the story depends entirely on how it was created and as a result if the graph was created with little or no order it makes it very difficult to read. In the Project Definition Document, a Paper Prototype was created for the user interface for displaying a story. The key goal was to assign each Node type its own column. This was to help preserve the flow of the graph and make it more readable. It also takes the pressure off the creator of the graph, they don’t need to worry about creating the graph in a neat and readable format.



*Paper Porotype created in the original Project Definition Document. Each node type has its own column.*

When designing the order in which we expect the nodes to follow we operate under the assumption that graphs for the most part will follow the DAPSIWR (Driver 🡪 Activity 🡪 Pressure 🡪 State 🡪 Impact 🡪 Welfare 🡪 Response) format. This means that all links will follow a logical flow from one vertex type to the next. The image we see below looks visually appealing but its success lies in how the graph itself was originally built. There remains a responsibility on the creator of the graph to try preserve the flow and layout.



*Here we see that each node type has been assigned to a specific column. The readability of the graph has been improved dramatically. Note that all knowledge, actors and backwards links have been removed to further improve the readability of the graph.*

**How we achieved this**

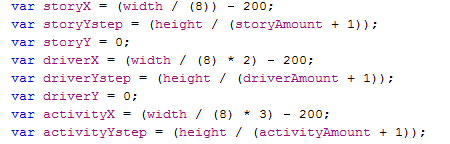
**Determine X and Y co-ordinates**

When the story is initially loaded from the DB using a pre-defined ajax call we cycle through the data to determine how much of each type of node is present. We cycle only through the vertex data as this reduces complexity.



*Code that cycles through the node of a graph to determine how much of each type is present*

The reason we do this is to allow us to determine how large our paper needs to be. As we can see from figure # nodes of type State are the most populous, from this we can then determine the max width of our paper. We want each node to have sufficient space to potentially display knowledge and not appear cluttered.



X and Y co-ordinates are assigned based on the width and height of the paper. We determine a step i.e. the vertical distance between each node type based on the number of that type of node and the overall height of the paper e.g. if there are 3 or one type of node and the paper height is 1000px then the node step will be 250. This means that each node will be placed with y-co-ordinate of 250,500 and 750 respectively. This means the nodes have the maximum space between each other possible.

**Draw Node**

For each category of node, we draw a custom JointJS shape. Each node type has the same underlying structure but with different attributes added upon creation.



As you can see each time a node of specific type is created the y co-ordinate is incremented by the amount specified by the step value. The x co-ordinate remains the same as we want all nodes of the same type to appear in a vertical line. Each node has the shortened text value assigned to it that gives a brief description to the property of the node. The text was shortened to 15 characters as it helped declutter the graph.

As you can see as the node is created new attributes are added to it. These are the databaseId, nodeData, type, xcoord and the ycoord.

**databaseId –** This is the unique identifier that orientDB maintains to distinguish between nodes. This attribute is extremely important when we look to add edges to the graph as we require a method of determining what edges connect to what nodes. By adding the databaseId attribute here we can make is easily accessible later. We will talk more on this later.

**NodeData –** To this attribute we assign all data that is received for each node to that node on its creation. By doing this we have access to all node data in a singular place that is accessible within the DOM. Because of this it is very easy access node data at a later point. Without adding this attribute, we would not have all node data easily accessible at a singular location and would make the building of dynamic lists very difficult.

**Type –** Type is used an identifier to distinguish between the different type of nodes created e.g. knowledge, actors, Storys etc. This is helpful later when we want to examine nodes in a graph and we know we don’t need to consider knowledge nodes, by having the type attribute defined we can ignore these nodes in our computations and saving resources.

**X and Y coords –** As each node is drawn we record it x and y position. This is helpful later when we want to position knowledge and actor nodes relevant to each node. It again allows easy access to this data.