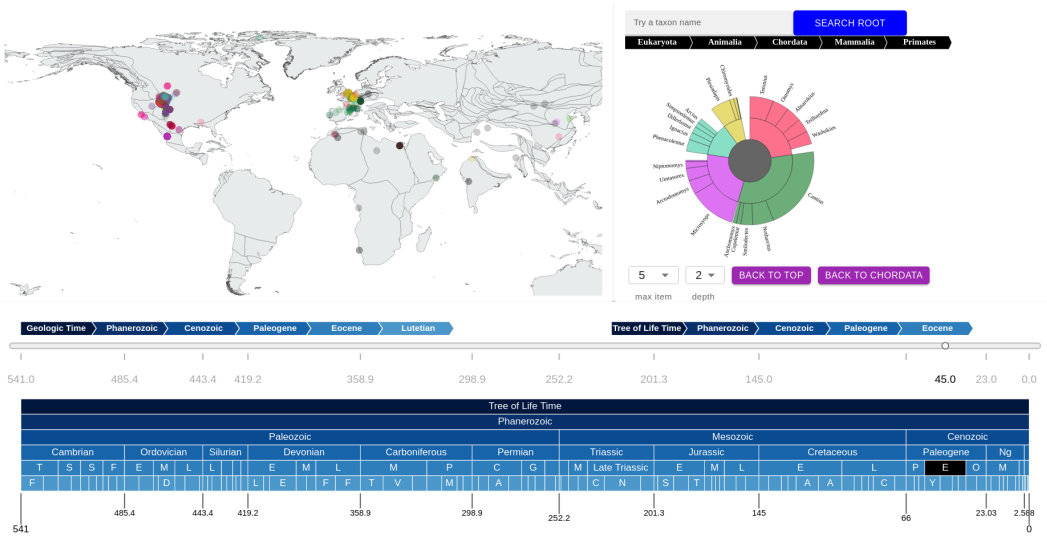
A visualization of biodiversity through time



Master Thesis
January 25, 2022

by Wuxing Dong, 18-946-038

Supervisors:
Prof. Dr. Renato Pajarola
Gaudenz Halter

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

Abstract

Contents

1 Introduction

1.1 Effective visualization of data

In the time of the information age, the Internet has become a primary source to inform ourselves about the world. It is therefore a great opportunity for the scientific community to take advantage of the web and create high quality products to communicate their findings and ideas to the general public. Textbook-styled verbal explanations can be rigorous, however, they easily become tedious and are already made available by school and college education. Videos are based on visual learning and are quick to absorb, yet there is little engagement with the viewers. Interactive data visualizations provide an excellent way for users to understand scientific concepts. New knowledge is conveyed by direct visual impact that easily draws attention, and insights can be formed by actively exploring the data and observing the response of the presentation due to the users' own actions. This learn-by-doing strategy is considered to be beneficial for consolidating new knowledge [Sch95].

Although graphics plays a major role to deliver scientific messages, there is a lack of formal theory on producing good graphics, and domain experts tend to rely on practical experiences that are guided by certain design principles [CHU07]. Increasing amount of studies have focused on concluding such principles [SI90, EM13, Mid20]. Chen and colleagues [CHU07] have summarized that three main aspects to pay attention to in a visualization project are its contents, the relevance in the context of the larger whole as well as the forms and aesthetics (the construction). Similarly, McCandless argued that a good visualization requires integrity of information, interestingness of story (concept), usefulness of goal and beauty of the visual form.

Before embarking on producing a visualization, one should note the characteristics of human visual perception to improve the design strategies in practice. Cognitive psychological studies aimed at examining human visual attention by exposing users to certain visual processing tasks. The results of those studies suggested some guidelines for more effective visualization. When the visual features of a certain type of data is expected or previously known, the accuracy and speed of visual perception is drastically improved [HW12]. Therefore, the authors suggested that a visual feature should be assigned to the same data dimension, or type of data [HW12]. It is also important to note that attention is limited and selective, therefore the amount and diversity of information should be kept within the limits of cognitive load [Alh18]. When many categories of data are present, compromises are needed to retain graphics complexity and only demonstrate a few number of categories at a time.

Through early years of research on how images are analyzed visually, it has been found that detailed vision is only achievable by actively focusing on small portion of the visual field [IK01, NS71]. However, a few visual features, such as colors and curvatures, are subjected to pre-attentive processing, which can be effortlessly and rapidly detected by low-level visual processes [HE11]. Therefore, appropriate use of limited but distinctive colors and simple shapes can be a powerful visualization technique. They are also the leading factors to make a visualization memorable [BVB+13].

With respect to the design process, it should be guided by the core messages to deliver. It is important to first prioritize what information to show, envision the final result, and then choose appropriate colors and geometries that fulfill the objectives [Mid20]. The most efficient geometry should be of high data-ink ratio, which is the ratio between the amount of ink allocated on the actual data to the total amount of ink a figure consumes [TGM83]. While striving for a high data-ink ratio, it is also important for a visualization to be simple, self-explanatory and straight to the points, such that the viewers are not overwhelmed with too much information.

The topic this thesis chooses to visualize is biodiversity through earth history. There are biodiversity visualizations that are tree-based [BHP+12], map-based [JNZ+16], or plot-based [LMD+19]. However, there lacks a rich and interactive visualization on the Internet that illustrates biodiversity with both a dynamic tree of life and a map that locates species on the planet throughout millions of years. This thesis is intended to fill this gap. Further more, it is important to raise the awareness of the general public on ecological issues, in light of the current age of climate change. By exploring how different forms of life came to exist, thrive and perish through a simple and interesting visualization, viewers could appreciate the inseparable relationship between life and its environment.

1.2 Background on biodiversity and tectonic change

Biodiversity is a result of evolution. Current views on species formation is based on modern evolution theory [Bow89] originated from Darwin's work. To summarize briefly, all forms of life are argued to share a common ancestor, which diversified over the course of evolution to form the current tree of life. An organism's genetic information is encoded in DNA or RNA, which is replicated over generations. However, recombination and random mutation events occur during the replication process, resulting in genetic variations, which in turn causes difference in phenotype, *i.e.*, the observable traits of organisms. Individuals with traits that are more suitable to their particular environment are more likely to survive and pass their genes to their offspring, comparing to those that are poorly adapted to the environment. This process is termed natural selection. Finally, a new species is formed during the process of speciation. It occurs when enough genetic variations are accumulated between the original population and the descendant population, such that organisms from the two groups are not capable of producing fertile offspring, namely, the two groups have formed reproductive isolation. A classic example of evolution is the adaptive radiation of finches on the Galapagos islands [Lac83]. The beak shapes of the finch species on different islands diversified to specialize on obtaining the dominant type of food on those islands.

It is believed that life emerged from 3.5 billion years ago [AS98]. The past 540 million years ago (the Phanerozoic Eon) sees rapid development and diversification of life. Throughout evolutionary history, there are records of drastic changes in biodiversity occurring in a relatively short period of time. On one hand, radiations are rapid increase in biodiversity. An example is the Cambrian explosion, during which most phylums of multi-cellular organisms originated [AMB+01]. On the other hand, biodiversity experiences rapid declines during mass extinction periods. There is evidence of five mass extinction periods, one of which occurred in Cretaceous and Tertiary era, when the dinosaurs were exterminated and mammals thrived due to their selective advantage given by their fur in the cooling environments [Cou02]. These periods of radiations or extinctions are caused by special events, plate motions, as well as change in climate, such as temperature and oxygen level.

Apart from biodiversity, the shape of the world also changes significantly throughout billions of years [Wil63]. The earth crust is composed of a number of tectonic plates undergoing constant motions. The heat generated from the interior of the planet causes the movement of the plates. The plate motion results in new landscapes and new continents, affecting the lives inhabiting on them. For instance, the current continents are thought to belong to one super continent called Pangaea [SHV+13], which gradually assembled at approximately 335 million years ago (Paleozoic-Mesozoic era) and drifted apart at around 175 million years ago (Triassic-Jurassic era).

As the continents separate, populations from the same species are geographically isolated into their particular environments resulted from the plate motion. Since organisms are under constant selective pressure, new species are formed and evolved in the newly created environments. Numerous studies support that tectonic movement plays an important role in driving the wax and wane of biodiversity at various regions of the world. To list a few examples, Miocene tectonics contributed the diversification of ungulate and carnivores in western United States [KF08]; the formation and break-up of Pangaea has driven the global marine invertebrates diversity for the past 443 million years [ZFP17], and the beginnings and movements of tropical reef biodiversity hotspots can be predicted by modeling plate motion for the past 140 million years [LDG+16].

1.3 Aim of the thesis

This thesis aims at building a web application that visualizes the change in biodiversity through the course of earth history. In the final product, users can view a customizable tree of life during a selected time period under a selected taxonomy. In the meantime, the fossil records associated to the displayed tree of life is shown on a map of the selected epoch.

1.4 Thesis structure

Chapter 1 introduces the topic of data visualization and domain specific background information, and it claims the goal of the thesis. Chapter 2 analyzes some related work, indicating what could be learned and improved for the visualization to be built. Chapter 3 outlines the problems to be solved at each stage for building the web application. Chapter 4 discuses the technical solutions for data processing and web development stage, respectively. In chapter 5, the implementation methods for data processing, backend and frontend are listed. Chapter 6 reports the result of the product. Chapter 7 concludes the project and suggests future work.

2 Related work

There already exists an abundance of visualizations on the Internet in regarding to tree of life as well as plate motion. This section discusses some selected works on these two topics.

2.1 Tree-based visualizations on biodiversity

OneZoom (https://www.onezoom.org/, Figure 2.1) visualizes all known living things in a fractal tree, and it is targeted for both professionals and the masses [RH12]. Users can zoom into or out of different levels of the tree by scrolling, and can search for a particular node by its name.

(a) The overview of OneZoom's tree from root.

(b) The view on OneZoom's tree when zoomed onto mammals.

Figure 2.1: Tree of life implementation in OneZoom.

The fractal design gives the tree a nice appearance, with high resemblance to an actual tree. Information such as images or earliest time of a node is also well packed in the interactive image. There is a reset button to return to the root, and a drop down list that provides links to the nodes along the path to root from the current position. These functions are convenient for navigation and similar functionality should be implemented in this thesis. It is claimed that all categorized organisms, which has a total amount of over two million, are included in the tree. Each species in the tree structure are incorporated with a phylogenetic classification. This leads to the fact that when users attempt to visit the root from a given species on a leaf node, they visit all of the nodes representing each of its ancestor throughout the evolutionary history. Although how a certain organism evolved from the common ancestor of all life can be rigorously inspected, from the user experience's point of view, the tree is extremely deep. Apart from manually searching for a name, navigating to a desired location in this deep tree is mainly achieved by scrolling and dragging the item of interest to the center of the screen. This makes it difficult for viewers to relocate to a different section of the tree, or to gain an overview of where the current node positions in the entire tree of life. To reach the root of all life (Figure 2.1(a)) from mammal (Figure 2.1(b)), users need to scroll through all upper levels of the tree, which have a similar-looking spiral structure that appears to be confusing and disorienting. A similar tree of life visualization is Lifemap (http://lifemap.univ-lyon1.fr/explore.html, [dV16]), which ameliorates the navigation difficulty by highlighting the ancestry path from root to a node (Figure 2.2). However, due to the sheer depth of the tree, users still need to travel deep into the tree by scrolling. Rather than demonstrating the exact ancestral paths, the purpose of the tree for this project is to organize each species

such that biodiversity can be viewed in an intuitive way. As a consequence, all nodes can be organized by the main standard taxonomic ranks, in the descending order of domain, kingdom, phylum, class, order, family, genus and species. This limits the tree's depth to be eight and thus simplifies the navigation process. In addition, to avoid scrolling and dragging, the tree can take a fixed position. Users can select a node of interest as a root, and only a part of the complete tree rooted by the chosen node is shown. In the meanwhile, the location of the selected node in relation to the complete tree of life can be shown in a clickable breadcrumb structure that 1, provides an overview of the whole tree by demonstrating the path from the root of all life to the current root and 2, allows users to easily navigate to any of the upstream node in the tree by clicking.

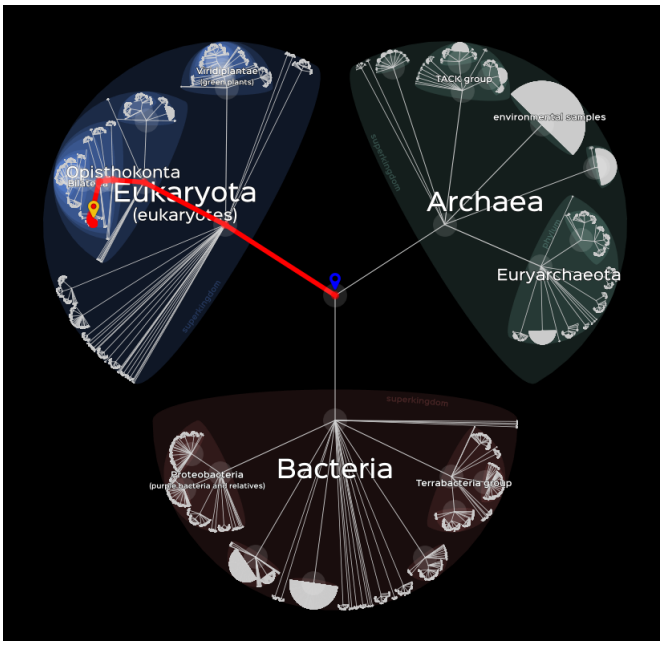
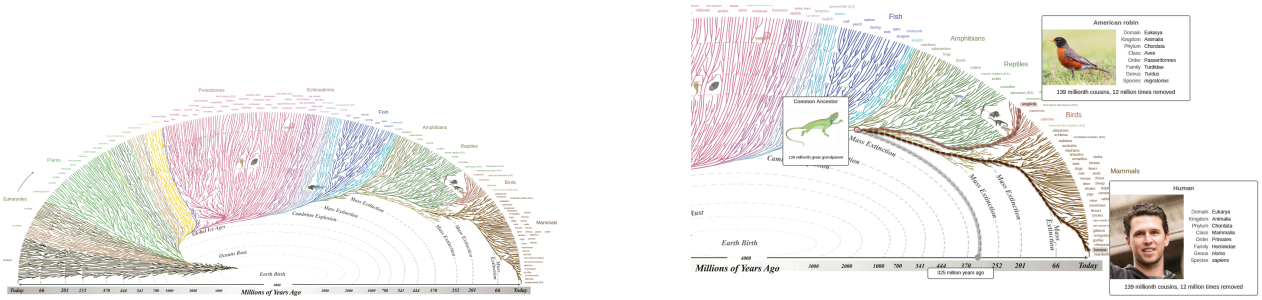


Figure 2.2: Lifemap enables a "path from root" functionality that highlights the ancestry path from *Homo sapiens* to the root.

A much simplified version of tree of life is Evogeneao (https://www.evogeneao.com/en/explore/tree-of-life-explorer, no publication related), which is intended for school education. As shown in Figure 2.3, all species are clustered into clearly colored regions, giving the viewers a direct and memorable visual impact. A similar coloring and grouping design could be adapted in this project. With the tree of life arranged in a half circle that is labeled with time and a few iconic events, users could get a view of how life radiated from the common ancestor into great diversity, and how this process is affected by special events on the planet. Since only a small subset of all organisms are shown, the tree is small and can be entirely shown, thus always giving the audience an overview. However, this web application provides little possibility of exploration due to its limited options for interaction, as the main functionality is to show the common ancestor of two selected items (Figure 2.3(b)).

Finally, there are a wide variety of tree of life visualized for computational biologists, such as AnnoTree [MCP+19], phylogeny.IO [JM19], T-BAS [CWM+17] and PhyloWidget [JP08], to list a few. These visualizations provide rich functionality for bioinformatic analysis that are well beyond the need of general public and often take efforts to use. A typical example is iTOL (https://itol.embl.de/itol.cgi, Figure 2.4, [LB19]). The design of arranging all leaf nodes on the edge of a circle and all non-leaf nodes in its interior utilizes space effectively, therefore providing a good hint for our implementation.



(a) The overview of Evogeneao's tree.

(b) The ancestral relationship between human and songbird.

Figure 2.3: Tree of life implementation in Evogeneao.

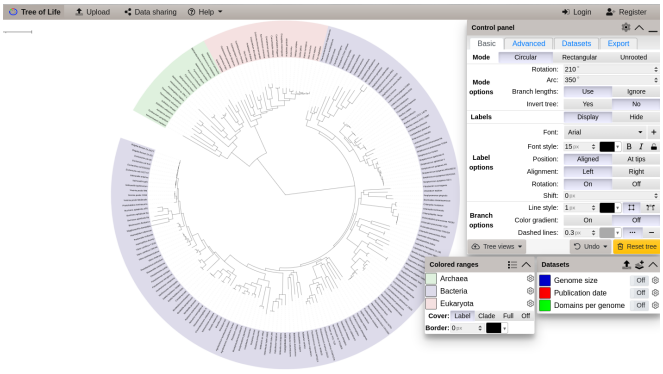
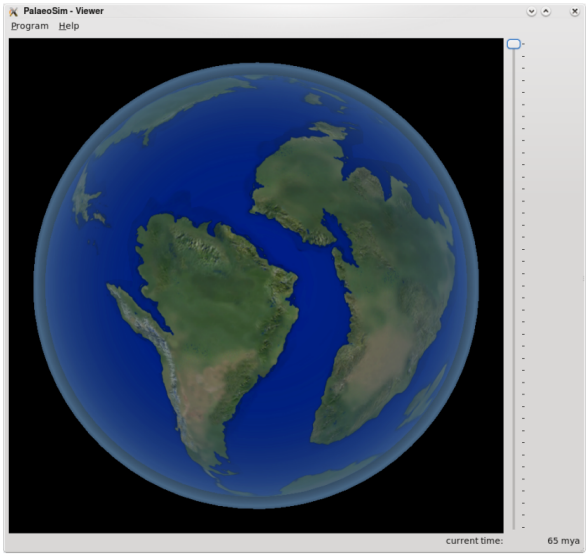


Figure 2.4: Tree of life in iTOL.

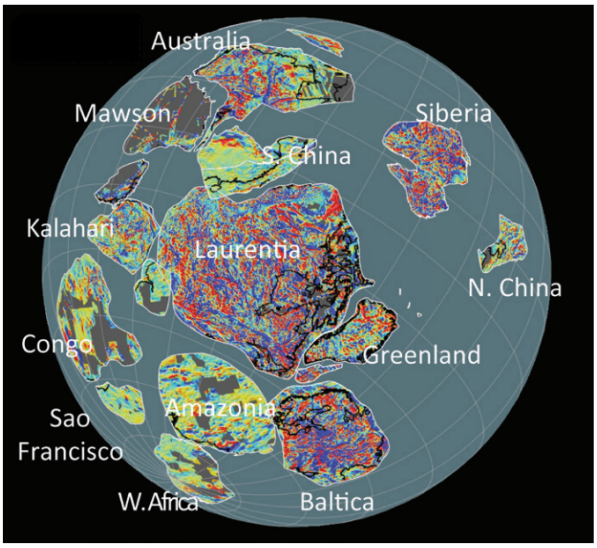
2.2 Visualizations on tectonic shift

Current project involves visualizing paleogeography at various time points, since biodiversity is displayed as fossil points on a world map during the select time period of interest. Rather than web-based, most visualizations on tectonic changes are commercial software products that are primarily intended for inspecting global petroleum and ore reserves from a paleogeographic perspective. Many products targeting the scientific community falls into the category of GIS software [SWW10]. They are designed for analytic purposes and not necessarily intuitive to operate or visually pleasing, such as GMAP [TS99], PaleoMac [Cog03], PLACA [MOAF05] and PPlates [SKR+07].

In more recent years, PaleoSim (Figure 2.5(a), [RLM10]) visualizes the ancient earth by proposing a tectonic morphing model that simulates a smooth transformation of plates through time from image data. Users can select a time point or a particular plate and observe its shape throughout ancient history. However, this offers limited functionality, as it is not capable of tracing the location change of a given point on the map, which is needed for visualizing fossils. However, GPlates (Figure 2.5(b)) is a next-generation software widely used for plate-tectonics purposes [BMG+11] [WMLW12] [GTZ+12]. In short, the software receives a tectonic simulation model that defines how the movement of plates occur through time, and vector data that represents geo-spatial information in points, lines or polygons. It can then assign all geo-spatial data to a plate-tectonic reference, before reconstructing the geography at a requested time point, and exporting the reconstruction as files. In relation to our project, it provides a method to trace the fossil points by assigning it to a certain plate, such that how its position alters can be calculated with a given tectonic model. As a result, this tool is used for our project, to reconstruct both the world map and the fossil points at a given time point in earth history.



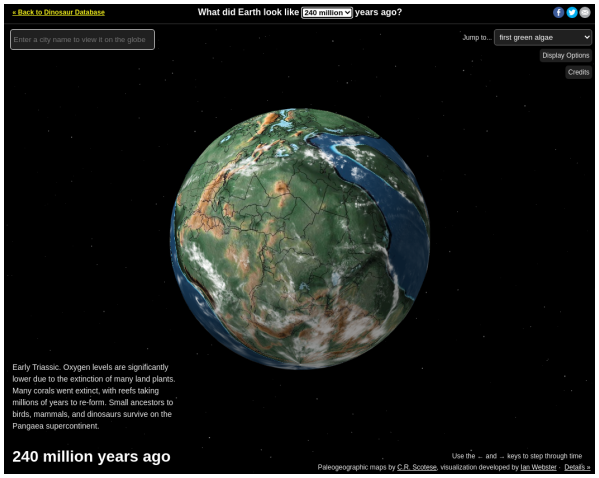
(a) PaeloSim visualization on South America and Africa 105 million years ago. The figure is directly adopted from [RLM10].



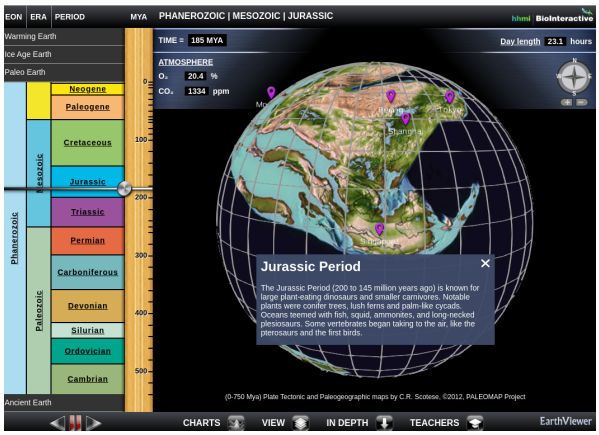
(b) GPlates visualization of ancient earth at 710 million years ago, based on a model by [LBC+08]. The figure is directly adopted from [WMLW12].

Figure 2.5: Example software-based visualizations on tectonic shift.

A few web-based projects on continental drift visualization can also be found. Two typical products are Ancient Earth (https://dinosaurpictures.org/ancient-earth#240, Figure 2.6(a), no publication) and Earth Viewer (https://www.biointeractive.org/classroom-resources/earthviewer, Figure 2.6(b), no publication). Both of them show the coastlines of the continents at a select time point on a three-dimensional globe. Ancient Earth allows users to pick a particular time point from a list of options and search for the ancient location of a city. Earth Viewer provides more detailed information on geological time, and selectable time points are incremented at every five million years on the scroll bar. Time periods are organized in a hierarchical structure, which can be adapted in this project for easy navigation. When clicking on a time period, instead of directing the user to view the time period, a pop-up window containing a brief introduction of the time period appears, which is counter-intuitive. An improvement is to show the earth of the selected time period upon clicking.



(a) Tectonic shift visualized in Ancient Earth.



(b) Tectonic shift visualized in Earth Viewer.

Figure 2.6: Example web-based visualizations on tectonic shift.

3 Problem statement

The data visualization project is divided into several sub-problems as follows.

- **Data processing**
  - Process the PBDB dataset to extract fossil records
  - Extract a tree of life from the taxonomy information of the fossil record
  - Trace the coordinates of each fossil record to ancient time
- **Database setup**: store the fossil data, coordinate data and the taxonomy data into a database
- **Back-end building**: enable data queries from the database upon users' request from the front-end
- **Front-end building**: visualize interactive components for users to explore biodiversity through deep time
  - A time component that lists ancient epochs that users can select
  - A tree component that shows the tree of life constructed by the fossil records found in the selected time period, under the selected taxonomy.
  - A map component that shows the shape of the continents in the selected time period, and the locations of the fossil records that constitutes the tree of life in the above component.

4 Technical solution

4.1 Overview

This section explains the technical solutions to the problems stated in chapter 3. To summarize, the project is divided into data processing in section 4.2, database setup in section 4.3, back-end query processing in section 4.4 and front-end development in section 4.5.

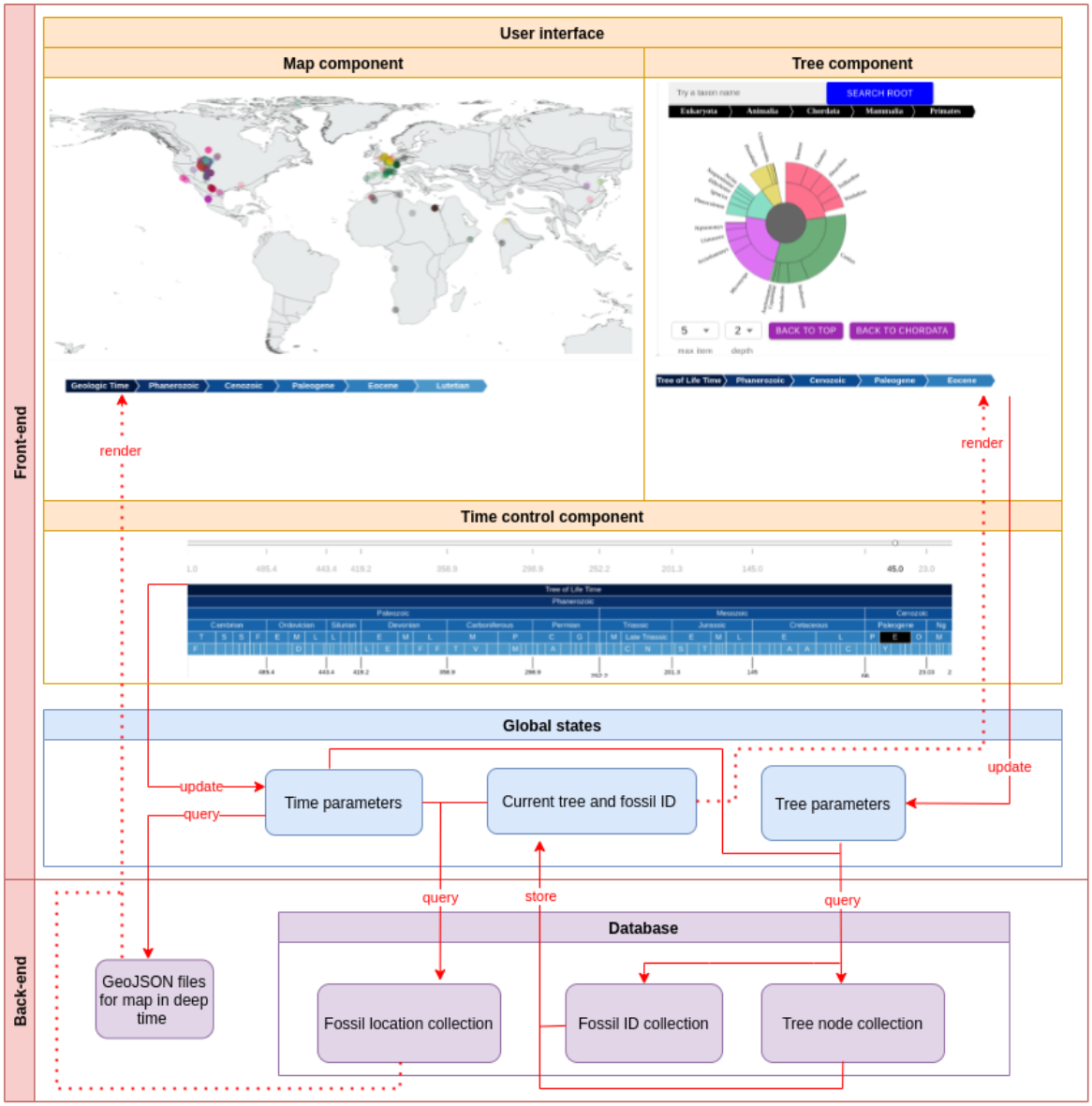


Figure 4.1: An overview of the architecture solution for the project. The user interface consists of map, tree and time control. The global states store tree and time parameters that are customized by users, and the current tree nodes attached with fossil ID sent from database. The back-end resolves queries on tree, fossil locations and map, before sending results back to front-end, where the data is visualized.

Diagram 4.1 proposes a structure of the designed visualization system. The front-end of the web application has a user interface and global states section that stores variables accessible to the components of the user interface. The back-end stores data either in database or as static files, processes front-end's data requests, and sends data back to either global states for further processing and storage, or directly to user interface for visualization. Map, tree and time control are the three components that users can view and interact with. The time control panel allows users to select a time to either view the tree of life and the related fossils, or track the fossil location change due to plate motion. The *Time parameters* from global states section listen to user-induced changes from the time control panel, and trigger requests of map data, tree data (together with tree parameters stored in global states) or fossil location data (together with fossil ID information stored in global states). The map component receives 1, coastline boundary data from GeoJSON static files, and 2, fossil location data from database, to render world map and fossils at a time point. The tree component shows the tree of life stored in *Current tree and fossil ID*. When users customize the tree view, the tree parameters are updated, and new data on tree nodes with attached fossil IDs is queried from the database. The data is returned to *Current tree and fossil ID*, which in turn processes the data and sends only the tree node information to the tree component to be rendered as the new tree.

4.2 Data processing

4.2.1 Raw data parsing

The data source for fossil records and the tree of life is provided by Paleobiology Database (PBDB) [PM16]. The raw data consists of two csv files that contain 1240202 animal fossil records and 105918 plant fossil records, respectively. Since the two files share the same data attributes, the data is concatenated into one file. Only the attributes that are relevant to this project are parsed, which are listed below.

Extracted attributes in the PBDB dataset:

- *occurrence_no*: a unique ID code for a fossil record
- *accepted_name*: the accepted name of a fossil record
- *accepted_rank*: the taxonomic rank in which the accepted_name belongs to
- *phylum*: the phylum name of the fossil record
- *class*: the class name of the fossil record
- *order*: the order name of the fossil record
- *family*: the family name of the fossil record
- *genus*: the genus name of the fossil record
- *max_ma*: the oldest estimated time of the fossil record, in million years
- *min_ma*: the earliest estimated time of the fossil record, in million years
- *lng*: the longitude of the fossil record's found location
- *lat*: the latitude of the fossil record's found location

In addition, we append a new attribute to the dataset to keep note of which one of the two files a fossil record is originated from:

- *kingdom*: the kingdom name of the fossil record

The kingdom of "Animalia" and "Plantae" is assigned to each fossil record, depending on whether it is from the animal or plant file, respectively.

By observing the available attributes in the raw data, we can determine how it can be used for the project. The biodiversity of a given time period is represented by 1, a dynamic tree of life formed by the fossil records during that time period, and 2, the locations of the fossils on a world map of that time period. The *occurrence_no* uniquely identifies a fossil record. The *max_ma* and *min_ma* together describe during which time period(s) the organism was alive. The *lat* and *lng* are used to calculate the location of a fossil at an ancient time point. Most importantly, a tree of life is needed to host each fossil record on a specific node. To generate a tree structure, information on parent-child relationship is essential. This relationship is represented in the names on each of the taxonomic ranks, in the descending order of *domain*, *kingdom*, *phylum*, *class*, *order*, *family*, *genus*, *species*. The raw data lacks information on *domain* and *species*. However, since all records are either animals or plants, they all belong to the common domain of "Eukaryota". A fossil record has a species name only if it is identified down to the species level. Then, the value of *accepted_rank* is "species", and the species name is found in *accepted_name*. Therefore, the raw data has sufficient information to construct a tree of life with the eight above-mentioned taxonomic ranks as its levels.

The datasets on fossils and tree are created from this parsed raw data. In the next steps, we first prepare the dataset that contains all fossil points, since the tree dataset is created based on it.

4.2.2 Fossil point dataset preparation

Multiple coordinates exist for each of the 1.3 million fossil record, as a certain location on the planet varies at different time points throughout earth history. As a result, two datasets are created in regarding to fossil data. This section addresses the first dataset *fossil point*, which concerns the taxonomy and the range of age for each record, and sub-section 4.2.4 addresses *fossil location*, the second dataset that stores the coordinates of each fossil record at various time points in history.

The purpose of *fossil point* dataset is to accurately describe a given fossil record in relation to the tree of life, without considering its location at a given time period. The parsed raw data already provides *occurrence_no*, which can serve as a fossil's unique identifier. However, one more attribute is required to describe its taxonomy, such that it can be attached to a particular node in the tree of life to be constructed in the next step. This attribute is essentially the unique identifier of a node in the tree of life, such that a given fossil record is attached to one and only one node in the tree. It is thus also the link between the fossil point dataset and the tree dataset. Although it is natural to consider *accepted_name* and *accepted_rank* in the parsed raw data for this purpose, they cannot be used for the following reasons. Firstly, *accepted_name* and *accepted_rank* does not correlate well with the names on the taxonomic ranks, as some *accepted_rank* values does not belong to any of the eight ranks that host the parent-child relationship in the tree. As a consequence, these nodes defined by *accepted_name* and *accepted_rank* cannot be placed in the tree, leaving their fossil records excluded from the tree as well. Secondly, the *accepted_rank* and *accepted_name* attributes are not unique. For instance, the genus "Banksia" is both an animal taxonomy and a plant taxonomy. In such cases, fossil records can be associated to a incorrect branch if only these two pieces of information are used. Thirdly, many names in the taxonomic ranks are missing. This leads to ambiguity when attaching a particular fossil to the tree of life, when there exists a rank above the *accepted_rank* with missing names. An example is shown below.

occurrence_no	accepted_name	accepted_rank	kingdom	phylum	class	order	family	genus	...
3285	Tetradium	genus	Plantae	Rhodophyta	NaN	NaN	NaN	Tetradium	...

Table 4.1: An example row of parsed raw data with missing information.

With only the *accepted_name* "Tetradium" on the *accepted_rank* "genus", we are uncertain about which class, order or family to associate this fossil record to under the phylum of "Rhodophyta". Extra knowledge in taxon-

omy is needed to retrieve the missing names on the path from "Rhodophyta" to "Tetradium" and this challenge is beyond the scope of the current project. The current solution only attaches the fossil record "3285" to the phylum of "Rhodophyta" and ignores its taxonomy below the phylum level.

To solve these three issues and provide a method for uniquely identifying a fossil record to the tree without ambiguity, the attributes *path_from_root* and *rank* are introduced.

- *path_from_root*: a comma separated string indicating the node's path from the root "Eukaryota" to the name on lowest possible taxonomic rank with no missing names in between
- *rank*: the lowest taxonomic rank of *path_from_root*

A path begins from the common root "Eukaryota" and terminates when reaching the lowest rank level "species", or encountering a missing name. For instance, the fossil record in Table 4.1 has "Eukaryota, Plantae, Rhodophyta" as *path_from_root* and "phylum" as *rank*. All names in *path_from_root* are guaranteed to be present in the tree, since the names are taken from the same taxonomy information that is used to construct the tree. The path from the common root Eukaryota describes the exact location of a node in the tree, which serves as the node's unique identifier. In addition, an important search scenario is to retrieve all fossils associated to a taxonomy's subtree. Apparently, those fossils' *path_from_root* values start with the path of the taxonomy of interest. Therefore, they can be easily obtained by string matching this path using regular expression.

To calculate *path_from_root* and *rank*, the following attributes are used: *domain*, *kingdom*, *phylum*, *class*, *order*, *family*, *genus*, *accepted_name*, *accepted_rank*. Instead of iterating over each row in the parsed raw data, we first aggregate the rows when those values are identical and then perform the iteration, before joining the result with the original data (similar to an "inner join" operation in SQL), in order to avoid repetitive computation.

At each iteration on the aggregated data, we execute the following algorithm. **FIX**: add pseudocode.We initialize the *path_from_root* as "Eukaryota" and visit each taxonomic rank from *kingdom* to *genus*, while appending a comma and the name on each rank to the end of the string and updating the current *rank*. The iteration terminates upon encountering the first missing name. At this point, both *path_from_root* and *rank* are those of the missing name's parent. If there are no missing names until visiting *genus*, the iteration finishes when the *accepted_rank* is also "genus". Otherwise, the *accepted_rank* could be either "species" or "subspecies". Under this circumstance, we append the value of *accepted_name* to *path_from_root*, update *rank* to be the value of *accepted_rank*, and terminate the iteration. The name in the path after genus can thus be either a species or a subspecies, which is acceptable, since a subspecies name already includes the species name (*i.e.*, the species name of the subspecies "Argopecten circularis impostor" is simply "Argopecten circularis").

It is important to note that the columns *path_from_root* and *rank* are precursors to the tree dataset, since they represent all nodes with at least one fossil record attached. The unique values of the two columns are used for further processing in sub-section 4.2.3. In the meantime, the aggregated data with *path_from_root* appended are joined with parsed raw data, such that every fossil record has a path. Finally, the *fossil point* dataset is generated, by extracting the following attributes:

- *occurrence_no*
- *path_from_root*
- *max_ma*
- *min_ma*

4.2.3 Tree dataset preparation

The purpose of the *tree* dataset is to describe every node on the tree of life that can also be easily searched. The following attributes are included:

- *path_from_root*: a comma separated string indicating the path from the root Eukaryota to the name on lowest possible taxonomic rank with no missing names in between
- *parent*: the *path_from_root* of its parent
- *name*: the name of the node
- *is_leaf*: a boolean value that indicates if the node is a leaf node, *i.e.*, has no children
- *max_ma*: the earliest age in million years among all fossils in the subtree of this node
- *min_ma*: the latest age in million years among all fossils in the subtree of this node

For each node, we need a unique identifier, which is provided by *path_from_root* as discussed in the above sub-section 4.2.2. The last *name* of the path is also extracted as an attribute, such that the nodes can be search via name. The node's parent attribute simplifies searching for all children of a given node. The *is_leaf* attribute indicates directly whether or not a node has children without searching for them. Lastly, *max_ma* and *min_ma* provide a quick way to tell if a node exist during a given time period without searching for fossils attached to its subtree.

To prepare this dataset, we start from the precursor dataset from 4.2.2, which has two columns: *path_from_root* and *rank*. As a reminder, the *path_from_root* are the unique identifiers of all nodes in the tree with at least one fossil record attached. However, there are more nodes to include in the tree. For each row in this precursor dataset, we must also include each of the upstream ancestor in the tree. For instance, when a node "a,b,c,d" exists, its ancestors "a,b,c", "a,b" and "a" should also be included in the tree. Therefore, the first step is to add the missing ancestors, by iterating every *path_from_root*. **FIX**: add pseudocode. Given a path, the *path_from_root* values of all its ancestors are simply strings sliced from the beginning of the path, until each of its commas. The *rank* value of an ancestor can also be computed by counting the number of commas in the path, as a "domain" has no comma in its path, and a "genus" has six commas. After the iteration, only the unique *path_from_root* and the corresponding *rank* values are kept .

The *parent* and *name* values can be easily computed from a *path_from_root*. The common root "Eukaryota" has no parent, and any other node's parent is the sub-string of its *path_from_root* sliced from beginning until the last comma (*e.x.*, the parent of "a,b,c" is "a,b"). Similarly, the *name* of a node is the sub-string beginning from the first character after the last comma in its path (*e.x.*, the name of the node "a,b,c" is "c"). Next, we visit each *path_from_root* once again. If the path is in the set of all *parent* values, then this node is a parent (*i.e.*, not a leaf node), and its *is_leaf* value is set to be false, otherwise true.

Finally, we calculate the *max_ma* and *min_ma* values for each node, using the processed *fossil point* data from 4.2.2, which now contains the path name and the age range information for each fossil record. **FIX**: add pseudocode. We first create a reference dataset, by grouping the fossil data by the same *path_from_root* on two attributes: 1, maximum of *max_ma* values among the fossils with identical path and 2, minimum of *min_ma* values. Next, we iterate through this reference dataset. For each path, we extract all its ancestors using the same string splitting method as mentioned above, visit those ancestors in the tree dataset, and 1, update its *max_ma* only if it is less than the current *max_ma* of the path under iteration and 2, update its *min_ma* only if it is greater than the current *min_ma* of the path under iteration.

4.2.4 Geospatial transformation of map and fossil data

Apart from the tree, a major component of the visualization is a world map responsive to time selection, on which the locations of the fossils are also shown. Therefore, given a geological timescale, we need to calculate the world maps in all time periods. For each fossil point, its coordinates during those time periods are also calculated and stored in the *fossil_location* dataset.

The timescale data is provided by PBDB. Similar to the tree of life, it is also hierarchical, with five layers in total. The highest layer Phanerozoic ranges from 541 million years ago to present time, and the intervals on the fifth layer often spans only several million years. Nevertheless, plates move constantly, such that the location of a given point at the end of a time interval varies from the location at the start of the interval. Our solution is to represent the geology of a time period using the middle value of a certain fifth-layer interval. When the user select a time interval in the fifth layer, the coordinates are transformed to that of the interval's middle time point rounded to an integer. When a higher-layer time interval is selected, we first find the fifth-layer in which the middle value of the selected interval falls into, before transforming the coordinates to the integer time point in the middle of that fifth-layer interval.

Both the global coastlines boundaries of modern day and the model to transform a modern location to an ancient location are provided by Matthew *et al.* [MMZ+16]. In this paper, they proposed a high-resolution model on the movement of plates from late Paleozoic (410 million years ago) to modern days. This model transforms the global coastlines or points to a desired time point no earlier than 410 million years ago. In the parsed raw PBDB data from sub-section 4.2.1, we extract the *lat* and *lng* attributes as the modern coordinates of a fossil point, and *occurrence_no* as the ID for each fossil point. For a given point on the planet, the model assigns it to a plate, before calculating its ancient position as it relocates along with the movement of its assigned plate. Finally, the map data is stored as GeoJSON files, and the ancient locations of each fossil point is stored in the *fossil_location* dataset with the following attributes:

- *occurrence_no*: the ID of a fossil record
- *mya*: the time point as an integer in million years
- *lat*: the latitude of the fossil record in the corresponding time point
- *lng*: the longitude of the fossil record in the corresponding time point

4.3 Database setup

The fossil and tree information are stored in a database, from which requested data is sent for visualization according to queries generated from user interactions. The three prepared datasets are uploaded to the database, with slight renaming and reorganizing in the following manner:

- *fossilLocation*: It stores the *fossil location* dataset, which contains the locations at different time points for each fossil record. It has the following fields: *id* (*occurrence_no* from *fossil location*), *mya* (*mya* from *fossil location*), *coordinates* (an array of [*lat*,*lng*] from *fossil location*).
- *fossilPoint*: It stores the *fossil point* dataset, which contains taxonomy and range of age for each fossil record. It has the following fields: *id* (*occurrence_no* from *fossil point*), *maxma* (*max_ma* from *fossil point*), *minma* (*min_ma* from *fossil point*), *pathFromRoot* (*path_from_root* from *fossil point*).
- *TreeNode*: It stores the *tree* dataset, which contains each node in the tree of life. It has the following fields: *pathFromRoot* (*path_from_root* from *tree*), *maxma* (*max_ma* from *tree*), *minma* (*min_ma* from *tree*), *parent* (*parent* from *tree*), *name* (*name* from *tree*), *isLeaf* (*is_leaf* from *tree*).

An entity relationship is shown in Figure 4.2. *FossilLocation* is linked to *FossilPoint* via *id*, the unique identifier for a fossil record. An instance of *FossilPoint* can relate to many instances of *FossilLocation*, because a fossil point has many locations at various time points. An instance of *FossilLocation* relates to one and only one fossil record in *FossilPoint*. An instance of *FossilPoint* is uniquely identified by its *id*. *FossilPoint* and *TreeNode* are linked via *pathFromRoot*. One fossil record is associated to one and only one node in *TreeNode*, however, a tree node can have zero or many fossil records. Indeed, a tree node of a high taxonomic rank has no fossil attached to it, when all fossils under this taxonomy are identified with a lower rank, thus are attached to its descendants. For instance, the root node "Eukaryota" has no fossil attachment, since for any given fossil, we have at least identified it as an animal or a plant. Finally, an instance of *TreeNode* represents a certain node in tree of life, therefore is uniquely identified by its *pathFromRoot*.

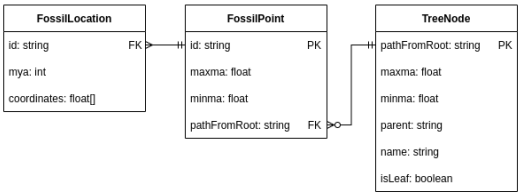


Figure 4.2: An entity relationship diagram of the database.

4.4 Back-end query processing

The back-end is responsible for processing the data queries requested by user actions, and sending the result to the front-end. How the queries are resolved is explained in this sub-section. There are two main queries to carry out: 1, given a time range and tree parameters, obtain a tree of life with fossil IDs attached to each node, and 2, obtain the locations of the fossil records given their IDs and a time point.

The first query returns a fossil-attached tree under a requested root node. The following parameters are passed to the database:

- *name*: the root node's name
- *maxma*: the start of the selected time period
- *minma*: the end of the selected time period
- *maxElement*: the maximum number of siblings to retrieve under each node
- *depth*: the number of tree layers to retrieve

The resulted tree should be rooted by the requested node, with a depth no greater than the parameter *depth*. The number of children under each node is no greater than number of *maxElement*. The IDs of fossils that exist during the time period is also attached to each node. The returned tree is an array of nodes with the following fields:

- *pathFromRoot*: the unique identifier of the node
- *name*: the node's name
- *parent*: the *pathFromRoot* value of its parent
- *isLeaf*: a boolean value indicating if the node is a leaf node with respect to the returned tree
- *fossils*: the IDs of fossil records existing during the requested time period that either are attached to this node (if the node is not a lead node) or belong to the node's subtree (if the node is a lead node)

To retrieve the tree, we first find the requested root node by searching for its name in *TreeNode*. The *name* is passed either from user input in a search bar or from a clicking event on a visualized node. Since the first letter of all names in *TreeNode* are capitalized, we capitalize the first letter in *name* in case it is not, before performing the search. The *parent* value of the root node is set to null, since although it may have a parent in the complete tree of life, it acts as the root with respect to the tree to be returned. The fossil IDs attached to the root node is also appended. We then recursively search for its children by using the *parent* field and append the children nodes to the result. The recursion terminates until reaching the requested depth, or encountering nodes with no children left. When processing a certain node during the recursion, we compute the *isLeaf* value in the following way. If this node has no children in the complete tree of of life (*i.e.*, this node's *isLeaf* value in database's *TreeNode* is already true), then it is certain a leaf node in the tree under construction. However, if the iteration is on the maximum depth, then the node is a leaf node with respect to the tree under construction, whether or not it has children in the complete tree of life. When it is a leaf node, we search for the IDs of all fossils belonging to its subtree. This can be easily performed with string matching the *pathFromRoot* of the fossil records in database's *FossilPoint* using regular expression, since all desired fossils' paths begin with the *pathFromRoot* of the current node. When the node is not a leaf node, we only attach fossils that are strictly associated with this node, by exact string matching of *pathFromRoot* between this node and the fossil records. To improve efficiency, the sibling nodes are filtered before continuing to the next recursion level. A count value is computed for each sibling node, which is the number of fossil records belonging to its subtree that is present during the time period. Then the siblings are ranked in descending order, and the top *maxElement* number of siblings are kept.

The second query receives a time point of interest and an array of nodes with fossil IDs attached, which is a result of the first query. However, only the relevant fields are passed:

- *name*: the node's name
- *fossils*: the IDs of fossils attached to the node (if it is a lead node) or its subtree (if it is not a leaf node)

From *FossilLocation*, we retrieve the fossils' coordinates by their IDs and time point, and organize the result in an array. Each item has the following fields:

- *id*: the fossil record's ID
- *coordinate*: the fossil location at the time point, in the form of [latitude, longitude]
- *name*: the name to which the fossil record is associated in the current tree
- *mya*: the requested time point

4.5 Front-end development

This section introduces the design solution for the front-end, which includes a global states section that locally stores variables to which different components in the user interface have access, and the user interface where viewers interactively explore the visualized data. The user interface is in turn divided into a time control component, a map component and a tree component.

4.5.1 The global states

The global states serve as the intermediate agents between the user interface and the back-end, and between different components of the user interface. As shown in Figure 4.1, global states include *Time parameters*, *Tree parameters* and *Current tree and fossils*.

Time parameters. Time parameters store a time period and a time point. Upon a time-related request from the user interface, certain values of time parameters are updated, and new data queries are triggered. When only the

time point is changed, new fossil location data is queried using time point and the fossil IDs stored in *Current tree and fossil ID*, and new map data is loaded from static files in the back-end. When time period is changed, a time point that represents the period is also calculated. Data on a new tree and attached fossil IDs are queried in regarding to the time period, before storing the new result by updating *Current tree and fossil ID*. Due to the updated time point value, data on map and fossil location is also queried.

Tree parameters. Tree parameters keep the record of how the tree is customized by the users. It includes the requested depth, the maximal number of children to retrieve under a given node, and the name of the node to search for. When any of the parameters are changed as a result of user actions, combining with the time period information in *time parameters*, a new data query is sent to compute a customized tree of interest formed by fossils found in the time period, and both the tree nodes and the attached fossil IDs are stored in *Current tree and fossil ID*. Lastly, in the user interface, when a user hover on a node in the tree graph, or a fossil point on the map, the identity of the node is recorded as a parameter, in order for both tree and map to be aware of, which is required for the communication between the two components. The tree parameters are also responsible for the communication between map and tree components. When a user hovers the mouse on either a node in the tree graph, or a fossil point attached to a certain node on the map, a parameter also records the identity (*pathFromRoot* value) of the tree node. As a result, the information on which node is being highlighted is shared with the other component, which also elicits a highlighting effect as a response.

Current tree and fossil ID. This global state variable stores information on current tree nodes with fossil IDs attached. It mainly serves two purposes. When users are interested in viewing how the same fossil points move as a result of plate movement through time, they change the time point value in *time parameters*, triggering a data query for new fossil locations with the pre-stored fossil ID values. Under this circumstance, the fossil IDs remain constant, it is thus more efficient to store them locally than re-querying upon a new time point. In addition, the colors of nodes shown in the tree component is consistent with the colors of their attached fossil points shown in the map component. As a result, upon receiving new tree and fossil ID data from the database, instead of sending directly to the tree component in the user interface for rendering, we compute the colors before storing in *Current tree and fossil ID*. Thereby, the color values are calculated for only once, and are shared by both map and tree component when they render fossil points and tree respectively. The coloring of the tree node is determined as follows. The current root node is in black. Its descendants are assigned with one of the five distinctive colors. The assignment initiates from the children of the root node, by visiting each node in a breath-first-search manner. A color is removed from the palette when it is assigned to a node. After all five colors are used, if the new node being visited is still a child of the current root, then the last of the five color is assigned to this node; otherwise, we assign the same color as its parent, which must already have one of the five colors, as the parent has been visited and assigned with a color in the breath-first-search process.

4.5.2 The user interface: time control component

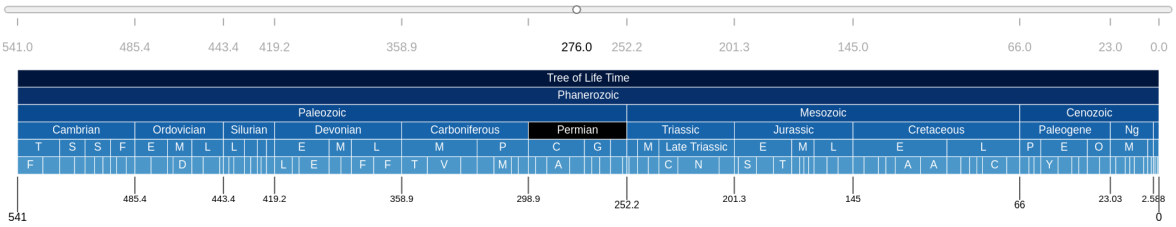
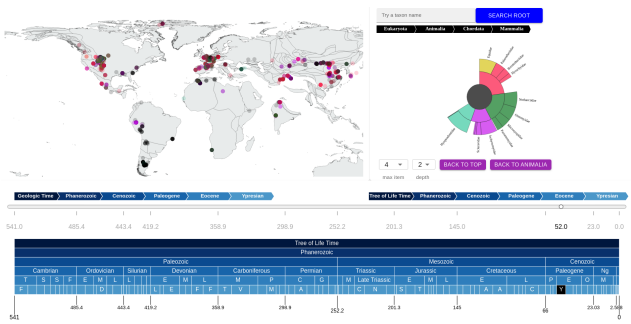
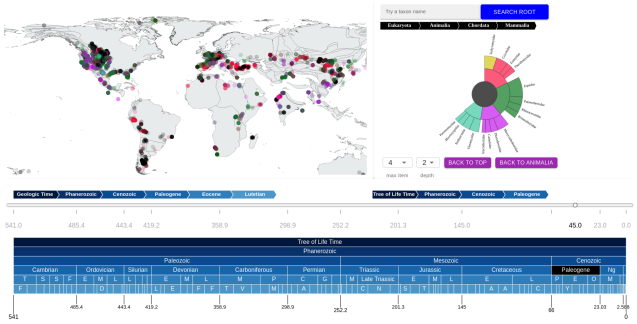


Figure 4.3: An overview of the time control component. A timescale table displays all time periods of the Phanerozoic that are organized hierarchically in five layers. When selecting a time period, *i.e.* "Permian", its cell is marked in black. Some tick values are shown below the table, marking the start and end years of all time periods on the third layer, in the unit of million years. There is a scroll bar on the top that allows users to drag, in order to view the world map and the fossil locations of a later time than the selected time period.

The time control component shown in Figure 4.3 sits at the bottom of the user interface. It allows users to control the map and tree components by either clicking in the timescale table, or dragging the scroll bar. The first functionality is shown in Figure 4.4. Users can pick a time period of interest to view the tree and fossil points during that period by clicking a cell in the time table. Only a time period on or below the third layer can be selected, in order to avoid requesting fossil data spanning hundreds of millions of years at a time, which may exceed the size of the heap memory. When a time period is selected, a time point at which the world map and the fossil locations will be queried is also calculated. This time point is intended for representing the geology during the time period. Due to the hierarchical nature of timescale, a certain time point must belong to a time period in the fifth layer. We intend to show the name of this time period, since it is of the highest detail. When a fifth-layer time period is selected, the time point is simply the middle value between the start and end year, rounded up to an integer (Figure 4.4(a)). Otherwise, the middle value of the time period is calculated, and we search for the fifth-layer time period where this calculated middle value falls into. Then, the middle integer value of that fifth-layer time period is used as the time point (Figure 4.4(b)). The second functionality is shown in Figure 4.5. Users can pick a time point that is later than the selected time period to view how fossil locations are shifted with the continents, by dragging the scroll bar on the top. Those time points are the middle values of all fifth-layer time periods later than the current time point, rounded up to integer.



(a) The time period "Phanerozoic -> Cenozoic -> Paleogene -> Eocene -> Ypresian" is selected. Since this is a time period in the lowest layer, the timelines below both map and tree show the name of this time period. The map in fact shows the continents and fossil locations at the middle point of this period.



(b) The time period "Phanerozoic -> Cenozoic -> Paleogene" is selected. This time period is not in the lowest layer, but its middle value falls into "Phanerozoic -> Cenozoic -> Paleogene -> Eocene -> Lutetian", a time period on the lowest fifth layer. Therefore, in the map component, Lutetian is the time period that represents for the selected Paleogene, and its name is shown beneath the map on the left. We then take the middle value of Lutetian as the time point at which the continents were shaped and the fossils were located.

Figure 4.4: The timescale functionality of the time control panel. Users can select a period in the timescale to view the map and tree during this time period. The tree of life is always in accordance to the selected time period, whereas when the time period is not on the lowest layer in the timescale, the map component picks the middle value of a time period on the lowest layer in which the middle value of the select time period falls.

The entire Phanerozoic spans over 500 million years, whereas certain time periods, especially those of lower layers, are only hundredth of that length. Without dynamic scaling of the time table, it is difficult to have a clear view or an easy accessibility of the shorter time periods. A solution is demonstrated in Figure 4.6. When hovering on a time period on the third degree or lower, a zooming effect takes place, where all time periods from the third layer and below are elongated, such that the third layer cell occupies 90% length of its immediate ancestor. As a result, labels indicating the start and end year of the shortest time periods on the fifth layer can also be shown. To further highlight the time period, when hovering on a time period, the opacity of all other time periods that are not in the path to the common ancestor "Phanerozoic" are decreased, and a small pop-up note also shows its name as a path from the common ancestor.

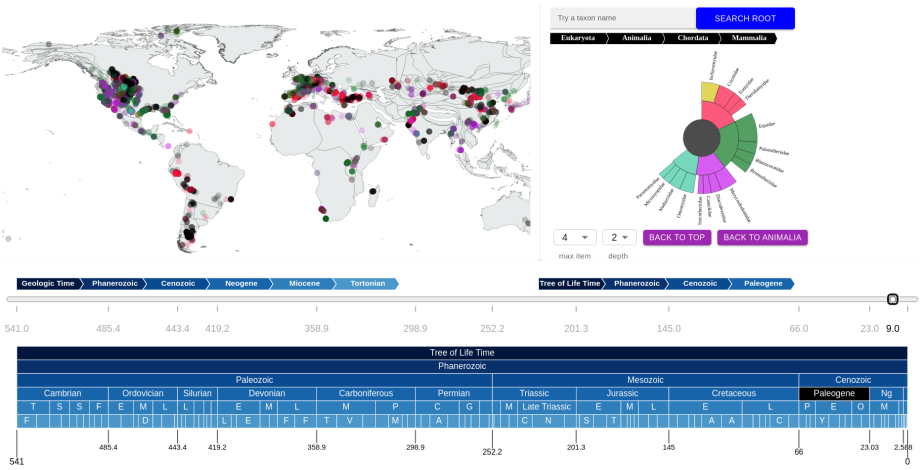


Figure 4.5: The scroll bar functionality of the time control panel. Users can drag the scroll bar to a later time point than the currently selected time period, in order to track how the continents and the fossil points move due to plate motion. From the time table and the breadcrumb beneath the tree, we can see that the tree of life and the attached fossils are still with respect to the selected "Phanerozoic -> Cenozoic -> Paleogene". However, the number of the scroll bar and the breadcrumb beneath the map imply that the scroll bar has brought the world map to a more recent time point at 9 million years ago, which is during "Phanerozoic -> Cenozoic -> Neogene -> Miocene -> Tortonian".

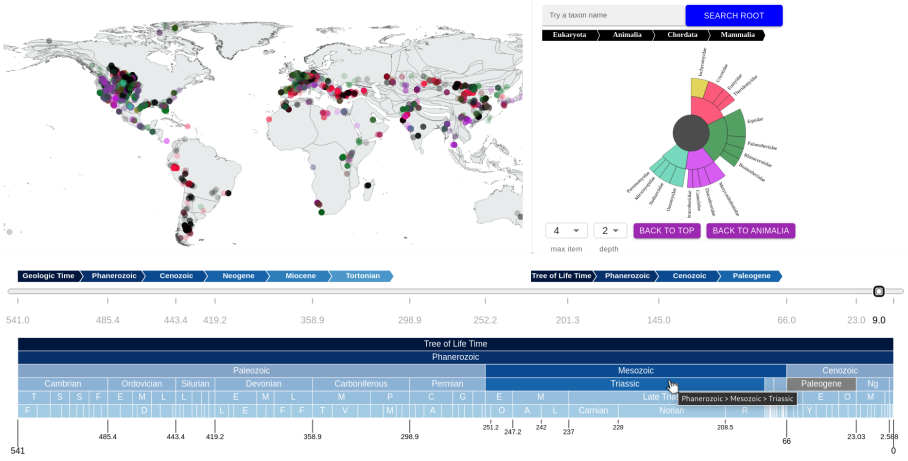
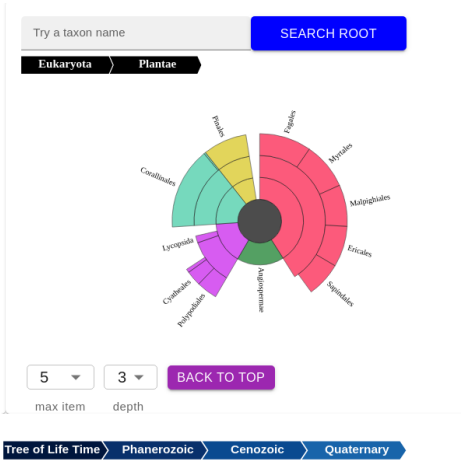
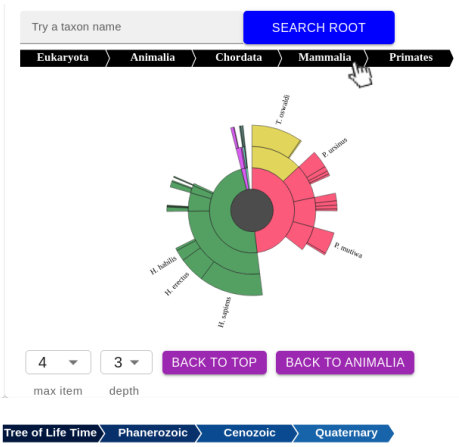


Figure 4.6: The zooming and highlighting effect of the timescale upon hovering. When users hover on a particular time period, a pop-up note with its name shows next to the pointer, and all other time periods that are not in its path to Phanerozoic are decreased in opacity. The selected "Triassic" belongs to a period on the third layer or below, as a result, a zooming effect takes place. All time periods at or below "Triassic" are proportionally elongated such that "Triassic" takes up 90% the length of its parent "Mesozoic". Due to this expansion, the tick values can now show all start and end years on the lowest layers.

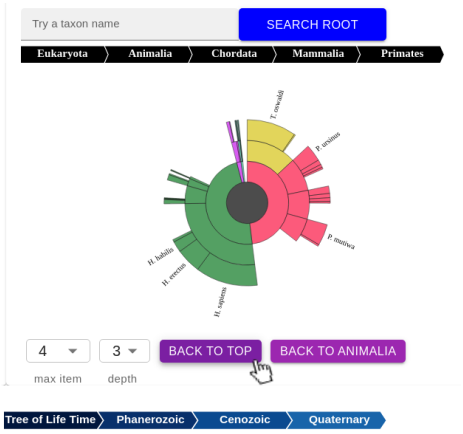
4.5.3 The user interface: tree component



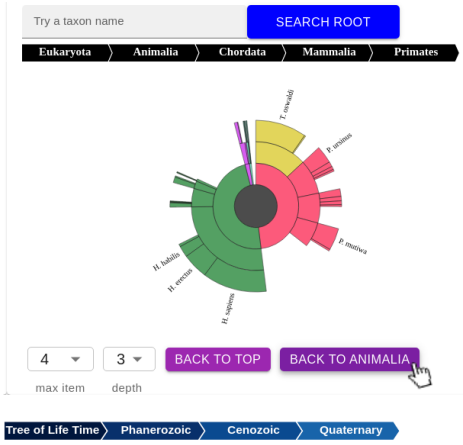
(a) A tree component showing "Plantae" and its descendants during "Quaternary" time period.



(b) A tree component showing "Primates" during "Quaternary" time period. The pointer is hovering on ancestor "Mammalia". It is possible to navigate to any ancestor of the current node in the breadcrumb by clicking.



(c) The tree is the same as (b). The pointer is hovering on "BACK TO TOP" button, which directs the users to the top of the tree "Eukaryota".



(d) The tree is the same as (b). The pointer is hovering on "BACK TO ANIMALIA". Animalia is the *depth*'th ancestor of the current root. Note that this button is absent in (a), since the current root does not have a *depth*'th ancestor.

Figure 4.7: The tree component of the user interface. From top to bottom, it has a search bar for user input, a breadcrumb in black that shows the position of the current root, a sunburst graph that reflects the prevalence of each node during the time period via arc angles, a control panel for tree customization, and a breadcrumb showing the current time period.

The tree component is on the right of the user interface. It shows the current tree of interest, while providing options to customize the tree. A typical tree component is seen in Figure 4.7(a). A search bar is located on the top, where users can search for a name. Below, a line of breadcrumb indicates the location of the current root in the complete tree of life, by showing all of its ancestors in a path from the common root. Each ancestor can be visited by clicking (Figure 4.7(b)). Since the ancestor nodes are upstream of the current root, they are colored in black, to distinguish from the current root's descendants in other colors. The tree is rendered as a sunburst graph from the global state variable *Current tree and fossils* addressed in sub-section 4.5.1, which contains information

on a node's name, color and fossils.

To demonstrate the prevalence of a certain taxonomy during the selected time period, the angle of each node's arc reflects the number of fossils attached to its subtree. The current node under inspection is the root of the tree being displayed, as a consequence, it is a full circle, since it includes all fossils in its subtree. The arc angles of sibling nodes are proportional to the ratio between the number of fossils in their respective subtree, and the number of fossils included in their parent's subtree. To calculate the number of fossils under a given node's subtree, we recursively sum up the fossil counts from leaf nodes to the current root node, by the number of fossils attached to each node. Note that when performing the sum up, if the node is a leaf node, we sum up by all fossils under its subtree, however, when the node is not a leaf node, we sum up by the fossils that are attached to the node. This is because if we still sum up by the fossils belonging to its subtree, we are double counting the fossils, as each of its children is already summed up by the fossils under their respective subtree. *Current tree and fossils* conveniently stores fossils attached to node for non-leaf nodes, and fossils associated to the node's subtree for leaf nodes. Therefore, when performing the sum up at the front-end, no further processing is needed to distinguish if we are counting the fossils with respect to only a node, or its subtree.

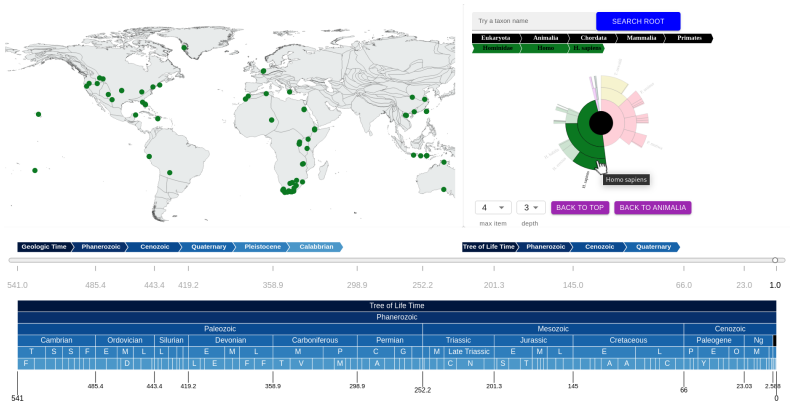
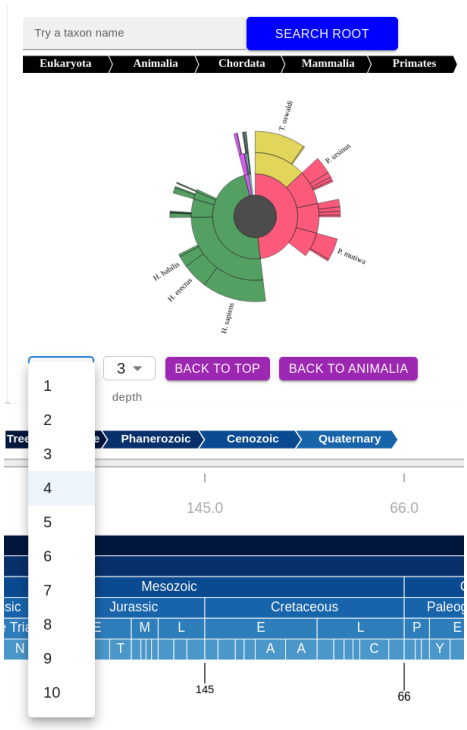


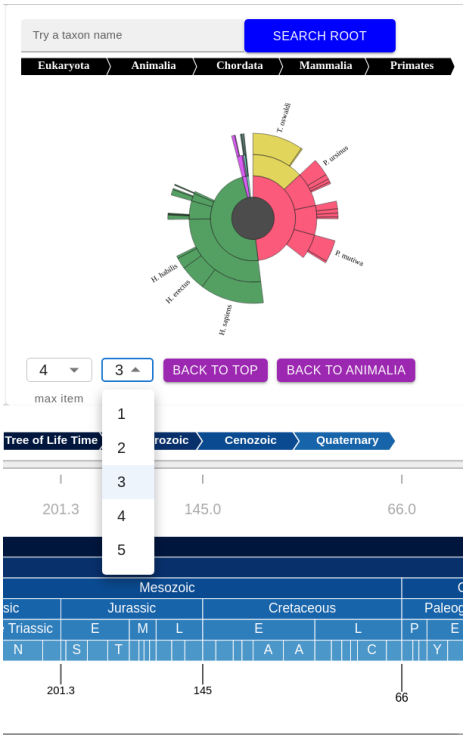
Figure 4.8: The tree's visual effect upon user hovering on a node. The tree is the same as in Figure 4.7(b). When user hovers on "Homo sapiens", a pop-up note shows its name, and an additional breadcrumb in the color that is consistent to that of the tree appears, indicating the position of the hovered node in the complete tree. The nodes that are in its path to the current root are in full opacity, whereas all other nodes have decreased opacity. Most importantly, the map on the left side only shows the fossils related to the hovered node, during the time when the user is hovering on it. Finally, it is possible to click on the node to navigate to the tree rooted by it.

To highlight a certain node of interest, we provide the following visual aids (Figure 4.8). First, a second line of breadcrumb indicates its position from the current root node. Second, similar to the time control component highlighting effect, the opacity of other nodes are decreased, and a pop-up note informs the user on the node's name. When a node's name is too long or a node's arc is too narrow to display on the tree, the pop-up note and the dynamic breadcrumb provide an opportunity to view this name upon hovering. Finally, the hovering effect is communicated to the map component, such that only the fossil points attached to the hovered node are shown and in full opacity, to inform the users where their interested taxonomy inhabited in the world during the selected time period.

To enable users to customize and navigate the tree, we designed a small control panel below the tree graph. If the current node is not the common root "Eukaryota", the *Back to top* button directs the user to it (Figure 4.7(c)). Clicking on the "Eukaryota" in the breadcrumb above the tree graph provides the same functionality. If the distance from the current root to "Eukaryota" is greater than the current *depth*, a second button redirects the users to upstream of the current node that is it's *depth*'th ancestor. As an example, in Figure 4.7(d), the *depth* value is 3, and current root is "Primates". From the breadcrumb in black, we can see that the third ancestor of "Primates" is "Animalia", hence the button in Figure 4.7(d) that provides a quick way to navigate to it. In Figure 4.7(a),



(a) Upon clicking on *max item*, a drop down list appears, prompting the users to select a number from 1 to 10, which is the maximal number of siblings to shown under a given node.



(b) Upon clicking on *depth*, a drop down list allows users to customize the number of depth to show with respect to the current root, which is a number from 1 to 5. This number also determines which upstream ancestor to navigate to in the second "BACK TO" button.

Figure 4.9: The drop down lists in the tree's control panel.

however, the current root is "Plantae", and it does not have a third ancestor, since its first ancestor is already the common root "Eukaryota", hence there is only a "BACK TO TOP" button. To the left of the buttons, the *max item* adjusts the maximal number of elements to display under each node (Figure 4.9(a)), and the depth adjusts the tree's depth (Figure 4.9(b)).

Finally, the tree's time period is reflected below the small control panel. This is to remind the viewers that the tree of life is subjected to this particular time period, since the map they are viewing might be of a different time period.

4.5.4 The user interface: map component

The map component is on the left of the user interface. It shows the world map with fossil points at the select time point. The geological data on coastlines are provided from static files in the back-end, and the fossil coordinates are rendered directly from database's *FossilLocation*, whereas their pre-computed colors are loaded from front-end in *Current tree and fossils* discussed in 4.5.1. An overview of the map component is shown in Figure 4.10. The zooming and panning functionality enables users to view regions of interest in greater detail, and when hovering on a fossil point, the tree component shows which node it attaches to, by highlighting the node and providing a second breadcrumb that indicates its position, in a similar way as hovering on a node in the tree (Figure 4.8). Lastly, often fossil points are of very close proximity, or even have identical locations, leading to the fact that the points can easily be on top of each other. As a solution, we decreased the opacity of the fossil points in order to reflect all fossil records' presence — a point of higher visibility implies that there are multiple records

found at or close to the location.

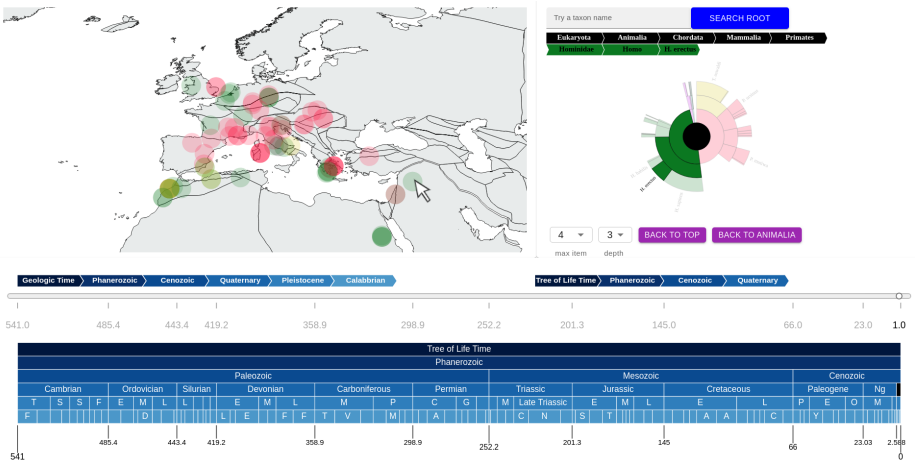


Figure 4.10: The map component of the user interface. The region around nowadays Europe is zoomed in, and the mouse is hovering on a fossil point around the nowadays Middle East region. As a result, the node this fossil is attached to, which is "Homo sapien", is highlighted in the tree component, such that the node is of higher opacity, and a second breadcrumb indicates its name and path from the root.

5 Implementation

This chapter explains how the technical solutions introduced in chapter 4 is implemented to produce the web application. First, the stacks used in both data processing stage and the web development stage are listed in section 5.1. Then, section 5.2 describes the code structure. Next, section 5.3 addresses how the data processing is implemented. Finally, section 5.4 explains the steps to realize the web application's structure shown in Figure 4.1.

5.1 Dependencies

At the data processing stage, Python3 and Pandas library are used to process the raw PBDB data to generate data on both tree and fossil information. GPlates is used to transform modern day coordinates of both fossil records and global coastlines to ancient time points. At the web development stage, MongoDB serves as the database to store tree and fossil data. Mongoose is used as the object document modeling (ODM) library for MongoDB and Node.js, as the back-end is written in JavaScript. GraphQL is the query language used for requesting data. The Apollo graph platform connects the front-end and the back-end, and it consists of Apollo Server, which communicates the GraphQL queries to the back-end and Apollo Client, which manages GraghQL queries induced from actions at the front-end's user interface. Axios is used to load world maps in GeoJSON by accessing a file's URL. Finally, React is the framework in which the front-end is implemented in, and MaterialUI is used to style the website. d3.js is the library used for generating and manipulating the visualized data that is SVG-based.

5.2 Code structure

The code for data processing, back-end and front-end are structured as follows:

- **app/**: includes the code for building the web application.
  - **server/src/**: includes the back-end code to run the server and to upload the processed datasets to the database.
    - **resolvers/**: includes the code for resolving the queries from the front-end, *i.e.*, retrieving and processed desired data from MongoDB before returning the result.
  - **src/**: includes the code for front-end.
    - **components/**: includes the code for different components and sub-components that the front-end framework is assembled from. The user interface consists of time control, tree and map components, and there is also a component to store global states and a component for data fetching.
- **data_processing**: includes the code for processing raw data to generate tree and fossil data, as well as the code to further organize fossil data to prepare it for coordinate transformation through time.

5.3 Data processing

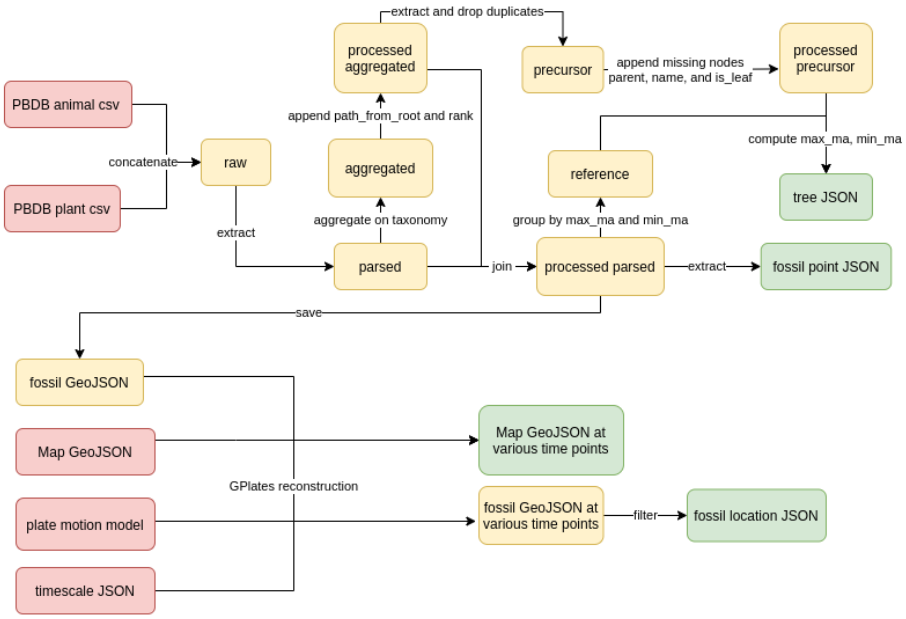


Figure 5.1: The data processing workflow. The unprocessed data sources are marked red; intermediate products are marked yellow; final products are marked in green. In short, The data on tree of life and fossil point is prepared from PBDB data on animal and plant fossils. The geospatial data on both map and fossil locations at different time points are prepared from GeoJSON data (unprocessed or intermediate files) and a published plate motion model using GPlates software.

Prior to developing the web application, we must obtain the necessary data to visualize, by implementing the strategies in section 4.2. Pandas, a standard Python library specialized in data science tasks is used for this purpose. The workflow is summarized in Figure 5.1. First, we concatenate the PBDB [PM16] data files for animal and plant fossil records into one file, the raw data. Then, we parsed the relevant attributes (listed in 4.2.1) to form the parsed data. The rows in parsed data with identical values on all taxonomic ranks are aggregated, such each row in aggregated data has unique taxonomy information. Then, the *path_from_root* and *rank* values for each row can be calculated. This processed aggregated data with appended information now represents all tree of life nodes with at least one fossil record attached. It is further processed in the following two directions. On one hand, it is joined with the parsed data, such that every fossil record in parsed data would also have the appended information. We then extract the *path_from_root*, *occurrence_no*, *max_ma* and *min_ma* from the parsed data to generate the *fossil_point* dataset, which is saved in a JSON file for uploading to database. On the other hand, we extract only the newly added *path_from_root* and *rank* in the processed aggregated data, before saving the unique rows in a precursor dataset in a csv file. This precursor dataset is responsible for the preparation of the *tree* dataset, as each of its row is a node in the tree of life. Note that the precursor dataset are still missing the tree nodes that have no fossil records attached. We first iterate it to include all missing nodes in the tree of life as mentioned in sub-section 4.2.3. Then, we compute the *parent*, *name* and *is_leaf* information for each row. The precursor data is now processed. Finally, we append *max_ma* and *min_ma* values for the rows. To do that, we use the processed parsed data again. As a reminder, a row in the processed parsed data represents an individual fossil record. Therefore, we group the rows with identical *path_from_root* (*i.e.*, the fossils attached to the same tree node) by 1, the maximum of all *max_ma* and 2, the minimum of all *min_ma*. We iterate through this new reference data, and for each tree node in the processed precursor data, we visit all its ancestors to update their *max_ma* and *min_ma*, as explained in 4.2.3. The dataset on tree is ready to be saved in JSON and uploaded to database.

With respect to geospatial data processing, GPlates is used to reconstruct all modern coordinates to ancient ones. We export the processed parsed data that contains modern coordinates of each fossil record into GeoJSON format, before loading the data on GPlates. In addition, we also load modern plate boundaries data as GeoJSON and the simulation model that defines the kinetics of plate motion since the past 410 million years, which are both available in the work by Matthew and colleagues [MMZ+16]. The timescale data is provided by PBDB [PM16]. As a reminder, it is a hierarchical structure with five layers, and we trace all modern locations to a particular fifth layer (most detailed) time period. As a consequence, all ancient time points we are reconstructing to are essentially the middle integer values of all layer five time periods that are no earlier than 410 million years. GPlates assigns all fossil points to the plate that they belong to, and reconstructs those locations together with plate boundaries, using the plate motion kinetics model. All results are exported in GeoJSON. The plate boundaries files describe the world map at a particular time point. We store them as static GeoJSON files that can be directly rendered at front-end upon requesting a certain time point. With respect to the reconstructed fossil location data, a further filtering is executed. For a given fossil record, it is in fact not necessary to keep all its ancient locations. At a time period before the earliest estimated existence of a fossil, it will never appear in the map. If a time period spans longer, then it can cover more fossils. The grossest time periods that users are capable of selecting are on layer three. If a time point is earlier than the start of the third layer time period to which the fossil's *max_ma* belongs, then we can filer out its location at this time point. After filtering, we extract each fossil's ID (*occurrence_no*), coordinates and time point in million years and store as the *fossil_location* dataset in JSON files for uploading to database. We use JavaScript for geospatial data processing due to its convenience when working with JSON and GeoJSON files.

5.4 Web development

5.4.1 Back-end

The back-end provides a database to store the data prepared in section 5.3, and an API to handle the data traffic with front-end. Although a relational database would also suffice for this project, MongoDB is used as the database of choice. It is relatively convenient to represent our prepared data in JSON with this document oriented database system. More importantly, as a NoSQL database, the lack of data normalization enables a higher horizontal scalability (*i.e.*, no rigid schema). This gives a faster iteration time in the early stage of development that is highly appreciated, since a lot of trails and errors are needed to decide the appropriate fields to include for both tree and fossil data. The *tree*, *fossil_point* and *fossil_location* JSON files from section 5.3 are uploaded to MongoDB database into collections *TreeNode*, *FossilPoint* and *FossilLocation* respectively, as described in 4.3.

When building the API between the data requesting front-end and the database, GraphQL is chosen as the data query language. A traditional REST API returns fixed data structures from the server, as a result, it may need to access multiple endpoints from the server, leading to over-fetching (downloading data attributes that are not actually needed) and under-fetching (accessing another endpoint since some attributes are not included in the returned result). Therefore, the APIs are designed according to the needs of the UI, such that a particular request corresponds to one endpoint from the server. However, a change of front-end's data needs requires the corresponding back-end adjustment. GraphQL allows more rapid iterations during early development. With GraphQL, a single data query that declares all the desired fields of a typed data is sent to the back-end, which returns a JSON object with only the required information. A data query is processed by a JavaScript resolver function that fetches data modeled by the ODM mongoose. The algorithms of the resolver functions are described in section 4.4. Now, any change in the data needs would only require re-writing the relevant data queries and changing the resolver functions, which is relatively more productive. There are two possible queries, *getTreeWithFossils*, which retrieves the tree with fossil IDs attached to each node, and *getFossilLocations*, which retrieves the fossil coordinates data at a requested time point. The GraphQL system is implemented by Apollo platform. It provides Apollo server, which connects the data from MongoDB, and Apollo Client, which sends front-end data queries and returns the data to UI. An overview of the API structure is shown in Figure 5.2. Finally, the world map at a certain time point

is stored as a GeoJSON file accessed by its URL, and it can be loaded by sending an HTTP request using axios.

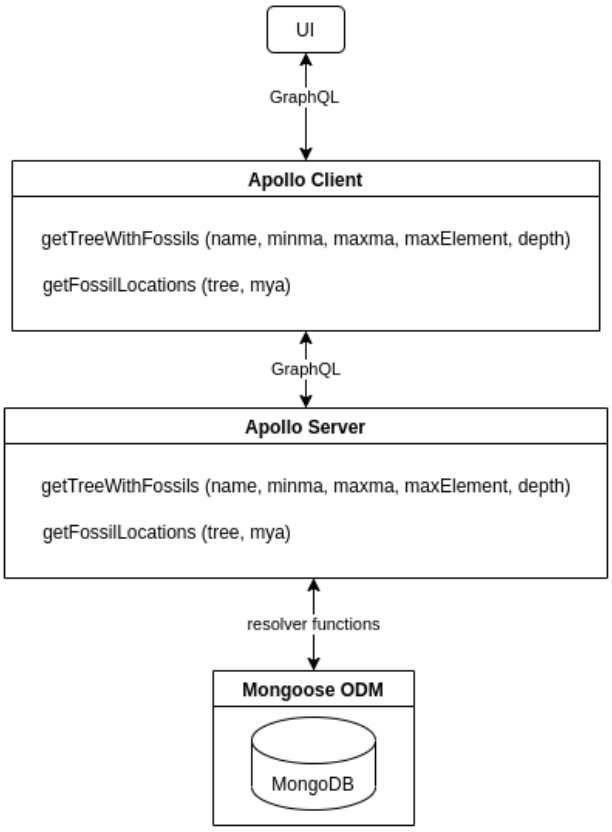


Figure 5.2: The API implementation. In short, Apollo Client handles GraphQL queries triggered by user action from UI. The Apollo Server then receives the GraphQL queries, which are in turn processed by resolver functions that retrieve data from Mongoose modeled data that is stored in MongoDB. The data is finally returned from the database all the way to the UI.

5.4.2 Front-end

The front-end design from section 4.5 is realized in a React framework. Conceptually, the front-end consists of the global states and the user interface, as seen in Figure 4.1 in chapter 4. The implementation of global states are listed in Figure 5.3. All global states can be classified into three categories. For time parameters, *mya* records the time point at which to show the world map and fossil locations, while *maxma* and *minma* together define the time period to retrieve the fossils and to form the tree of life. For tree parameters, *name* is the name of the root node of the tree of interest; *depth* and *maxElement* are the number of layers and the max number of siblings under each node when retrieving the tree of life. Finally, *mapFocusNode* and *treeFocusNode* are the tree node identified by *pathFromRoot* on which a user is hovering in map (users can highlight a tree node by hovering on a fossil point that is attached to it) and tree component, respectively. They are implemented to communicate the highlighting effect in between map and tree components (Figure 4.8 and Figure 4.10).

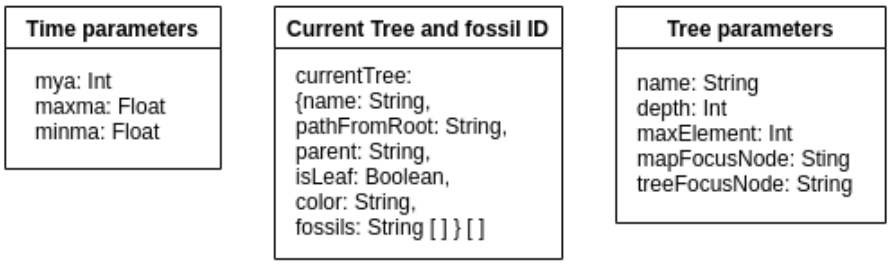


Figure 5.3: A list of all global states. The front-end's global states falls into three categories: time parameters, tree parameters and current tree and fossil ID. While time parameters and tree parameters include several variables of primitive types, current tree and fossil ID only has one variable, *currentTree*, that is an array of complex objects.

Next, we introduce how the components of the user interface are assembled and how they could interact. An overview of the user interface is shown in Figure 5.4. The user interface is presented by *App*, which has one *Header*, one *Footer* and one *TimeControl*. The *TimeControl* has *mya*, which is needed to show the position of the current time point in the scroll bar (Figure 4.3). By clicking on selectable a time period, both time point and the *maxma* (the start of the time period) and *minma* (the end of the time period) will be changed, triggering both a *getTreeWithFossils* query for the tree, and a *getFossilLocations* query for the map. Since both map and tree requires data traffic from the back-end, they both inherit from a more general *DatFetcher* component for higher code re-usability. *DataFetcher* is responsible for sending queries via *useQuery()* that passes the appropriate *queryVariables* depending on its *queryName* (*i.e.*, whether it is *getTreeWithFossils* or *getFossilLocations*). The exact *queryVariables* for each query is addressed in Figure 5.2. The *data* in *DataFetcher* hosts the returned data from the database, and is passed down to its children for further processing and rendering.

Tree receives the tree node data, and calculates the colors for each node, before updating the result in *CurrentTree*. *TreeGraph* renders the tree stored in *CurrentTree* in a sunburst graph. The *mapFocusNode* alerts the tree graph to provide a highlight effect for the node whose fossil a user is hovering on in the map. In the meantime, when a user hovers on a node in the tree graph, it can alert the map component to highlight its fossil point by *setTreeFocusNode()*. When users are intended to navigate into a node, by clicking on it from the graph or the breadcrumb above it (Figure 4.7), *setName* is called, which in turn triggers new queries from the database.

TreeSearchName renders the search bar above the tree graph (Figure 4.7). It keeps notes of what a user is typing in *searchNameBuffer*, and upon clicking on the search button, *setName()* updates the name of the node of interest, which also triggers new queries. *TreeGraphCustumize* renders the small control panel below the tree graph (Figure 4.7). It shows the current *maxElement* and *depth*, while updating these values when users change them in the drop down lists (Figure 4.9). When users are clicking either of the "Back to" buttons, *setName()* is responsible for changing the name of the new node and re-directing the users to a different tree.

Lastly, *map* renders the world map in GeoJSON, and the fossil locations returned to the *data* inherited from its parent *DataFetcher*. It acknowledges the colors of the fossil points from *CurrentTree*, and the fossil points to highlight (as a response to user hovering in the tree component) from *treeFocusNode*. *setMapData()* uses axios to load the correct GeoJSON map data according to the current *mya*, before storing it in *mapData* for rendering. When a user hovers on a fossil point, its attached node is updated by *setMapFocusNode()* to communicate this information to *TreeGraph*.

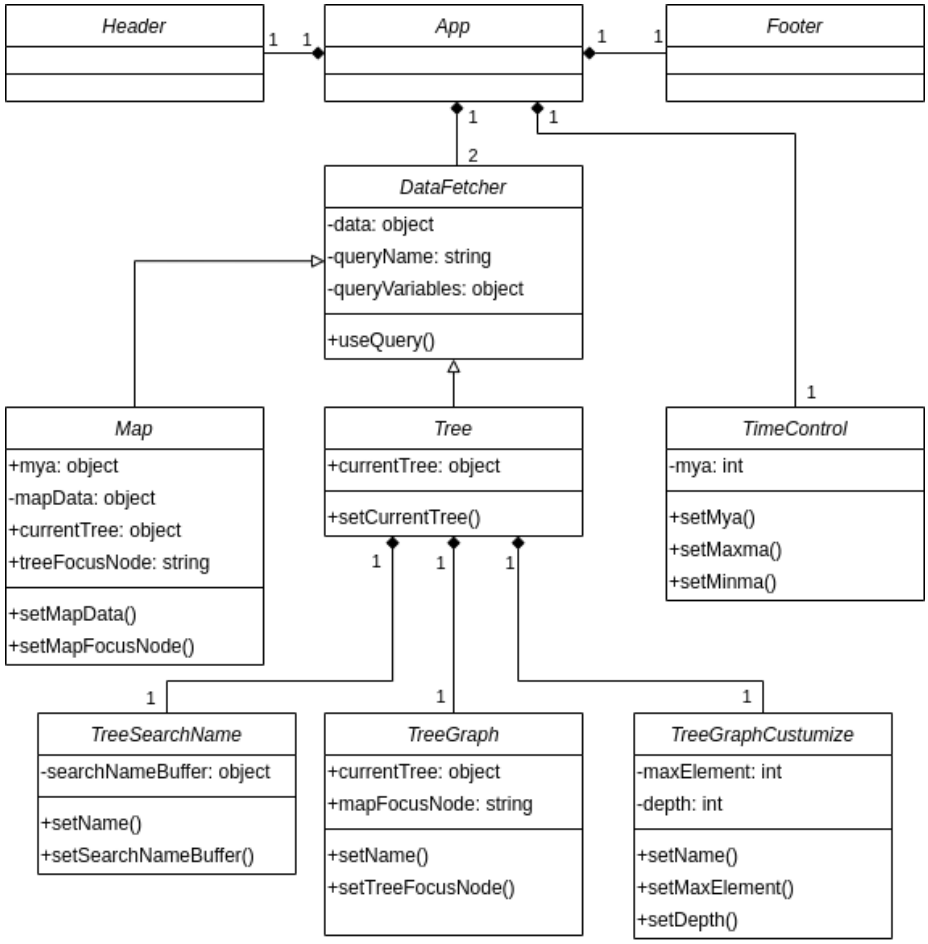


Figure 5.4: The implementation of the user interface by assembling different components. The user interface is the *App* components, which is assembled by a *Header*, a *Footer*, a *TimeControl* as well as *Map* and *tree*, which are both children of a more general *DataFetcher* component. The *Tree* is further broken down into three sub-components. For each component, the properties that belong to global states are marked with a "+", since they are accessible by all components.

6 Results

6.1 Data analysis

6.1.1 fossil data

6.1.2 tree data

6.2 Data visualization case studies

7 Conclusion

7.1 Limitations and Future Work

Acknowledgements

Bibliography

[Alh18] Sakinah SJ Alhadad. Visualizing data to support judgement, inference, and decision making in learning analytics: Insights from cognitive psychology and visualization science. *Journal of Learning Analytics*, 5(2):60–85, 2018.

[AMB+01] John Alroy, CR Marshall, RK Bambach, K Bezusko, M Foote, FT Fürsich, TA Hansen, SM Holland, LC Ivany, D Jablonski, et al. Effects of sampling standardization on estimates of phanerozoic marine diversification. *Proceedings of the National Academy of Sciences*, 98(11):6261–6266, 2001.

[AS98] Thomas J Algeo and Stephen E Scheckler. Terrestrial-marine teleconnections in the devonian: links between the evolution of land plants, weathering processes, and marine anoxic events. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 353(1365):113–130, 1998.

[BHP+12] Florian Block, Michael S Horn, Brenda Caldwell Phillips, Judy Diamond, E Margaret Evans, and Chia Shen. The deeptree exhibit: Visualizing the tree of life to facilitate informal learning. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2789–2798, 2012.

[BMG+11] James A Boyden, R Dietmar Müller, Michael Gurnis, Trond H Torsvik, James A Clark, Mark Turner, Hamish Ivey-Law, Robin J Watson, and John S Cannon. Next-generation plate-tectonic reconstructions using gplates. 2011.

[Bow89] Peter J Bowler. *Evolution: the history of an idea*. Univ of California Press, 1989.

[BVB+13] Michelle A Borkin, Azalea A Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister. What makes a visualization memorable? *IEEE transactions on visualization and computer graphics*, 19(12):2306–2315, 2013.

[CHU07] Chun-houh Chen, Wolfgang Karl Härdle, and Antony Unwin. *Handbook of data visualization*. Springer Science & Business Media, 2007.

[Cog03] JP Cogné. Paleomac: a macintosh™ application for treating paleomagnetic data and making plate reconstructions. *Geochemistry, Geophysics, Geosystems*, 4(1), 2003.

[Cou02] Vincent Courtillot. *Evolutionary catastrophes: the science of mass extinction*. Cambridge University Press, 2002.

[CWM+17] Ignazio Carbone, James B White, Jolanta Miadlikowska, A Elizabeth Arnold, Mark A Miller, Frank Kauff, Jana M U'Ren, Georgiana May, and François Lutzoni. T-bas: Tree-based alignment selector toolkit for phylogenetic-based placement, alignment downloads and metadata visualization: an example with the pezizomycotina tree of life. *Bioinformatics*, 33(8):1160–1168, 2017.

[dV16] Damien M de Vienne. Lifemap: exploring the entire tree of life. *PLoS biology*, 14(12):e2001624, 2016.

[EM13] Stephanie Evergreen and Chris Metzner. Design principles for data visualization in evaluation. *New directions for evaluation*, 2013(140):5–20, 2013.

[GTZ+12] Michael Gurnis, Mark Turner, Sabin Zahirovic, Lydia DiCaprio, Sonja Spasojevic, R Dietmar Müller, James Boyden, Maria Seton, Vlad Constantin Manea, and Dan J Bower. Plate tectonic reconstructions with continuously closing plates. *Computers & Geosciences*, 38(1):35–42, 2012.

[HE11] Christopher Healey and James Enns. Attention and visual memory in visualization and computer graphics. *IEEE transactions on visualization and computer graphics*, 18(7):1170–1188, 2011.

[HW12] Steve Haroz and David Whitney. How capacity limits of attention influence information visualization effectiveness. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2402–2410, 2012.

[IK01] Laurent Itti and Christof Koch. Computational modelling of visual attention. *Nature reviews neuroscience*, 2(3):194–203, 2001.

[JM19] Nikola Jovanovic and Alexander S Mikheyev. Interactive web-based visualization and sharing of phylogenetic trees using phylogeny. io. *Nucleic acids research*, 47(W1):W266–W269, 2019.

[JNZ+16] Julia Janicki, Nitish Narula, Matt Ziegler, Benoit Guénard, and Evan P Economo. Visualizing and interacting with large-volume biodiversity data using client–server web-mapping applications: The design and implementation of antmaps. org. *Ecological Informatics*, 32:185–193, 2016.

[JP08] Gregory E Jordan and William H Piel. Phylowidget: web-based visualizations for the tree of life. *Bioinformatics*, 24(14):1641–1642, 2008.

[KF08] Matthew J Kohn and Theodore J Fremd. Miocene tectonics and climate forcing of biodiversity, western united states. *Geology*, 36(10):783–786, 2008.

[Lac83] David Lack. *Darwin's finches*. CUP Archive, 1983.

[LB19] Ivica Letunic and Peer Bork. Interactive tree of life (itol) v4: recent updates and new developments. *Nucleic acids research*, 47(W1):W256–W259, 2019.

[LBC+08] Zheng-Xiang Li, SVb Bogdanova, AS Collins, Anthony Davidson, Bert De Waele, RE Ernst, ICW Fitzsimons, RA Fuck, DP Gladkochub, J Jacobs, et al. Assembly, configuration, and break-up history of rodinia: a synthesis. *Precambrian research*, 160(1-2):179–210, 2008.

[LDG+16] Fabien Leprieur, Patrice Descombes, Théo Gaboriau, Peter F Cowman, Valeriano Parravicini, Michel Kulbicki, Carlos J Melián, Charles N De Santana, Christian Heine, David Mouillot, et al. Plate tectonics drive tropical reef biodiversity dynamics. *Nature Communications*, 7(1):1–8, 2016.

[LMD+19] Lu Liang, Chunfeng Ma, Tengfei Du, Yufei Zhao, Xiaoyong Zhao, Mengmeng Liu, Zhonghua Wang, and Jianping Lin. Bioactivity-explorer: a web application for interactive visualization and exploration of bioactivity data. *Journal of cheminformatics*, 11(1):1–6, 2019.

[MCP+19] Kerrin Mendler, Han Chen, Donovan H Parks, Briallen Lobb, Laura A Hug, and Andrew C Doxey. Annotree: visualization and exploration of a functionally annotated microbial tree of life. *Nucleic acids research*, 47(9):4442–4448, 2019.

[Mid20] Stephen R Midway. Principles of effective data visualization. *Patterns*, page 100141, 2020.

[MMZ+16] Kara J Matthews, Kayla T Maloney, Sabin Zahirovic, Simon E Williams, Maria Seton, and R Dietmar Mueller. Global plate boundary evolution and kinematics since the late paleozoic. *Global and Planetary Change*, 146:226–250, 2016.

[MOAF05] Luis Manuel Matias, Jean-Louis Olivet, Daniel Aslanian, and Luis Fidalgo. Placa: a white box for plate reconstruction and best-fit pole determination. *Computers & geosciences*, 31(4):437–452, 2005.

[NS71] David Noton and Lawrence Stark. Scanpaths in saccadic eye movements while viewing and recognizing patterns. *Vision research*, 11(9):929–IN8, 1971.

[PM16] Shanan E Peters and Michael McClennen. The paleobiology database application programming interface. *Paleobiology*, 42(1):1–7, 2016.

[RH12] James Rosindell and Luke J Harmon. Onezoom: a fractal explorer for the tree of life. 2012.

[RLM10] Lorenz Rogge, Christian Lipski, and Marcus Magnor. Visualization of the continental drift in real-time. 2010.

[Sch95] Roger C Schank. What we learn when we learn by doing. 1995.

[SHV+13] GM Stampfli, C Hochard, C Vérard, C Wilhem, et al. The formation of pangea. *Tectonophysics*, 593:1–19, 2013.

[SI90] Hikmet Senay and Eve Ignatius. *Rules and principles of scientific data visualization*. Institute for Information Science and Technology, Department of Electrical . . . , 1990.

[SKR+07] Martin Smith, Joe Kurtz, Simon Richards, Margaret Forster, Gordon Lister, et al. A re-evaluation of the breakup of south america and africa using deformable mesh reconstruction software. 2007.

[SWW10] Stefan Steiniger, Robert Weibel, and B Warf. Gis software: a description in 1000 words. 2010.

[TGM83] Edward R Tufte and Peter R Graves-Morris. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.

[TS99] Trond Helge Torsvik and Mark Andrew Smethurst. Plate tectonic modelling: virtual reality with gmap. *Computers & Geosciences*, 25(4):395–402, 1999.

[Wil63] J Tuzo Wilson. Continental drift. *Scientific American*, 208(4):86–103, 1963.

[WMLW12] Simon E Williams, R Dietmar Muller, Thomas CW Landgrebe, and J Whittaker. An open-source software environment for visualizing and refining plate tectonic reconstructions using high-resolution geological and geophysical data sets. *GSA today*, 22(4-5):4–9, 2012.

[ZFP17] Andrew Zaffos, Seth Finnegan, and Shanan E Peters. Plate tectonic regulation of global marine animal diversity. *Proceedings of the National Academy of Sciences*, 114(22):5653–5658, 2017.

List of Figures

List of Tables