Visualization

Winter Semester 2021/2022

Prof. Tobias Günther

Friedrich-Alexander-Universität
Technische Fakultät

Programming 1 – The Basics

Am I able to solve the programming exercises without prior experience in programming?

No, most likely not. Note that the programming exercises are not needed to reach a top score in the exam.

Introduction

This is the first of the four programming exercises for this semester. On the second and the fourth exercise you will be able to score four points plus one bonus point from an optional task. The first and the third exercise will both contain tasks worth five points. In total, you will be able to score 18 (+2 bonus points). You need 12/18 points to get the bonus for the exam. Each exercise will be graded by the TAs **during** the exercise session **before/on** the due date (check the schedule for due dates). To submit your exercise you must present your visualization to one of the TAs during the exercise sessions. The TAs will also ask a couple of questions to ensure that you did the exercise on your own and fully understand how it works. You may work together as a group of up to three students but make sure that each of you understands all the elements that you've added to the visualization since the TAs will be asking questions individually.

For all of the four exercises you will be working on the same visualization framework, and you will continuously add more elements which will form a full interactive visualization in the end. The visualization allows users to explore the missing migrants data set. Please note that most of the tasks (except the last exercise) are independent of each other. This allows you to skip certain tasks completely but you might end up with an incomplete visualization and fewer points in total.

Last year the whole exercise was just one single html file. This file got quite big in the end which made navigating it quite difficult. For this year we instead decided to split up the files into multiple smaller ones but this requires setting up a small web-server on your system (more below in Section Preparation). Let us know what you think about the file structure!

Task 1: Preparation (0 Pts)

From our StudOn page download the skeleton.zip for the exercise. After extracting it, you will find a number of files. Save the files to a folder somewhere you can remember because this folder will be your workspace for this years exercise. You can use your favorite file editor or sometimes even the browser (special setup needed) to edit your files.

Next, we have to setup a small web-server to run our exercise. For this you will need python. Most of you already have done so in previous courses but if not you may install it from the official python website. Now, open a terminal in your workspace folder and run python -m http.server. This will start a minimal webserver which you can access from your browser. For this simply navigate to http://localhost:8000/index.html. You should see 1.

In the files you will find `TODOs` which are comments in the source code sometimes starting with // or wrapped in <! – – – – >. They always start with a reference to the task they belong to, for example: `TODO 4.1: Instructions go here`. When comments are indented, it usually means that something is

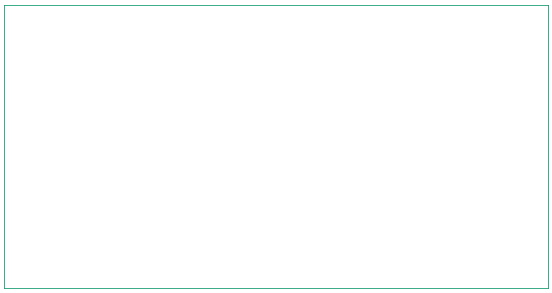

Fig. 1: Before.

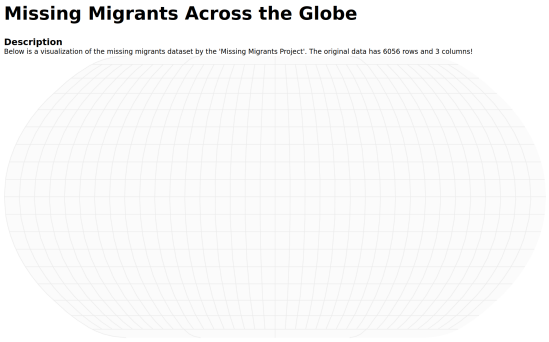

Fig. 2: After.

added to or nested within something that was added before. We recommend to remove the `TODOs` that you have completed for each task to keep track of the things that you still have to do.

While working, it is good practice to continuously refresh your browser and check the browser console if something went wrong. To refresh you often need to do a "hard refresh". A simple search on the web should tell you how to do this in your browser of choice. Do the same to find out how to open the console in your browser!

Task 2: Getting familiar with HTML and React (1.0 Pts)

In this first part you have to insert a title and a description into your visualization. The title is static and can be added as a simple `html` element. You will find a `TODO` in *index.html* showing where to add the title. You can use *Missing Migrants Across the Globe* as the title or come up with your own title.

We also want to add a description to the visualization. Later we would like to add some dynamic content to this text. So, instead of using a static html element we will use a react component. Check the *static_content.js* file. You will already find a number of components which are responsible for drawing static content in our visualization. Later, we will implement all of them but for now we will focus on the **Introduction** component.

The new component will return two *div* elements. One will contain a subtitle for the description and the other the description text. Both of them will be placed inside a react fragment which is necessary whenever we return more than one `html` tag from a component. Both of them require a `className` attribute to be able to style the text. You will find the values of these attributes and more detailed instruction in the `TODO`. Next, we have to add the variable which will contain our introduction text. This variable will be inserted into our second div element but we have to wrap the variable name into curly braces to make sure React and JSX understand that the content of the variable should be inserted here. Again you can find additional information in the `TODOs`. To be able to see the text, you have to add the component to the `App` component in the same way as the `WorldGraticule` components was added. You should now be able to see the title, the subtitle and the description in your browser.

This task is now complete. Feel free to style the title and the description the way you like it by checking out styling information in the *style.css* file. You can try to color the title, change the font, etc.. The code inside the file is called CSS and is quite powerful. If you don't know how to do certain things you can quickly search it on the internet. There are great resources out there which describe the capabilities of CSS.

Task 3: Data Loading (1.5 Pts)

In this task we need to load the data that we will work with throughout the upcoming exercises. We require two data sets. The first is a JSON data set which describes the topology of the world and the borders of countries. The second contains the actual data we are visualizing and is stored in CSV format. Both data sets are loaded in a separate function inside the file *data_loading.js*. The `useWorldAtlas()` function, that loads the topology data, is already given to you right above the code for loading the CSV data.

Have a look at it and try to understand whats going on. The way we will load the CSV data is quite similar.

To load the CSV data, we already defined the `useData()` function for you, which returns the data. For efficient data loading, we have to use `React.useState` and `React.useEffect`. The first allows us to define a state which can be modified later with a set function. This way, the data can be downloaded in the background and set at a later point in time. The second, allows us to ensure that the data downloading is only performed once. In this case we need to use `d3.csv` because we are working with CSV data. When the download is finished, the function defined inside the `.then` function call will be invoked which we will use to call the set function of our previously defined state. Check the `TODO`s to complete the `useData()` function!

To make sure we can work with the data, we need to first apply some transformations on it. For example we have to transform the number of dead and missing migrants from a string to a number and the date string to a Date object. Furthermore, we have to split up the coordinates string such that we have two individual numbers to work with. You will find detailed instructions in the source code. The last thing we have to do is to add a special case in the `App` component whenever the data has not been loaded yet.

Task 4: Data Metrics (0.5 Pts)

Lets use the data to write some dynamic content to the introduction. Add anything you like to the introduction. For example number of rows and columns in the data. To be able to access the data from within the `Introduction` component you have to pass it to the component. For reference you can check how width and height are passed to the `Histogram` and `WorldGraticule` component.

Task 5: World Sphere and Graticule (2.0 Pts)

In this task we will draw a background for our visualization to provide users with context of where certain events occured. For this, we draw a sphere and graticules on it which will represent the earth for now.

To start, we must define a projection which tells us how a spherical object (the Earth) will be mapped to a 2D plane. `d3.geo` contains a number of useful projection methods that can transform longitude/latitude coordinates to pixel coordinates. Given a projection method, we will first define a path generator that takes a d3 object as input and transforms it to pixel coordinates. We will use this path generator for everything to be drawn on the map.

In the script, find the `TODO` that says "`TODO: define a projection`". Use `d3.geoNaturalEarth1()` to create a projection and assign the result of this function to a `const` variable called `projection`. Feel free to later try other projection methods, such as `d3.geoEquirectangular()` or `d3.geoMercator()`. We will be able to change the entire visualization just by changing this line of code! Afterwards, feed the `projection` object into `d3.geoPath( )`, and assign the result to a `const` variable called `path`. This path object may later receive d3 geometry and then transforms it using the given projection to image space. As a first visual element on our globe, we want to visualize a lon/lat grid. Thus, define a variable `const graticule` and assign the result of `d3.geoGraticule()` to it. Note that geoGraticule does not need the projection or the path element yet. This transformation will come later down below.

Next, we want to display the first elements on the map. Find the React component `WorldGraticule` and look into the group element `<g>` that it returns. Inside this group element, you find a number of `TODO`s. This is where we will create all the visual elements to be displayed. First, we will draw a simple sphere, using `<path className="sphere" d=path( type: 'Sphere' ) />` This will create a sphere. We draw the sphere using an SVG `<path>` element. The path element will be named "sphere", for stylization in the CSS `<style>` section. The data d of the path receives the projection of a default sphere.

Next, we will draw the graticules on top. In that same group element `<g>` insert a `<path>` as child element after the sphere, namely: `<path className="graticule" d=path(graticule()) />`. With this, we are doing two things. First, we give the path object a class name, which will be used in the CSS `<style>` section above to define the color of the lines. Second, we define the path geometry, for which we pass the graticule into the path generator that we defined above. When done correctly, you should see the grid lines on the sphere as shown in 2.