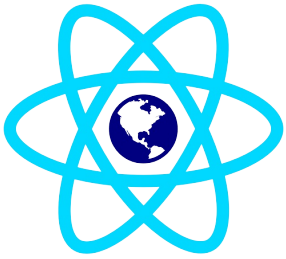
# Table of Contents



# Workshop *react-geo - mapping mit React*

Welcome to the workshop **react-geo - mapping mit React**. This workshop is designed to give you a comprehensive overview of react-geo as a library of geo-related application components available in combination with React, Ant Design and OpenLayers.

> If you want to visit this page on your own device or to print the PDF version, you can download the workshop materials here.

## Setup

The following instructions and exercises assume that you have some requirements fulfilled on your local machine. Please check if you have the consequent packages installed:

- A suitable text editor, e.g. the lightweight Atom editor.
- NodeJS in version 8 or higher.

All set? Then, lets' go!

## Overview

This workshop is presented as a set of modules. In each module you will perform tasks designed to achieve a specific goal for that module. Each module builds upon lessons learned in previous modules and is designed to iteratively build up your knowledge base.

- Basics - Dive into the basics of EcmaScript 6, React and npm.
- First steps - Learn how to create your own React app and how to include react-geo in it.
- react-geo components - Extend your application with some react-geo components.
- Higher order components / Provider - Have a look at more advanced components.

## Authors


- André Henn (henn@terrestris.de)
- Daniel Koch (koch@terrestris.de)
- Kai Volland (volland@terrestris.de)

# Basics

Before we get started with react-geo we have a short look at some basic stuff.

- NPM - Node / Node package manager
- ES6 - EcmaScript 6
- React - ReactJS

# npm

npm is the package manager for Node.js (a JavaScript runtime environment) and the world's largest software registry (more than 600k packages) with approximately 3 billion downloads per week.

You can use npm to:

- Adapt packages to your apps, or incorporate them as they are.
- Download standalone tools you can use right away.
- Run packages without downloading using npx.
- Share code with any npm user, anywhere.
- Restrict code to specific developers.
- Form virtual teams (orgs).
- Manage multiple versions of code and code dependencies.
- Update applications easily when underlying code is updated.
- Discover multiple ways to solve the same puzzle.
- Find other developers who are working on similar problems.

## package.json

The command `npm init` in your project folder opens an interactive dialogue to establish a npm project. The result is the `package.json` including all important settings, scripts and dependencies of your project.

```
{
  "name": "name_of_your_package",
  "version": "1.0.0",
  "description": "This is just a test",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "http://github.com/yourname/name_of_your_package.git"
  },
  "author": "your_name",
  "license": "ISC"
}
```

Please check the npm docs for further information.

## Install packages with npm

The most common way to install new packages with npm is via the CLI. To install a package simply type:

```
npm install packagename
```

You find the installed packages in the `node_modules` subfolder.

# Node version manager NVM

- bash script to manage multiple active node.js versions
- See here

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
nvm i v8
```

# ES6


ES (ECMAScript) is a trademarked scripting-language specification created to standardize JavaScript. As the name suggests, ES6 (later renamed to ES2015) is the sixth edition and came with significant new syntax for writing complex applications, including classes and modules. Some browsers do not (or only partially) support ES6, but the ES6 code can be transpiled in ES5, which enjoys a broader compability.

JavaScript frameworks and libraries to build modern web-applications are written in ES6.

### `import`

```
import { CircleMenu } from 'react-geo';
```

### `export`

```
const name = 'Peter';
export default name;
```

# Variable declaration

- ES5: `var`
- ES6: `var`, `let` and `const`:
  - scope dependent

## Function definition

```
// ES5
var myFunc = fucntion (myArg) {
  if (!myArg) {
    myArg = 'Peter'
  };
  return myArg + ' is the best arg!';
}
// ES6
const myFunc = (myArg = 'Peter') => {
  return myArg + ' is the best arg!';
} // myFunc() ----> 'Peter is the best arg!'
// ES6 shortened
const myFunc = myArg => myArg + ' is the best arg!';
```

## Template string

```
// ES5
var a = 1909;
```

```
console.log('Year: ' + a)
// ES6
console.log(`Year ${a}`)
```

## Destructuring assignment

See also here

*Example 1*: Object destructuring

```
// ES5
var obj = {
  name: 'Peter',
  age: 55
}
var age = obj.age;

// ES6
const obj = {
  name: 'Peter',
  age: 55
}
const {
  age
} = obj;
```

*Example 2* (also uses Spread operator):

```
// ES5
var user = {name: 'peter', age: 12};
user = Object.assign(user, {email: 'peter@love.de'});
// ES6
let user = {name: 'peter', age: 12};
user = {...user, email: 'peter@love.de'};
```

# React


React is a modern and open-source JavaScript library for building user interfaces based on ES6. Originally, it has been developed by a software engineer at Facebook and is still being maintained by Facebook (among others).

React allows developers to create large web-applications that use data and can change over time without reloading the page. It aims primarily to provide speed, simplicity, and scalability. React processes only user interfaces in applications. This corresponds to View in the Model-View-Controller (MVC) pattern, and can be used in combination with other JavaScript libraries or frameworks in MVC, such as AngularJS.

The smallest React example looks like this:

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);>
```

Check the docs and Tutorial for more information.

# Props

Props are the component's configurations that you pass to instances. They are received from the above component and immutable. For detailed information check Components and Props

# State

The state stores internal values of a component. It's a *serializable* representation of one point in time—a snapshot. The state can be manipulated within a component via `setState` . For detailed information check State and Lifecycle

# Lifecycle

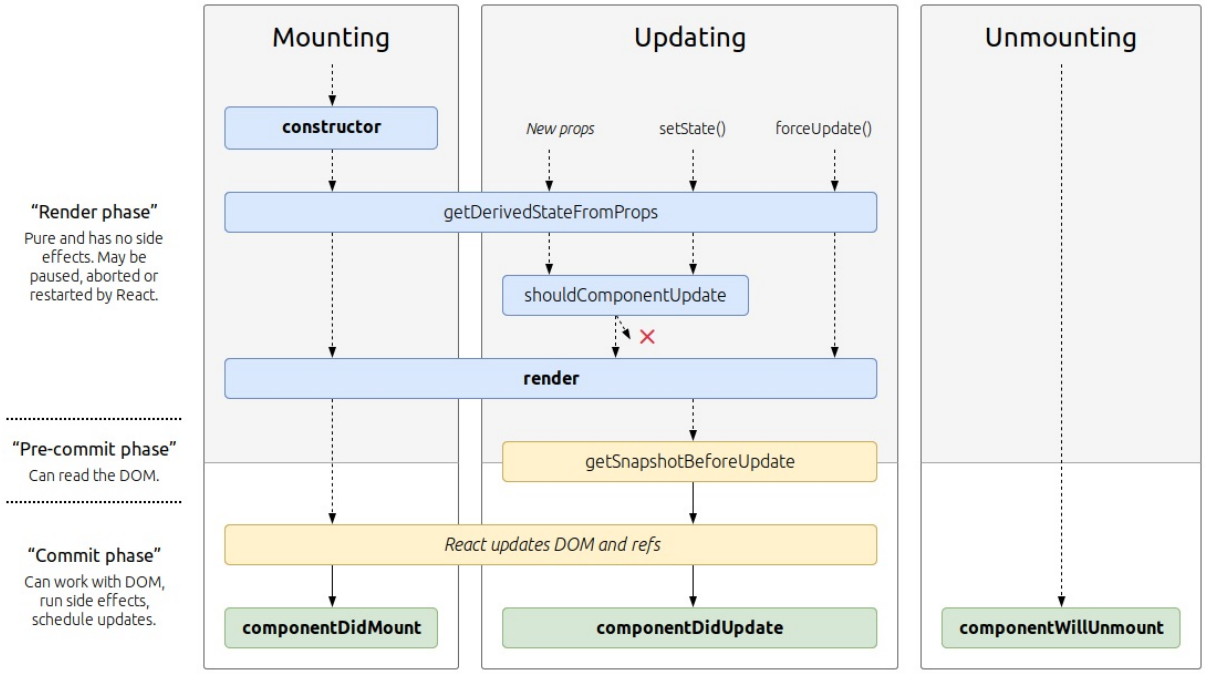



Image source, last accessed 01/03/2019

Check State and Lifecycle

# JSX

React components are typically written in JSX, a JavaScript extension syntax allowing quoting of HTML and using HTML tag syntax to render subcomponents. HTML syntax is processed into JavaScript calls of the React framework. Developers may also write in pure JavaScript. An example of JSX code:

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <p>Header</p>
        <p>Content</p>
        <p>Footer</p>
      </div>
    );
  }
}

export default App;
```

Syntactic sugar for React

# First steps

Now that we have set up our development setup and learned the basics about React and EcmaScript 6, we will start by creating a simple React based webapplication by the use of create-react-app, that will include a simple react-geo component. This application will be extended towards a fully functional mapping application little by little later on.

Content of this chapter:

# First steps

As a matter of course we could start this workshop by creating a React based webapplication by hand, but as you could imagine this would be a tough job for starters. So we want to dive into react-geo directly without the need to stick together all development tools to get a webapp running. Thankfully there is a project available, that we can use to generate an application for us (even without any configuration!): create-react-app.

Creating a new application is easy. Just navigate to a folder of your choice and create a new app named *my-app* inside this directory with:

```
npx create-react-app my-app
```

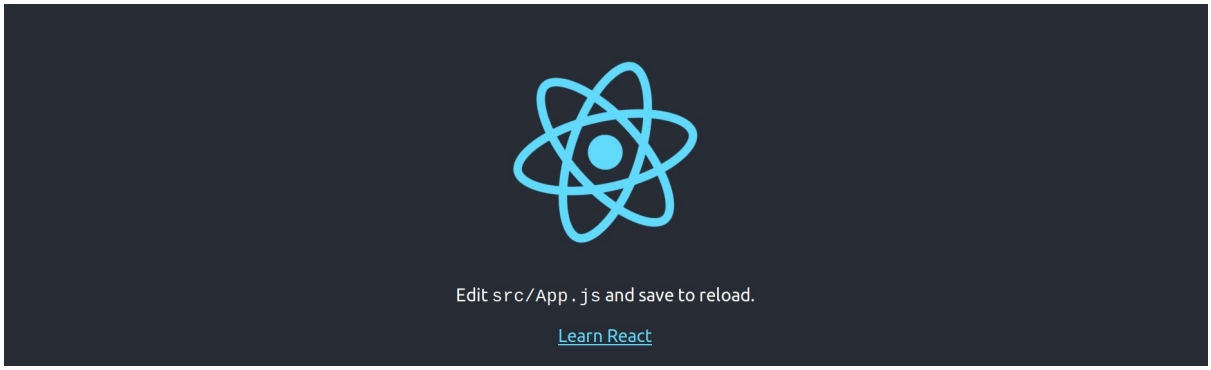
This will take a while, but finally you will see a list of commands you can run inside the created folder. Now switch to the project's folder with:

```
cd my-app
```

Finally we can start the development server with:

```
npm start
```

To view the application in your browser please open http://localhost:3000/.



# Development notes

`create-react-app` includes a webpack development server. This server allows you to 'hot deploy' your edited code. This means that your code changes will be immediatly visible in the browser.

- Edit src/App.js to modify the example… and let the magic happen!


Hallo Welt!

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Hallo Welt!</h1>
        </header>
        <p className="App-intro">
            Hallo Welt!
        </p>
      </div>
    );
  }
}

export default App;
```

# Include react-geo dependency

`react-geo` is published at https://www.npmjs.com/package/@terrestris/react-geo and can be integrated and installed in your *my-app* application via basic `npm` commands.

# Add react-geo dependency

To add the dependency `react-geo` please navigate to your project's folder (if not already done) and execute:

```
npm i @terrestris/react-geo --save
```

This will add the latest version of `react-geo` to your local `package.json` file (into the `dependencies` section) and download the distributed version of the library to the `node_modules` directory.

# Add Ant Design und OpenLayers dependencies

You may have noticed that the step from above has produced some warnings, which include `react-geo`:

```
npm WARN @terrestris/react-geo@11.0.0 requires a peer of antd@~3.0 but none is install
npm WARN @terrestris/react-geo@11.0.0 requires a peer of ol@~5.0 but none is installed
```

`npm` has three different types of dependencies:

### `dependencies`

dependencies are used to directly specify packages needed to *run* your application's code (e.g. a front-end library like Bootstrap)

### `devDependecies`

devDependencies are reserved to specify packages needed to *build* your application's code (e.g. test harnesses like Jest or transpilers like Babel).

### `peerDependencies`

However, under some conditions, one wants to express the *compatibility* of a certain package with the host package and npm calls this dependency a peerDependencies. Ususally this is used to express the dependency of a plugin inside this host package or similiar. In `react-geo` we need to have `antd`, `ol` and `react` defined as peer dependencies due to scope issues, because all of them were usually referenced by the host package/the application itself in a certain version.

As `npm` handles dependencies hierachically, including those packages in `react-geo` twice would lead to two different dependencies available in your application at runtime. To share the dependencies between your host application and `react-geo`, we advice `react-geo` to use the dependencies given by the host package.

To meet these requirements we have to install the requested peer dependencies by ourselves with:

```
npm i antd ol
```

Now we're ready to make use of all `react-geo` components and utilities inside our *my-app* application.

# Include a react-geo component

Now that we have `react-geo` installed, we can use it's components in the *my-app* application. For demonstration purposes we'll now add a simple button to the application.

Please open a text editor (if not already done) and open the file `App.js` from the `src` directory of your *my-app* application. Now import the `SimpleButton` class with the following statement:

```
import {
  SimpleButton
} from '@terrestris/react-geo';
```

The style definitions of `react-geo` and `antd` need to be imported as well:

```
import 'antd/dist/antd.css';
import './react-geo.css';
```

If the `react-geo.css` file is not yet located in the `src` directory please paste it from here.

> Please note that we are importing css files with the ES6 `import` here. This needs a properly configured css-loader. `create-react-app` includes this.

Now make use of the imported class by integrating it to the `render` method inside the `App` div (e.g. within the `<p>` -element):

```
<SimpleButton>
  Hello world!
</SimpleButton>
```

Save the file, open the application in your browser. You should see the changes directly, otherwise reopen http://localhost:3000. A blue button labelled with the text *Hello world* will be rendered.


Congratulations! You just created a complete React application including your first `react-geo` component with a few commands! We will now enhance the button to alert once it has been clicked by implementing a `onClick` callback function:

```
<SimpleButton
  onClick={() => {alert('Hello World!');}}
>
```

Save the changes and test the results by clicking on the button. You should now see a simple alert message with a *Hello World!* message.

We can also replace the text of the button with an icon. Simply remove the *Hello world* text and add the `icon` property with the value `"bars"` to the component.

```
<SimpleButton
  onClick={() => {alert('Hello World!');}}
  icon="bars"
/>
```

Voilá! You added a nice menu button to your app!

Your final solution should look like the following snippet:

```
import React, { Component } from 'react';
import logo from './logo.svg';

import './App.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import {
  SimpleButton
} from '@terrestris/react-geo';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Hello World!</h1>
        </header>
        <p className="App-intro">
          <SimpleButton
            onClick={() => {alert('Hello World!');}}
            icon="bars"
          />
        </p>
      </div>
    );
  }
}
```

# react-geo components

Currently (v5.6.2), react-geo provides a bunch of components that can be used for building Web-GIS applications. For example:

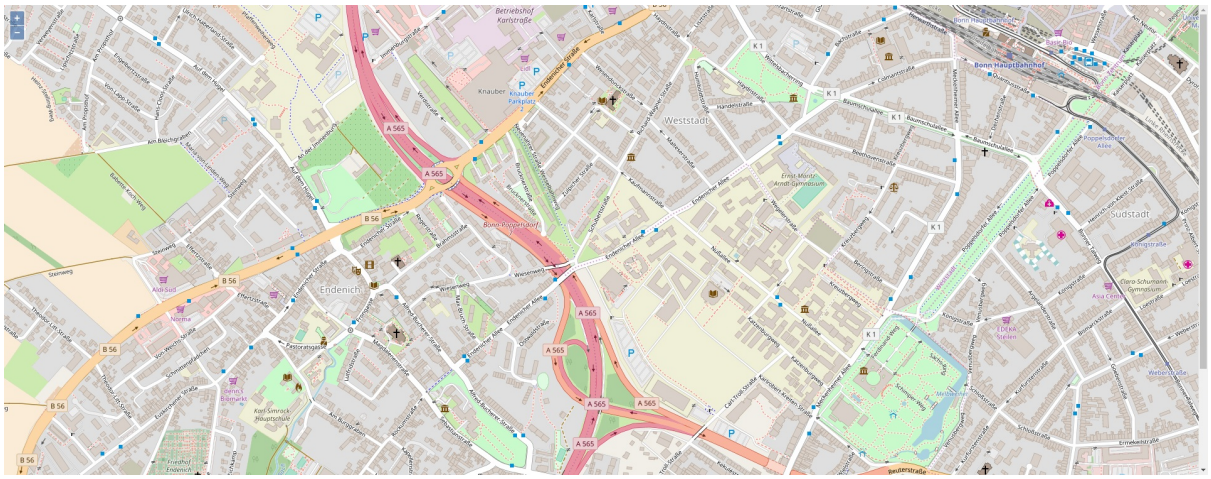
- Buttons (e.g. to en/disable map interactions)
- CircleMenu
- NominatimSearch
- ScaleCombo
- PropertyGrid
- FeatureGrid
- LayerTree
- Legend
- MapComponent
- Panel / Window
- Slider (LayerTransparency)
- Toolbar
- AddWMS-Container
- Utils
  - GeometryUtil (topological operations)
  - FeatureUtil
  - MapUtil
- HigherOrderComponents
  - VisibleComponent
  - MapProvider

You find the full feature list (also containing future components) here.

In the following, we'll have a closer look at some components in detail and show how they can be used in combination.

# MapComponent

Wrapper for an OpenLayers map. The `ol.map` is passed to the `MapComponent` as a prop.




- The map object can be shared across the whole application without passing it as prop to the whole render tree.
- The map can be created asynchronusly (using a Promise) so that every child of the MapProvider is just rendered when the map is ready.

- Documentation

**Task:** Add a map to your application. Use openstreetmap as tile layer.

```
import React, { Component } from 'react';

import './App.css';
import 'ol/ol.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import OlMap from 'ol/Map';
import OlView from 'ol/View';
import OlLayerTile from 'ol/layer/Tile';
import OlSourceOsm from 'ol/source/OSM';

import {
  MapComponent
} from '@terrestris/react-geo';

const layer = new OlLayerTile({
  source: new OlSourceOsm()
});

const center = [ 788453.4890155146, 6573085.729161344 ];

// create a new instance of ol.map in ES6 syntax
const map = new OlMap({
  view: new OlView({
    center: center,
    zoom: 16,
  }),
  layers: [layer]
});

map.on('postcompose', map.updateSize);

class App extends Component {
```

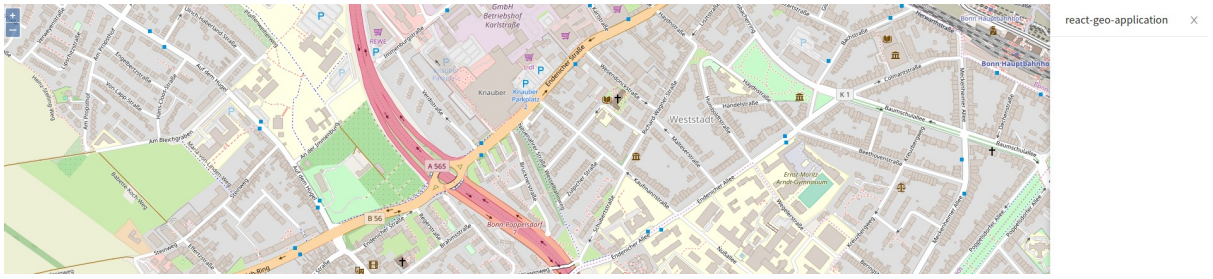
```
  render() {
    return (
      <div className="App">
        <MapComponent
          map={map}
        />
      </div>
    );
  }
}

export default App;
```

Beside the OpenLayers style the app needs its own stylesheet, e.g. to size the map.

```
html, body, #root, .App, #map {
  margin: 0;
  padding: 0;
  height: 100%;
  width: 100%;
}
```

# Drawer

Drawers are a nice method to add features in a visual appealing way. In this case we make use of the Drawer component provided by antd. This can be opened and closed with the SimpleButton we created previously.




**Task:** Add a drawer with the title `react-geo-application` to the right side of the app and let it open and close via a SimpleButton.

Your solution should look something like this:

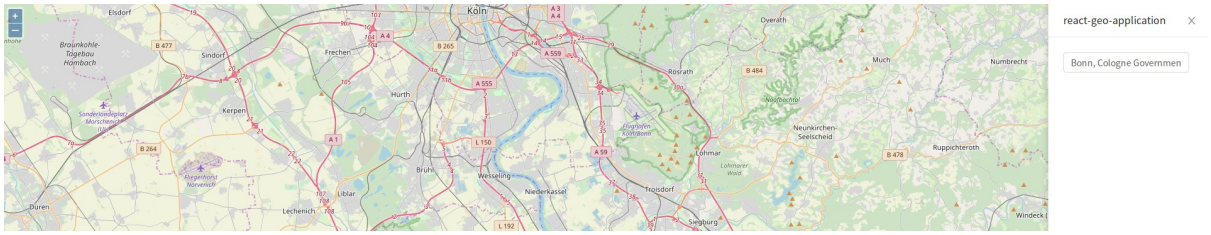
```
import React, { Component } from 'react';

import './App.css';
import 'ol/ol.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import OlMap from 'ol/Map';
import OlView from 'ol/View';
import OlLayerTile from 'ol/layer/Tile';
import OlSourceOsm from 'ol/source/OSM';

import { Drawer } from 'antd';
import {
  SimpleButton,
  MapComponent
} from '@terrestris/react-geo';

const layer = new OlLayerTile({
  source: new OlSourceOsm()
});

const center = [ 788453.4890155146, 6573085.729161344 ];

const map = new OlMap({
  view: new OlView({
    center: center,
    zoom: 16,
  }),
  layers: [layer]
});

map.on('postcompose', map.updateSize);

class App extends Component {
  state = {visible: false};

  toggleDrawer = () => {
    this.setState({visible: !this.state.visible});
  }

  render() {
    return (
```

```
      <div className="App">
        <MapComponent
          map={map}
        />
        <SimpleButton
          style={{position: 'fixed', top: '30px', right: '30px'}}
          onClick={this.toggleDrawer}
          icon="bars"
        />
        <Drawer
          title="react-geo-application"
          placement="right"
          onClose={this.toggleDrawer}
          visible={this.state.visible}
          mask={false}
        ></Drawer>
      </div>
    );
  }
}

export default App;
```

# NominatimSearch

NominatimSearch is a component that provides a search field querying nominatim search as geocoder. However, it is not limited to Nominatim search, see props `nominatimBaseUrl` .

- Documentation



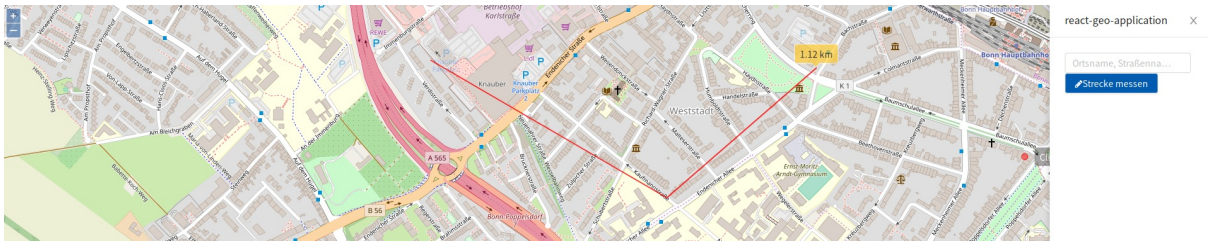

**Task:** Add the NominatimSearch component to the drawer.

```
import React, { Component } from 'react';

import './App.css';
import 'ol/ol.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import OlMap from 'ol/Map';
import OlView from 'ol/View';
import OlLayerTile from 'ol/layer/Tile';
import OlSourceOsm from 'ol/source/OSM';

import { Drawer } from 'antd';
import {
  SimpleButton,
  MapComponent,
  NominatimSearch
} from '@terrestris/react-geo';

const layer = new OlLayerTile({
  source: new OlSourceOsm()
});

const center = [ 788453.4890155146, 6573085.729161344 ];

const map = new OlMap({
  view: new OlView({
    center: center,
    zoom: 16,
  }),
  layers: [layer]
});

map.on('postcompose', map.updateSize);

class App extends Component {
  state = {visible: false};

  toggleDrawer = () => {
    this.setState({visible: !this.state.visible});
  }

  render() {
    return (
      <div className="App">
        <MapComponent
```

```
          map={map}
        />
        <SimpleButton
          style={{position: 'fixed', top: '30px', right: '30px'}}
          onClick={this.toggleDrawer}
          icon="bars"
        />
        <Drawer
          title="react-geo-application"
          placement="right"
          onClose={this.toggleDrawer}
          visible={this.state.visible}
          mask={false}
        >
          <NominatimSearch
            key="search"
            map={map}
          />
        </Drawer>
      </div>
    );
  }
}

export default App;
```

# MeasureButton

- Button (toggle) to en/disable certain `ol.interaction`s and `ol.layer`s to measure a distance, a polygonal area or angles
- Documentation




**Task:** Add a MeasureButton to the drawer.

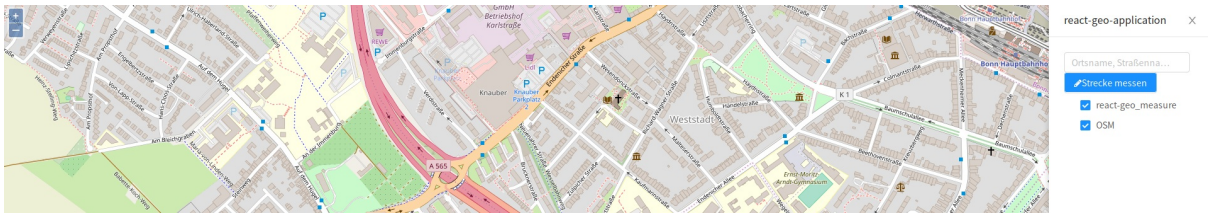
```
import React, { Component } from 'react';

import './App.css';
import 'ol/ol.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import OlMap from 'ol/Map';
import OlView from 'ol/View';
import OlLayerTile from 'ol/layer/Tile';
import OlSourceOsm from 'ol/source/OSM';

import { Drawer } from 'antd';
import {
  SimpleButton,
  MapComponent,
  NominatimSearch,
  MeasureButton
} from '@terrestris/react-geo';

const layer = new OlLayerTile({
  source: new OlSourceOsm()
});

const center = [ 788453.4890155146, 6573085.729161344 ];

const map = new OlMap({
  view: new OlView({
    center: center,
    zoom: 16,
  }),
  layers: [layer]
});

map.on('postcompose', map.updateSize);

class App extends Component {
  state = {visible: false};

  toggleDrawer = () => {
    this.setState({visible: !this.state.visible});
  }

  render() {
    return (
      <div className="App">
        <MapComponent
```

```
          map={map}
        />
        <SimpleButton
          style={{position: 'fixed', top: '30px', right: '30px'}}
          onClick={this.toggleDrawer}
          icon="bars"
        />
        <Drawer
          title="react-geo-application"
          placement="right"
          onClose={this.toggleDrawer}
          visible={this.state.visible}
          mask={false}
        >
          <NominatimSearch
            key="search"
            map={map}
          />
          <MeasureButton
            key="measureButton"
            name="line"
            map={map}
            measureType="line"
            icon="pencil"
          >
            Strecke messen
          </MeasureButton>
        </Drawer>
      </div>
    );
  }
}

export default App;
```

# LayerTree

- Tree component displaying the map layers in a hierarchical way
- Documentation

Autoconfigured with the topmost layer group (OpenLayers LayerGroup) of passed map.




**Task:** Add a basic LayerTree to the drawer.

**Hint:** To display the OSM Layer properly, add the `name` property to the `layer` variable.

```
<LayerTree
  map={map}
/>
```

The layer group (OpenLayers LayerGroup) can be passed as a prop as well.


**Task:** Add the layers *Food insecurity layer* and *World borders layer* as a layer group to the LayerTree.

**Steps:**

1. Create the layer group for the two layers
   - Food insecurity layer `'https://api.tiles.mapbox.com/v3/mapbox.20110804-hoa-foodinsecurity-3month.json?secure'`
   - World borders layer `'https://api.tiles.mapbox.com/v3/mapbox.world-borders-light.json?secure'`
2. Add the layer group to the layers of the map
3. Extend the LayerTree component with the property `layerGroup`

```
const layerGroup = new OlLayerGroup({
  name: 'Layergroup',
  layers: [
    new OlLayerTile({
      name: 'Food insecurity layer',
      minResolution: 200,
      maxResolution: 2000,
      source: new OlSourceTileJson({
        url: 'https://api.tiles.mapbox.com/v3/mapbox.20110804-hoa-foodinsecurity-3month.json?secure',
        crossOrigin: 'anonymous'
      })
    }),
    new OlLayerTile({
      name: 'World borders layer',
      minResolution: 2000,
      maxResolution: 20000,
```

```
      source: new OlSourceTileJson({
        url: 'https://api.tiles.mapbox.com/v3/mapbox.world-borders-light.json?secure',
        crossOrigin: 'anonymous'
      })
    })
  ]
});
```

```
<LayerTree
  layerGroup={layerGroup}
  map={map}
/>
```

**Complete Example:**

```
import React, { Component } from 'react';

import './App.css';
import 'ol/ol.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import OlMap from 'ol/Map';
import OlView from 'ol/View';
import OlLayerTile from 'ol/layer/Tile';
import OlSourceOsm from 'ol/source/OSM';
import OlSourceTileJson from 'ol/source/TileJSON';
import OlLayerGroup from 'ol/layer/Group';

import { Drawer } from 'antd';
import {
  SimpleButton,
  MapComponent,
  NominatimSearch,
  MeasureButton,
  LayerTree
} from '@terrestris/react-geo';

const layer = new OlLayerTile({
  source: new OlSourceOsm(),
  name: 'OSM'
});

const layerGroup = new OlLayerGroup({
  name: 'Layergroup',
  layers: [
    new OlLayerTile({
      name: 'Food insecurity layer',
      minResolution: 200,
      maxResolution: 2000,
      source: new OlSourceTileJson({
        url: 'https://api.tiles.mapbox.com/v3/mapbox.20110804-hoa-foodinsecurity-3month.json?secure',
        crossOrigin: 'anonymous'
      })
    }),
    new OlLayerTile({
      name: 'World borders layer',
      minResolution: 2000,
      maxResolution: 20000,
      source: new OlSourceTileJson({
        url: 'https://api.tiles.mapbox.com/v3/mapbox.world-borders-light.json?secure',
        crossOrigin: 'anonymous'
      })
    })
  ]
});
```

```
const center = [ 788453.4890155146, 6573085.729161344 ];

const map = new OlMap({
  view: new OlView({
    center: center,
    zoom: 16,
  }),
  layers: [layer, layerGroup]
});

map.on('postcompose', map.updateSize);

class App extends Component {
  state = {visible: false};

  toggleDrawer = () => {
    this.setState({visible: !this.state.visible});
  }

  render() {
    return (
      <div className="App">
        <MapComponent
          map={map}
        />
        <SimpleButton
          style={{position: 'fixed', top: '30px', right: '30px'}}
          onClick={this.toggleDrawer}
          icon="bars"
        />
        <Drawer
          title="react-geo-application"
          placement="right"
          onClose={this.toggleDrawer}
          visible={this.state.visible}
          mask={false}
        >
          <NominatimSearch
            key="search"
            map={map}
          />
          <MeasureButton
            key="measureButton"
            name="line"
            map={map}
            measureType="line"
            icon="pencil"
          >
            Strecke messen
          </MeasureButton>
          <LayerTree
            map={map}
            layerGroup={layerGroup}
          />
        </Drawer>
      </div>
    );
  }
}

export default App;
```

# Higher-order components

A higher-order component (HOC) is a function that takes a component and returns a new component.

A HOC is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are a pattern that emerges from React's compositional nature.

To learn more about HOCs visit https://reactjs.org/docs/higher-order-components.html.

Commonly used HOC are redux or react-i18next, for example.

`react-geo` provides currently:

- `mappify`
- `isVisibleComponent`
- `timeLayerAware`

# `MapProvider` / `mappify`

## `MapProvider`

- The `MapProvider` supplies the passed map to the React context (see also here) for the child elements.

## `mappify`

- The `mappify` HOC grabs a map object from the context (see also here) and passes it as a prop to the wrapped component
- Commonly used in combination with the `MapProvider`

**Task:** Update your app by mappifying your components.

```
import React, { Component } from 'react';

import './App.css';
import 'ol/ol.css';
import 'antd/dist/antd.css';
import './react-geo.css';

import OlMap from 'ol/Map';
import OlView from 'ol/View';
import OlLayerTile from 'ol/layer/Tile';
import OlSourceOsm from 'ol/source/OSM';
import OlSourceTileJson from 'ol/source/TileJSON';
import OlLayerGroup from 'ol/layer/Group';

import { Drawer } from 'antd';
import {
  SimpleButton,
  MapComponent,
  NominatimSearch,
  MeasureButton,
  LayerTree,
  MapProvider,
  mappify
} from '@terrestris/react-geo';

const MappifiedNominatimSearch = mappify(NominatimSearch);
const MappifiedMeasureButton = mappify(MeasureButton);
const MappifiedLayerTree = mappify(LayerTree);
const Map = mappify(MapComponent);

const layer = new OlLayerTile({
  source: new OlSourceOsm(),
  name: 'OSM'
});

const layerGroup = new OlLayerGroup({
  name: 'Layergroup',
  layers: [
    new OlLayerTile({
      name: 'Food insecurity layer',
      minResolution: 200,
      maxResolution: 2000,
      source: new OlSourceTileJson({
        url: 'https://api.tiles.mapbox.com/v3/mapbox.20110804-hoa-foodinsecurity-3month.json?secure',
        crossOrigin: 'anonymous'
      })
    }),
    new OlLayerTile({
```

```
      name: 'World borders layer',
      minResolution: 2000,
      maxResolution: 20000,
      source: new OlSourceTileJson({
        url: 'https://api.tiles.mapbox.com/v3/mapbox.world-borders-light.json?secure',
        crossOrigin: 'anonymous'
      })
    })
  ]
});

const center = [ 788453.4890155146, 6573085.729161344 ];

const map = new OlMap({
  view: new OlView({
    center: center,
    zoom: 16,
  }),
  layers: [layer, layerGroup]
});

map.on('postcompose', map.updateSize);

class App extends Component {
  state = {visible: false};

  toggleDrawer = () => {
    this.setState({visible: !this.state.visible});
  }

  render() {
    return (
      <div className="App">
        <MapProvider map={map}>
          <Map/>
          <Drawer
            title="react-geo-application"
            placement="right"
            onClose={this.toggleDrawer}
            visible={this.state.visible}
            mask={false}
          >
            <MappifiedNominatimSearch
              key="search"
            />
            <MappifiedMeasureButton
              key="measureButton"
              name="line"
              measureType="line"
              icon="pencil"
            >
              Strecke messen
            </MappifiedMeasureButton>
            <MappifiedLayerTree
              layerGroup={layerGroup}
            />
          </Drawer>
          <SimpleButton
            style={{position: 'fixed', top: '30px', right: '30px'}}
            onClick={this.toggleDrawer}
            icon="bars"
          />
        </MapProvider>
      </div>
    );
  }
}

export default App;
```

## `VisibleComponent` / `isVisibleComponent`

Wrapped components will be checked against the activeModules array of the state: If the wrapped component (identified by it's name) is included in the state, it will be rendered, if not, it won't.

```
import React from 'react';
import { render } from 'react-dom';
import { Button } from 'antd';
import { isVisibleComponent } from '../../index.js';

// Enhance (any) Component by wrapping it using isVisibleComponent().
const VisibleButton = isVisibleComponent(Button);

// The activeModules is a whitelist of components (identified by it's names) to
// render.
const activeModules = [{
  name: 'visibleButtonName'
}, {
  name: 'anotherVisibleButtonName'
}];

render(
  <div>
    <VisibleButton
      name="visibleButtonName"
      activeModules={activeModules}
      type="primary"
      shape="circle"
      icon="search"
    />
    <VisibleButton
      name="notVisibleButtonName"
      activeModules={activeModules}
      type="primary"
      shape="circle"
      icon="search"
    />
    <VisibleButton
      name="anotherVisibleButtonName"
      activeModules={activeModules}
      type="primary"
      shape="circle"
      icon="poweroff"
    />
  </div>,
  document.getElementById('exampleContainer')
);
```