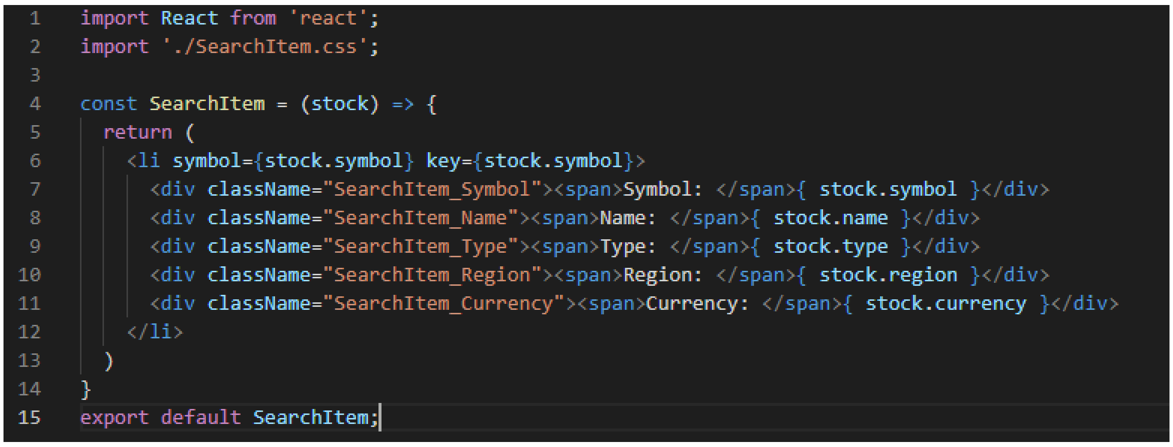
How have you split your application up into components? Are these components independent and reusable? How do you handle data flow and state management between your components?

For my web app project, I created a React app that allows users to search for different stock prices and compare their values.

With freedom that React provides, I had to face challenges with regards to decisions how to separate program into distinct sections and how to organize the code. I could simply implement the whole logic in 1 file - App.js, and style it another App.css, however, with the *Separation of Concern* and *Don't Repeat Yourself* principles in mind I split the application into different components. I would classify them into three different categories:

1. Wrapper component (parent component):
   - App (stateful component) – the container of all the other components, that is also responsible for client-server communication, processing data and responding to user events
2. Stateless components (responsible for displaying data; their logic revolves around the *props* they receive from the parent component) implement using an anonymous lambda function:
   - SearchBar – displays search bar for searching stocks
   - SearchItem – displays single search result
   - SearchResultList – displays all the search results (parent component of SearchItem)
   - StockTable – displays detailed data stocks that user wants to compare
   - Title – displays title of the app
   - LineChart – component from react-d3-component library

Example:



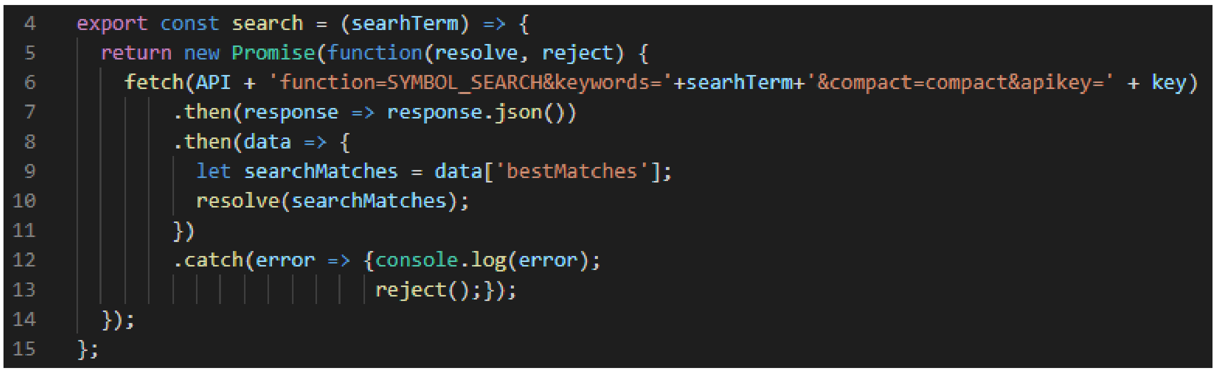
```
import React from 'react';
import './SearchItem.css';

const SearchItem = (stock) => {
  return (
    <li symbol={stock.symbol} key={stock.symbol}>
      <div className="SearchItem_Symbol"><span>Symbol: </span>{ stock.symbol }</div>
      <div className="SearchItem_Name"><span>Name: </span>{ stock.name }</div>
      <div className="SearchItem_Type"><span>Type: </span>{ stock.type }</div>
      <div className="SearchItem_Region"><span>Region: </span>{ stock.region }</div>
      <div className="SearchItem_Currency"><span>Currency: </span>{ stock.currency }</div>
    </li>
  )
}
export default SearchItem;
```

Figure 1. SearchItem component

3. Exportable functions to handle the communication with API (using fetch function)
   - GetStock – responsible for specific stock data
   - Search – responsible for getting best matching search results

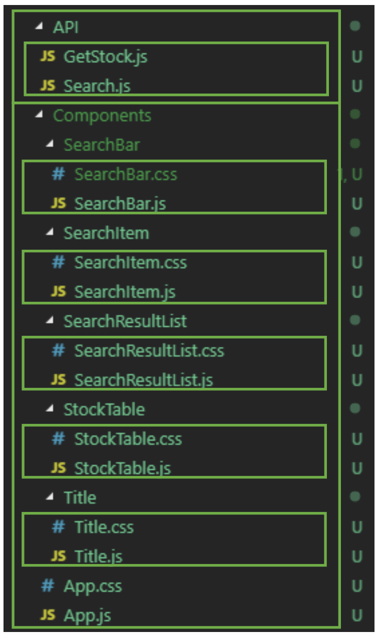
Example:



```
export const search = (searhTerm) => {
  return new Promise(function(resolve, reject) {
    fetch(API + 'function=SYMBOL_SEARCH&keywords='+searhTerm+'&compact=compact&apikey=' + key)
      .then(response => response.json())
      .then(data => {
        let searchMatches = data['bestMatches'];
        resolve(searchMatches);
      })
      .catch(error => {console.log(error);
                        reject();});
  });
};
```

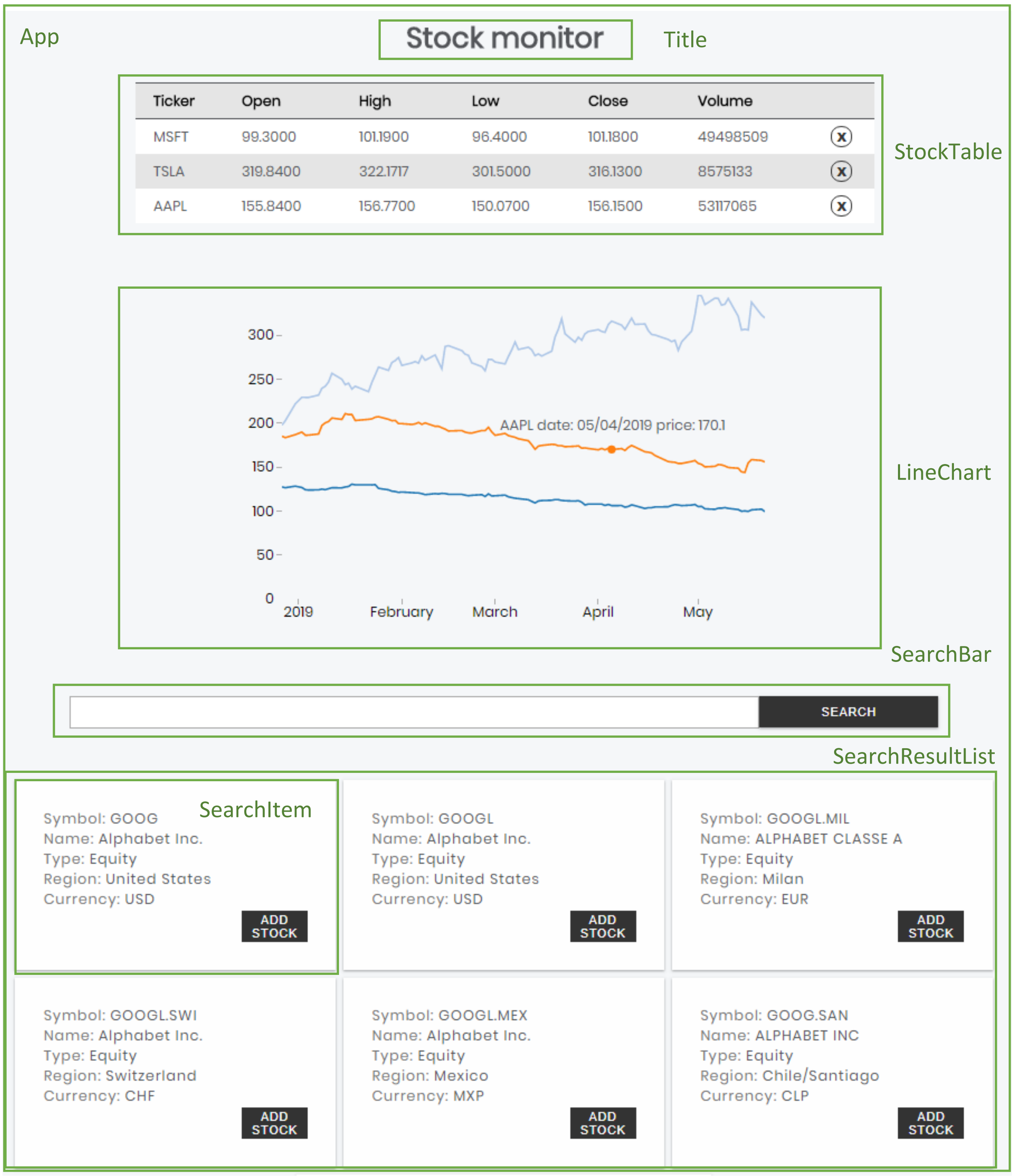
Figure 2. Search function

Each stateless components and App component have their own separate .css file responsible for styling.



```
API
  GetStock.js        U
  Search.js          U
Components
  SearchBar
    SearchBar.css    U
    SearchBar.js     U
  SearchItem
    SearchItem.css   U
    SearchItem.js    U
  SearchResultList
    SearchResultList.css  U
    SearchResultList.js   U
  StockTable
    StockTable.css   U
    StockTable.js    U
  Title
    Title.css        U
    Title.js         U
  App.css            U
  App.js             U
```

Figure 3. File structure



Ticker	Open	High	Low	Close	Volume	
MSFT	99.3000	101.1900	96.4000	101.1800	49498509	ⓧ
TSLA	319.8400	322.1717	301.5000	316.1300	8575133	ⓧ
AAPL	155.8400	156.7700	150.0700	156.1500	53117065	ⓧ

Symbol: GOOG
Name: Alphabet Inc.
Type: Equity
Region: United States
Currency: USD

Symbol: GOOGL
Name: Alphabet Inc.
Type: Equity
Region: United States
Currency: USD

Symbol: GOOGL.MIL
Name: ALPHABET CLASSE A
Type: Equity
Region: Milan
Currency: EUR

Symbol: GOOGL.SWI
Name: Alphabet Inc.
Type: Equity
Region: Switzerland
Currency: CHF

Symbol: GOOGL.MEX
Name: Alphabet Inc.
Type: Equity
Region: Mexico
Currency: MXP

Symbol: GOOG.SAN
Name: ALPHABET INC
Type: Equity
Region: Chile/Santiago
Currency: CLP

Figure 4. Division of Stock monitor App into components

The idea of using stateless functional components increases code readability, render-only components are simplified and through that optimization, it is possible to use far less memory, as no instance is created.

Additionally, as it comes to reusability, components like Title, SearchBar, LineChart or SearchItem could be easily reused in other parts of the application (while adding new features) as for following purposes:

- Title – for displaying tiles of other pages / sections of application
- SearchItem – for displaying detailed stock data in different context than in search result list

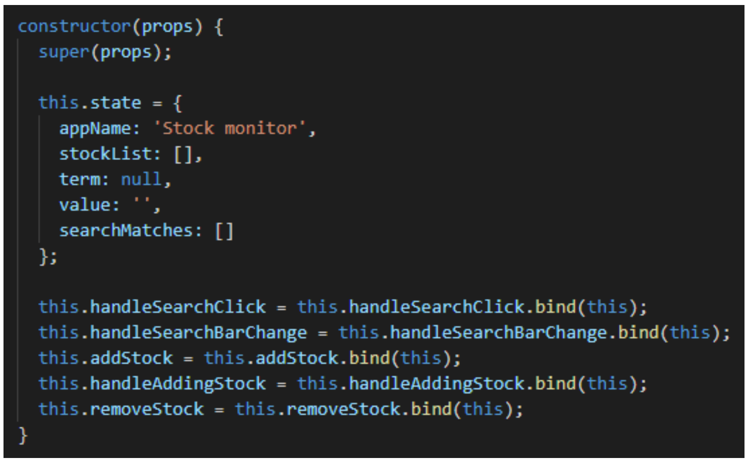
- SearchBar – for one field input form (not necessary for search, as button name and onClick functions are passed through properties)
- LineChart – displaying different set of data in form of the line chart

On the other side, StockTable and SearchResultList components, are not as generic as those listed above and they can be used only in context of displaying a table with stock details and as search result list, respectively.

Criticizing my application of *Separation of Concern* principle, the following changes could be applied:

- further division of the StockTable component into wrapper component, single table row component, and rounded button component; number of columns and their names could be more flexible with respect to the data passed through *props*
- creation of reusable button component that could be used both in the search bar and search result list

Regarding the data flow, in React we have unidirectional data flow. Higher-order parent components (e.g. the *App* component) have a container for the state of the app, and passes that state down to child components through read-only *props*.

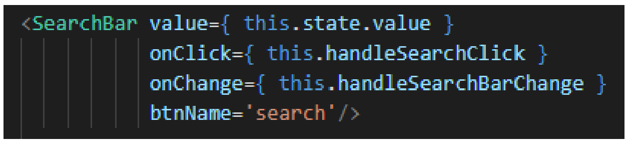


```
constructor(props) {
  super(props);

  this.state = {
    appName: 'Stock monitor',
    stockList: [],
    term: null,
    value: '',
    searchMatches: []
  };

  this.handleSearchClick = this.handleSearchClick.bind(this);
  this.handleSearchBarChange = this.handleSearchBarChange.bind(this);
  this.addStock = this.addStock.bind(this);
  this.handleAddingStock = this.handleAddingStock.bind(this);
  this.removeStock = this.removeStock.bind(this);
}
```

Figure 5. Constructor of the App with the state



```
<SearchBar value={ this.state.value }
           onClick={ this.handleSearchClick }
           onChange={ this.handleSearchBarChange }
           btnName='search'/>
```

Figure 6. Example of passing to the SearchBar (child component) input value, button name and functions handling click and input changes