SEBA Master: Web Application Engineering

Tutorial 2: Developing single-page web applications with React 16

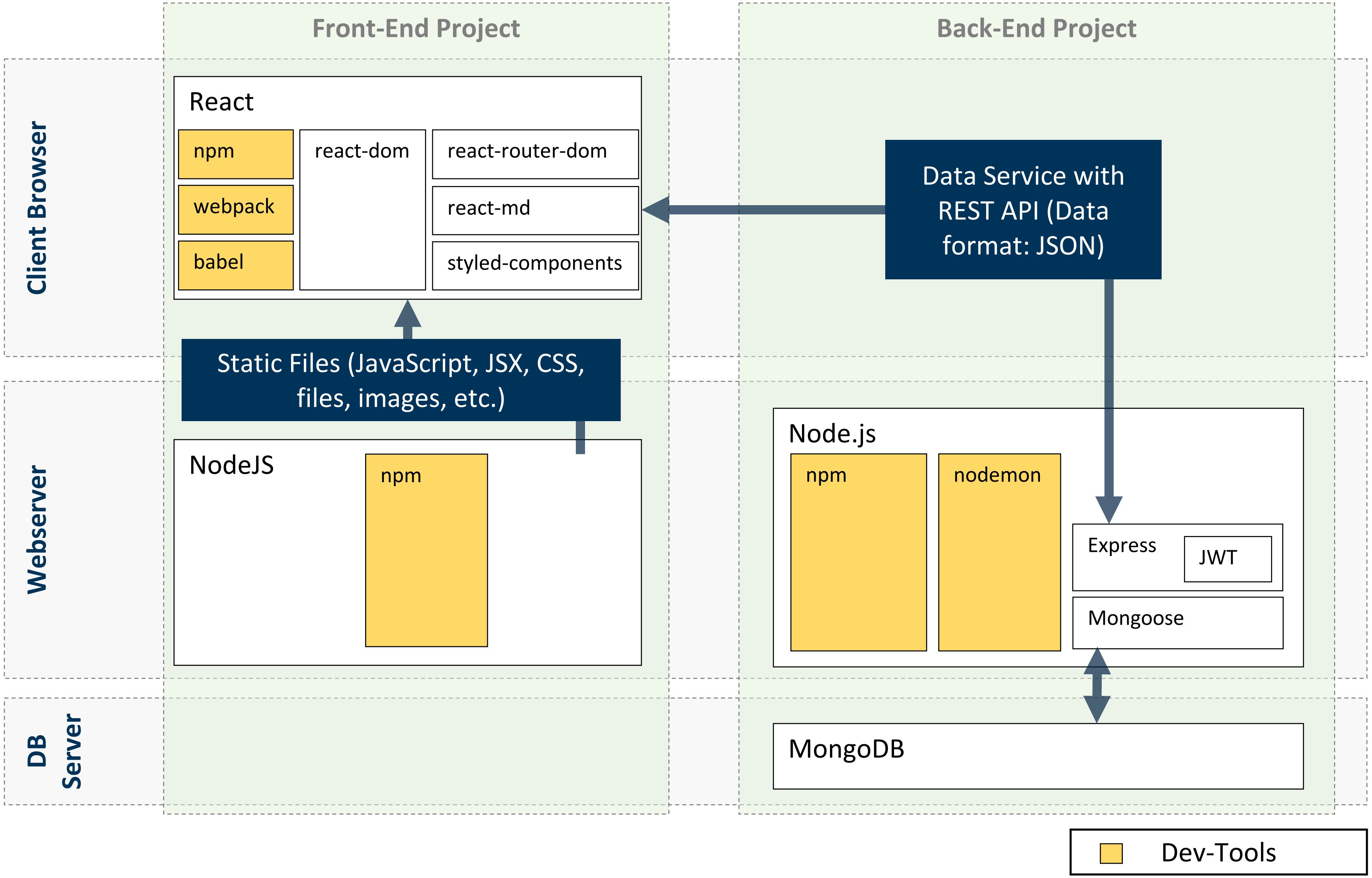
19th of May 2020, Ingo Glaser, Munich

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

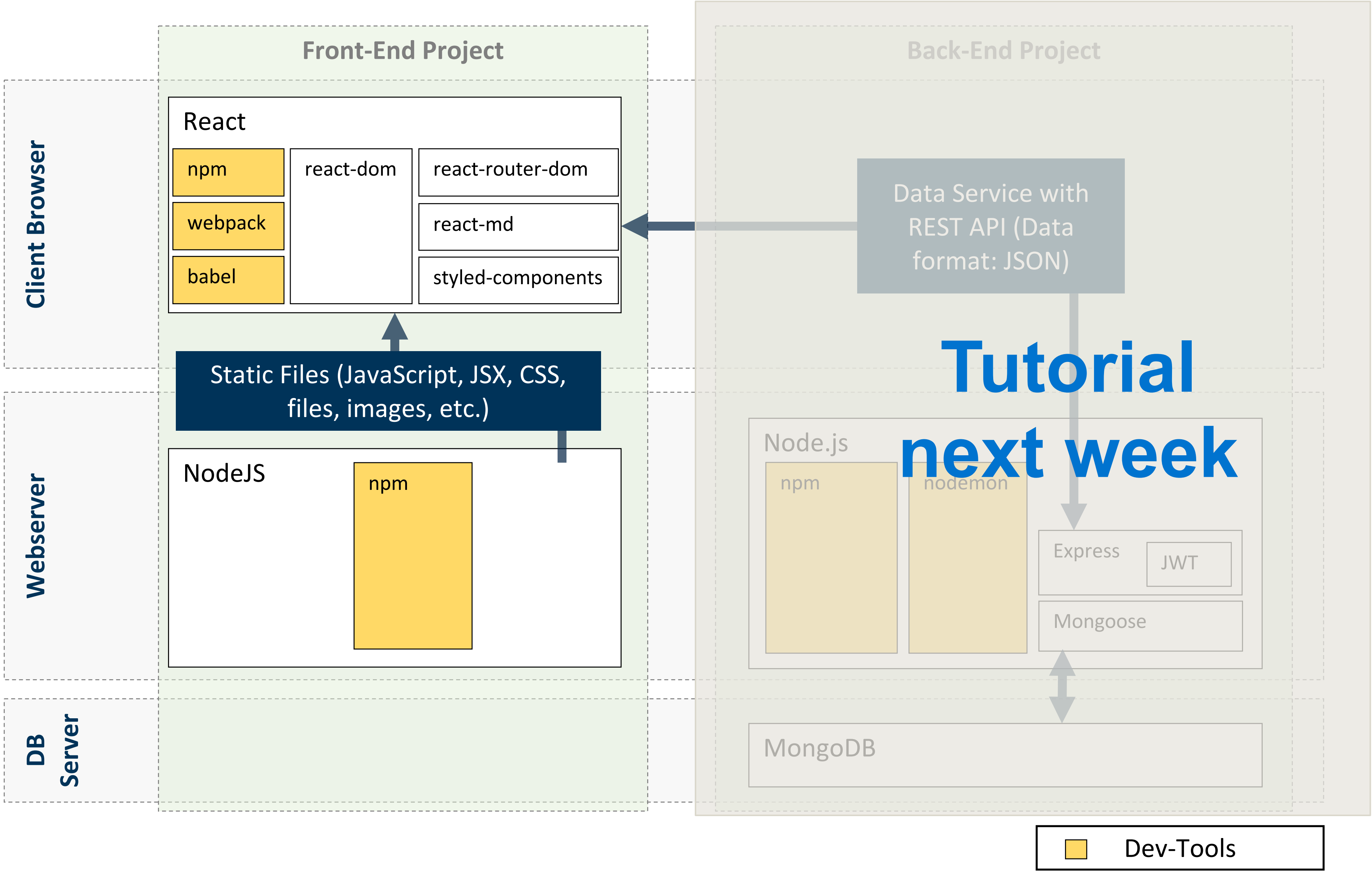
Outline

- NodeJS Configuration
- Webpack Configuration
- Frontend Architecture
- Coding Conventions
- Styled Components

Movie Application Overview

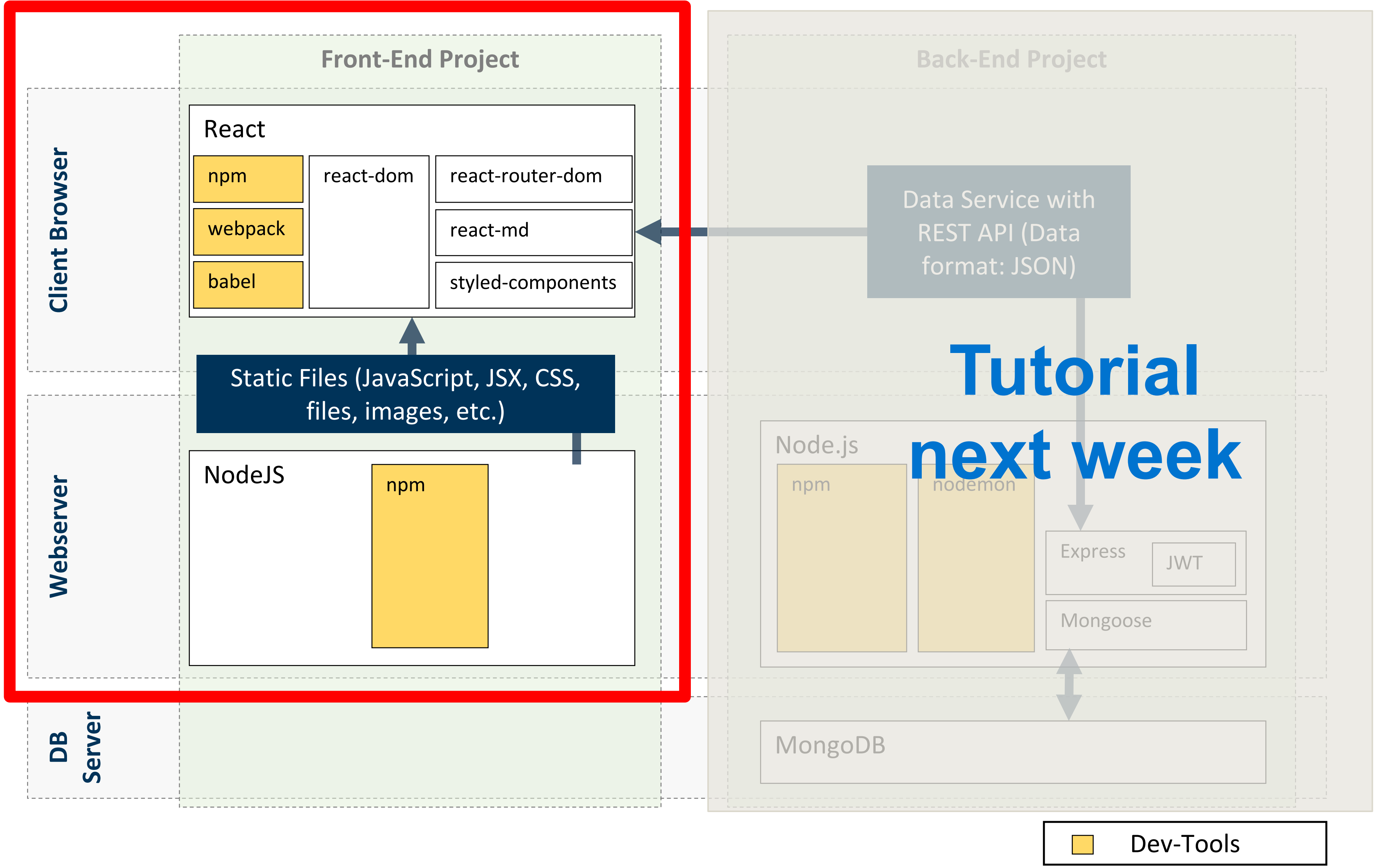


Movie Application Overview



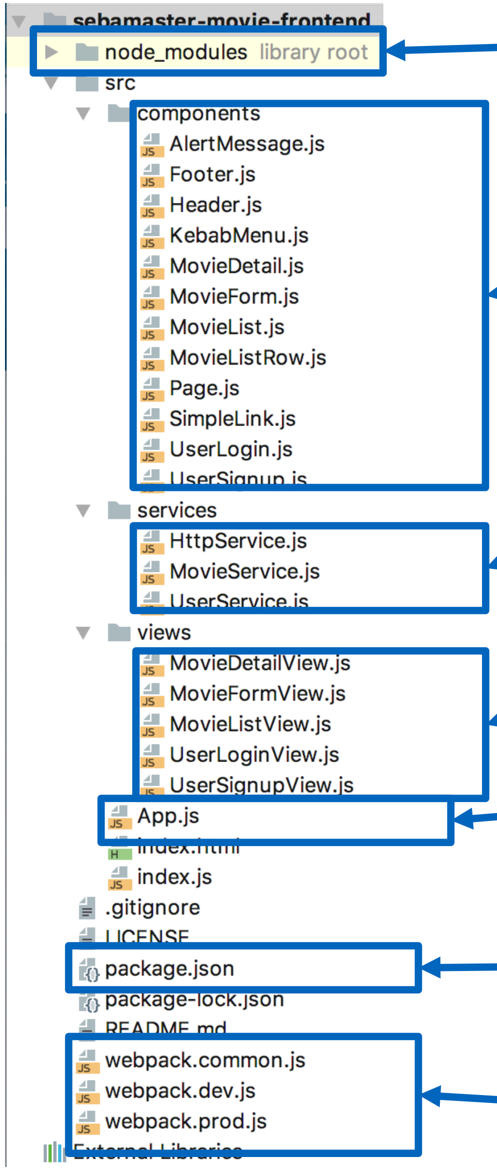
Tutorial next week

Movie Application Overview



Tutorial next week

File & Folder Structure



```
▼ sebamaster-movie-frontend
  ▶ node_modules  library root
  ▼ src
    ▼ components
        AlertMessage.js
        Footer.js
        Header.js
        KebabMenu.js
        MovieDetail.js
        MovieForm.js
        MovieList.js
        MovieListRow.js
        Page.js
        SimpleLink.js
        UserLogin.js
        UserSignup.js
    ▼ services
        HttpService.js
        MovieService.js
        UserService.js
    ▼ views
        MovieDetailView.js
        MovieFormView.js
        MovieListView.js
        UserLoginView.js
        UserSignupView.js
      App.js
      index.html
      index.js
    .gitignore
    LICENSE
    package.json
    package-lock.json
    README.md
    webpack.common.js
    webpack.dev.js
    webpack.prod.js
  External Libraries
```

NodeJS / NPM depencies are stored in the node_modules folder.

Components are located in the components sub-folder and start with capital letter and follow a camel-case style naming convention.

Generic and concrete services are stored in the services sub-folder.

Views are stored in the views sub-folder and follow a ComponentNameView naming convention.

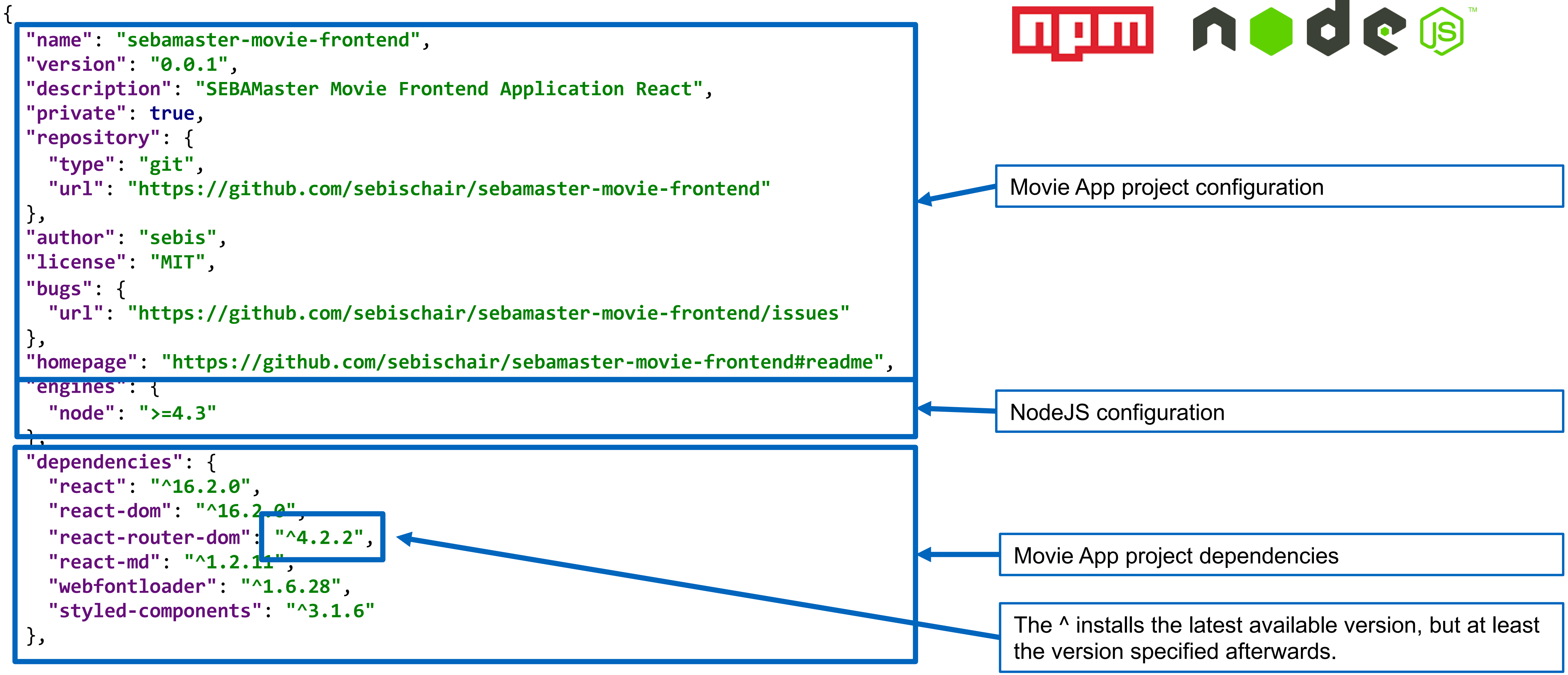
The App.js is the root source file for a React app. For the Movie App it contains mostly navigation configuration.

The application and dependencies are configured in the package.json file.

The webpack bundler configuration files.

Frontend NodeJS Configuration

package.json (1)



```
{
  "name": "sebamaster-movie-frontend",
  "version": "0.0.1",
  "description": "SEBAMaster Movie Frontend Application React",
  "private": true,
  "repository": {
    "type": "git",
    "url": "https://github.com/sebischair/sebamaster-movie-frontend"
  },
  "author": "sebis",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/sebischair/sebamaster-movie-frontend/issues"
  },
  "homepage": "https://github.com/sebischair/sebamaster-movie-frontend#readme",
  "engines": {
    "node": ">=4.3"
  },
  "dependencies": {
    "react": "^16.2.0",
    "react-dom": "^16.2.0",
    "react-router-dom": "^4.2.2",
    "react-md": "^1.2.11",
    "webfontloader": "^1.6.28",
    "styled-components": "^3.1.6"
  },
```

Movie App project configuration

NodeJS configuration

Movie App project dependencies

The ^ installs the latest available version, but at least the version specified afterwards.

https://nodejs.org
https://www.npmjs.com/

Frontend NodeJS Configuration

package.json (2)

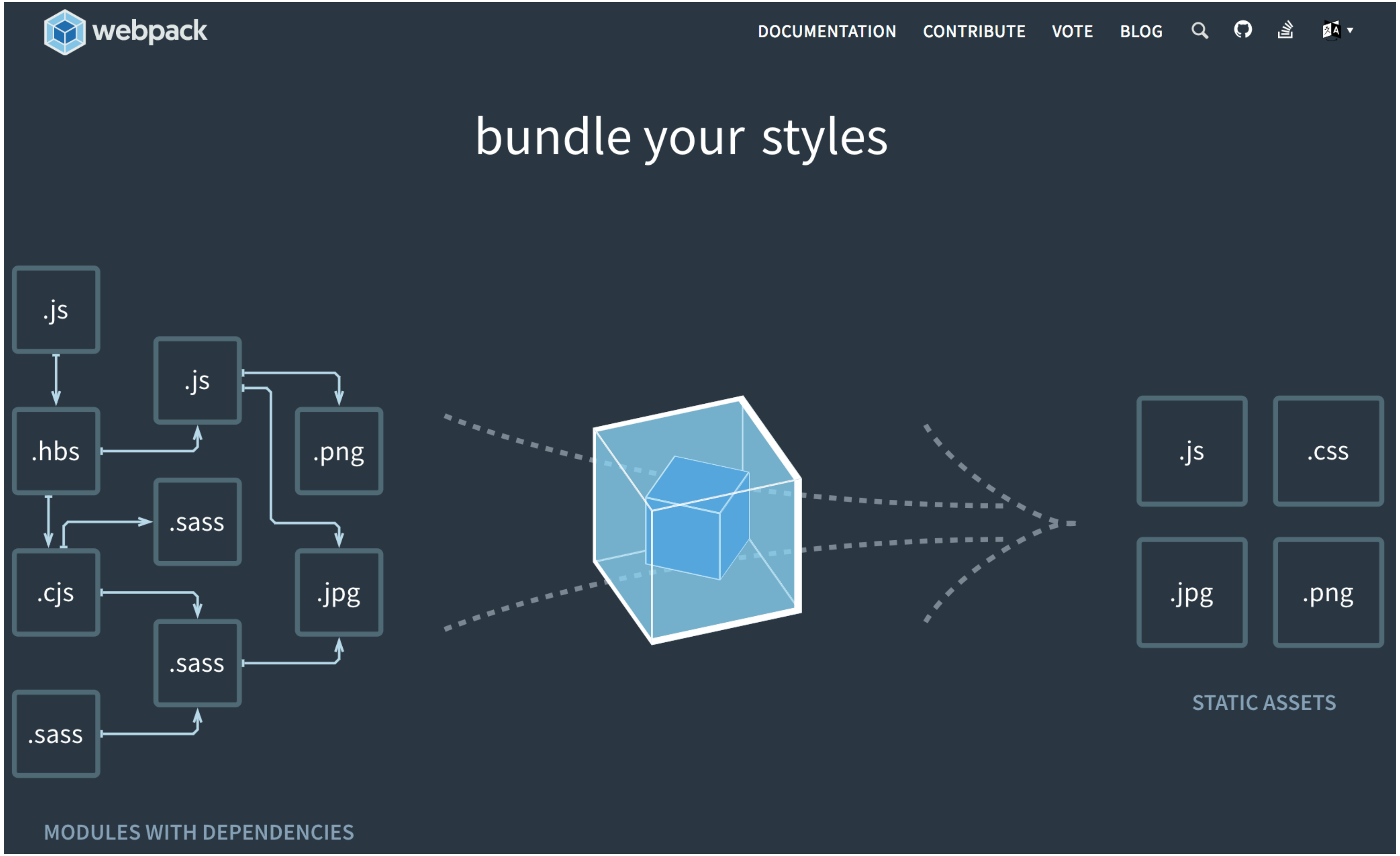
```
"devDependencies": {
   "babel-core": "6.25.0",
   "babel-loader": "7.1.1",
   "babel-preset-env": "1.6.0",
   "babel-preset-react": "6.24.1",
   "clean-webpack-plugin": "0.1.17",
   "css-loader": "0.28.7",
   "extract-text-webpack-plugin": "3.0.1",
   "html-loader": "0.4.5",
   "html-webpack-plugin": "2.29.0",
   "style-loader": "0.19.0",
   "webpack": "3.3.0",
   "webpack-dev-server": "2.9.1",
   "webpack-merge": "4.1.0"
 },
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "build": "webpack --config webpack.prod.js",
   "start": "webpack-dev-server --open 'Google Chrome' --config webpack.dev.js"
 }
}
```

Additional dependencies for development only, especially webpack and resource loading

Build management scripts, for example to start the application, run with npm run start

https://nodejs.org
https://www.npmjs.com/

Webpack Module Bundler



https://webpack.js.org/

Frontend Webpack Configuration (1)

webpack.common.js (1)

```
"use strict";

const webpack            = require('webpack');
const path               = require('path');
const ExtractTextPlugin  = require("extract-text-webpack-plugin");
const HtmlWebpackPlugin  = require('html-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');


module.exports = {
  entry: {
      'vendor': ['react','react-dom','react-router-dom'],
      'app': path.resolve(__dirname,'src/index.js')
  },
  output: {
      path: path.resolve(__dirname,'dist'),
      filename: 'scripts/[name].js'
  },
  module: {
      rules: [
          {
            test: /\.js$/,
            exclude: /(node_modules)/,
            use: {
                loader: 'babel-loader',
                options: {
                    presets: ['env', 'react']
                }
            }
          },
```

Load webpack modules

Specify required react modules and Movie App root source code directory

Specify the folder where all generated files shall be stored (dist folder) and name of root js file (app.js)

Configure webpack modules

Specify input to babel transpiler: Select all JavaScript files (*.js) except JS files from included vendor dependencies

Configure Babel: transpile ES6 and JSX for React in default mode ('env' setting of babel: https://babeljs.io/docs/usage/babelrc/#env-option)

https://webpack.js.org/

Frontend Webpack Configuration (2)

webpack.common.js (2)

```
        {
            test: /\.html$/,
            use: [ {
                loader: 'html-loader',
                options: {
                    minimize: true,
                    removeComments: false,
                    collapseWhitespace: false
                }
            }]
        },
        {
            test: /\.css$/,
            use: ExtractTextPlugin.extract({
                fallback: "style-loader",
                use: "css-loader"
            })
        }

    ]
},
plugins: [
    new CleanWebpackPlugin(['dist']),
    new webpack.optimize.CommonsChunkPlugin({name: "vendor", minChunks: Infinity,}),
    new HtmlWebpackPlugin({
        template: './src/index.html',
        filename: 'index.html',
        inject: 'body'
    }),
    new ExtractTextPlugin("styles/app.css")
]

};
```

Configure HTML Loader: Apply to all HTML files minimization.

Configure CSS Loader: inject all css files to HTML file

Binding of CSS and JS in HTML file

https://webpack.js.org/

Frontend Webpack Configuration (3)

webpack.**prod**.js

```
"use strict";

const webpack        = require('webpack');
const merge          = require('webpack-merge');
const UglifyJSPlugin = require('uglifyjs-webpack-plugin');

const common         = require('./webpack.common.js');

module.exports = merge(common, {
    plugins: [
        new UglifyJSPlugin({
            output: {
                comments: false,
            },
        }),
        new webpack.DefinePlugin({
            'process.env': {
                'NODE_ENV': JSON.stringify('production')
            }
        })
    ]
});
```

Load webpack modules and common configuration

Configure resources uglification module

https://webpack.js.org/

Frontend Webpack Configuration (4)

webpack.**dev**.js

```
"use strict";

const path   = require('path');
const merge  = require('webpack-merge');

const common = require('./webpack.common.js');

module.exports = merge(common, {
    devtool: 'inline-source-map',
    devServer: {
        contentBase: path.resolve(__dirname,'dist'),
        compress: true,
        port: 8000
    }
});
```

Load webpack modules and common configuration

Configure webpack dev server
An advantage of the webpack development server is that the application is automatically restarted when file changes are saved.

https://webpack.js.org/

React Components (Repetition)

What is a React Component?

- React components are used to split the UI into independent, re-usable pieces.
- Components can be nested, i.e. they are organized in a hierarchy
- Components are the core concept of a component-based web application architecture
- Has a well-defined input and output
- See the official documentation: https://reactjs.org/docs/react-component.html

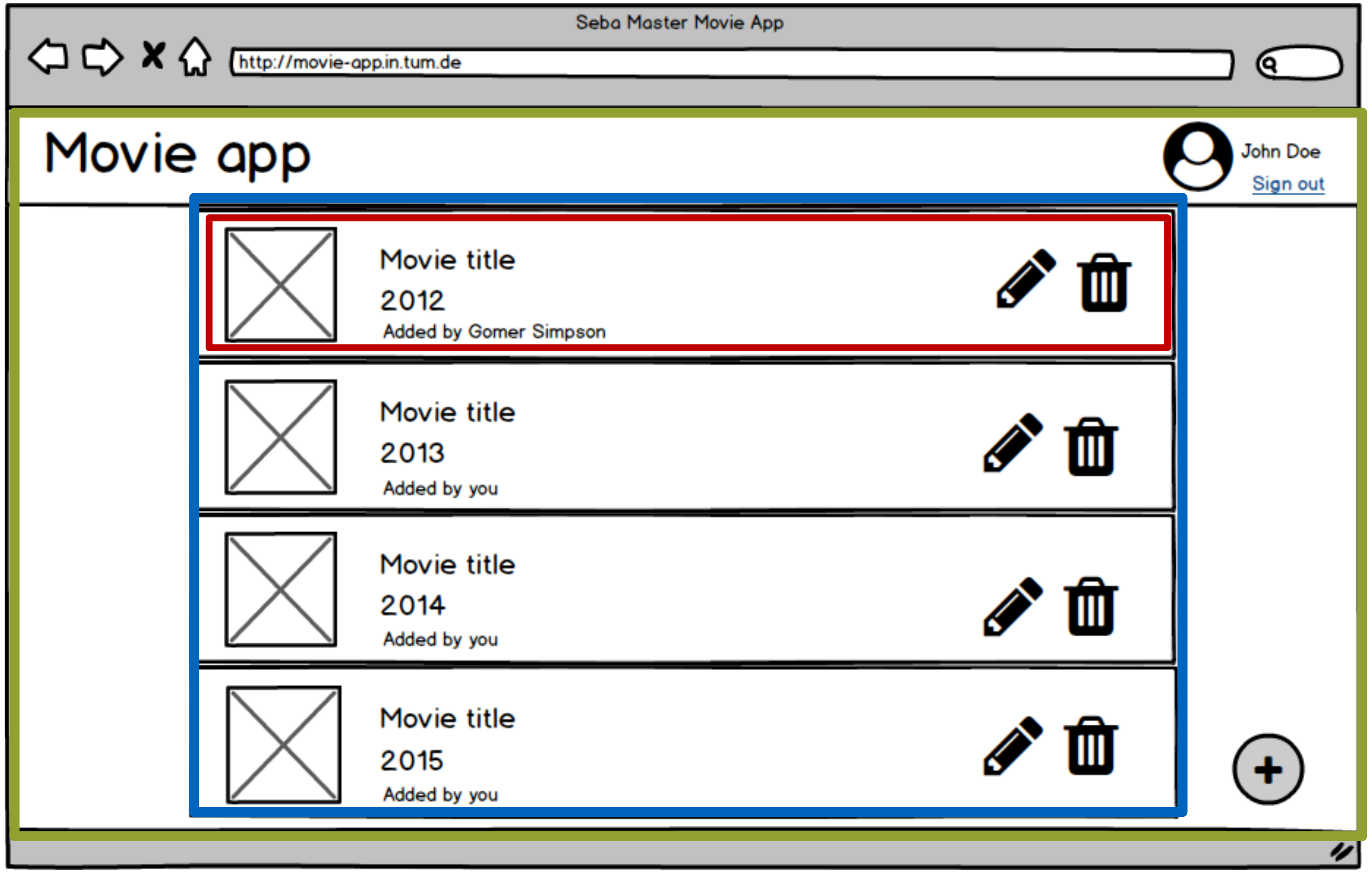
React Components in the Movie App (Repetition)

The root of a React app is the **App** component. It typically, resides in the App.js file in the source code root folder.

The **MovieList** component contains a sequence of movie list items.

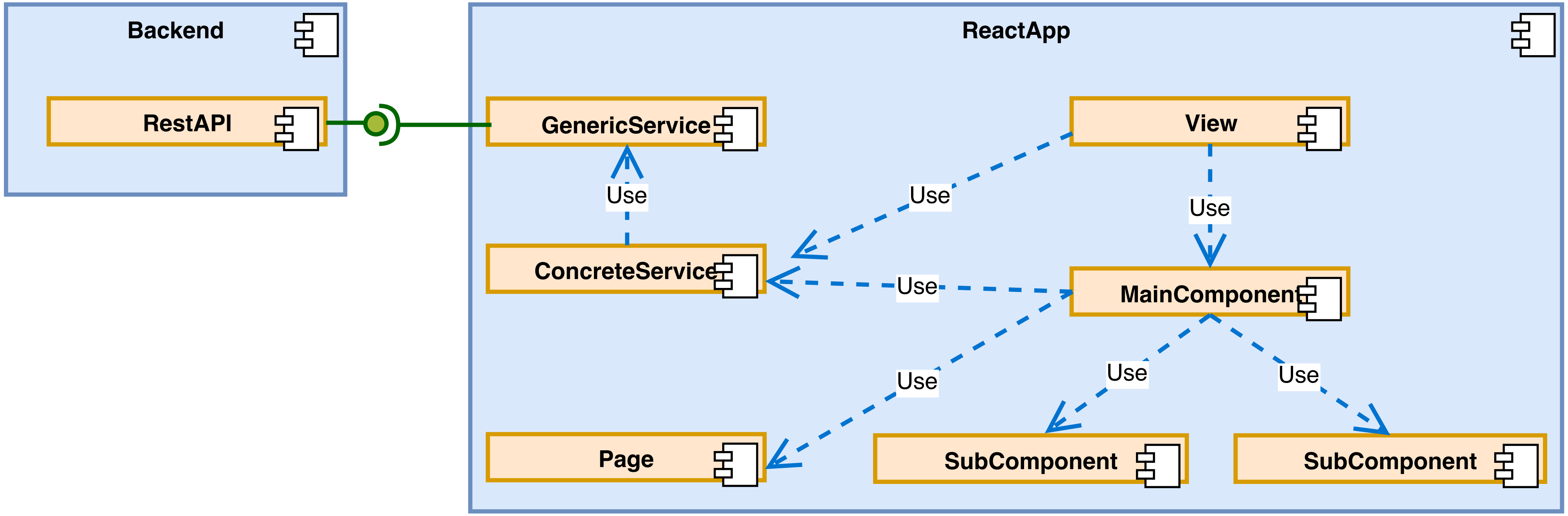
A MovieListItem is again a component.

Each movie is represented by an image, a title and a year.



https://reactjs.org/docs/thinking-in-react.html

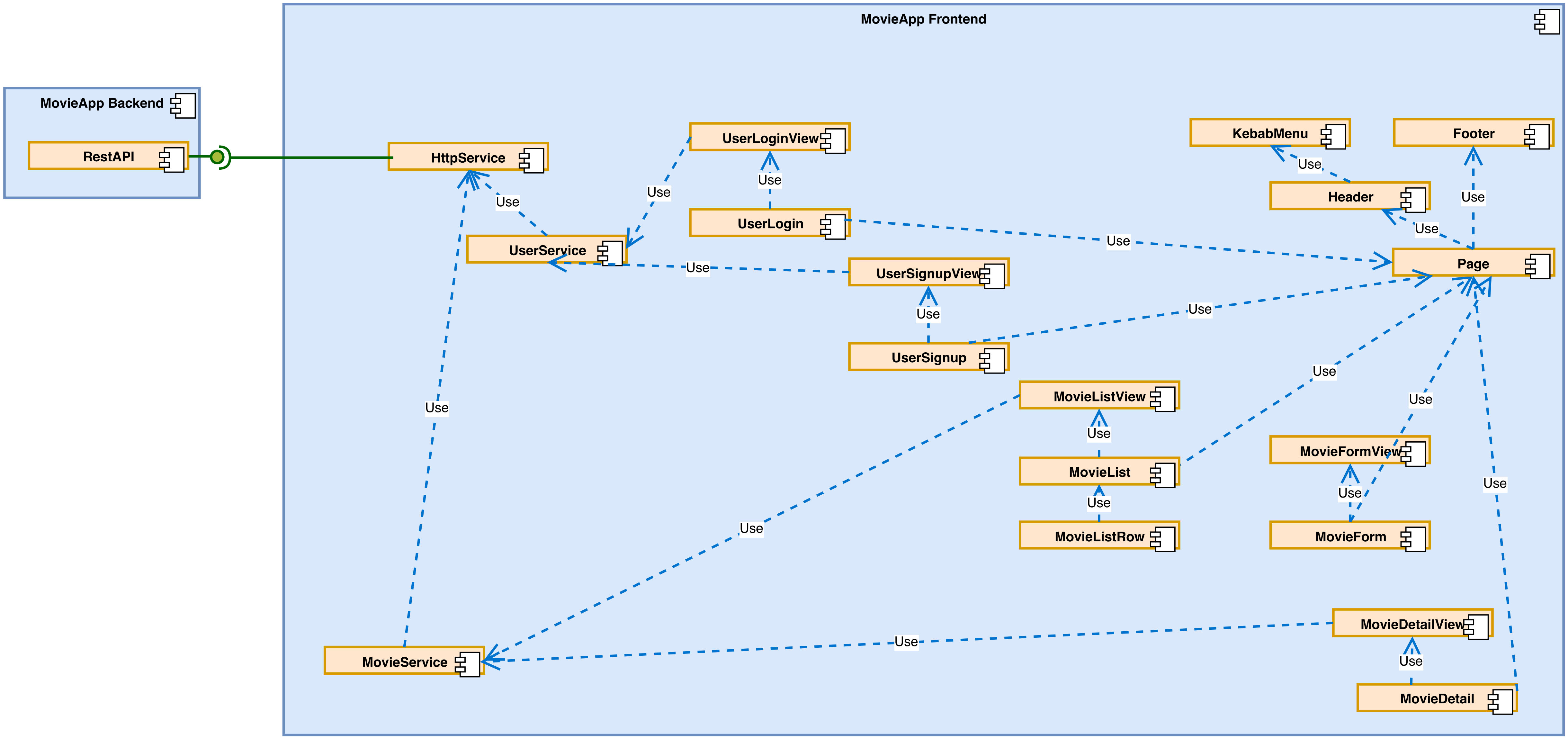
Recommended React Frontend Reference Architecture



We recommend to structure a SPA React app according to our reference architecture:

- A generic service encapsulates functionality to communicate in generic ways, e.g. via the HTTP protocol.
- Concrete APIs are consumed with a ConcreteService, e.g. the MovieService.
- Views represent different SPA-pages of a SPA and usually load the information to be displayed in that SPA-page.
- A Page component encapsulates application-wide content like headers and footers.

Movie App Frontend Architecture (1)



Movie App Frontend Architecture (2)

General comments:

- A React component name should start with a capital letter and defined in a single file named ComponentName.js
- Two major guidelines for the structuring of our frontend reference architecture are to separate code that frequently changes from code that is more stable and to assign components clear responsibilities.

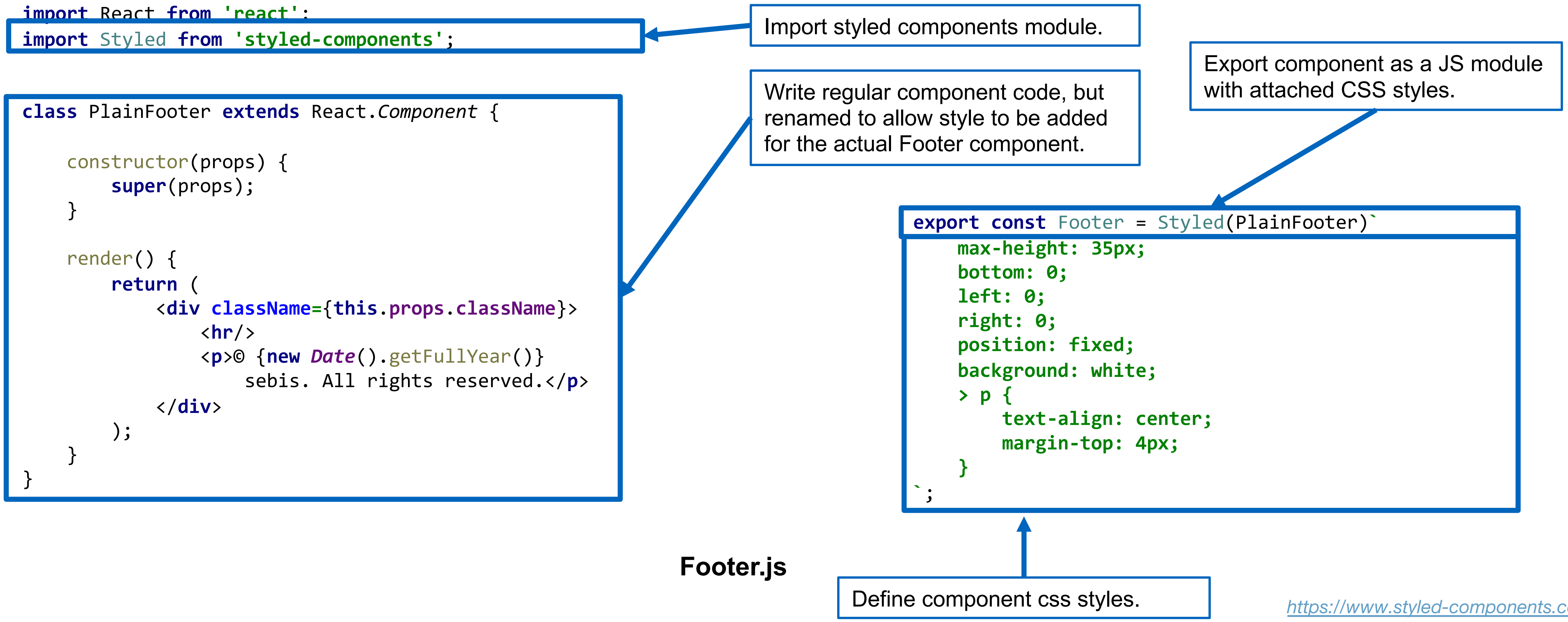
Concrete derivation of the Movie App frontend architecture to our frontend reference architecture:

- The HTTPService is an implementation of a GenericService.
- UserService and MovieService uses the generic HTTPservice to consume REST APIs.
- Each page of the Movie App single-page application is encapsulated in a View component like UserSignUpView or MovieListView.
- The Views contain one major component called UserSignUp or MovieList that loads a Page component that contains application-wide site elements like header, footer and navigation.
- We recommend that a View component also loads the data that should be displayed on the page. However, in certain cases, sub-components can use Services, too.

Styled Components

Styled components is a react community package.

The core of the concept of styled components is to encapsulate CSS styles for a component together with all other (JS) source code of a component. This can reduce possible error sources as often happens with global CSS styles. With styled components CSS classes are automatically prefixed with a component identifier.



```
import React from 'react';
import Styled from 'styled-components';
```

Import styled components module.

```
class PlainFooter extends React.Component {

    constructor(props) {
        super(props);
    }

    render() {
        return (
            <div className={this.props.className}>
                <hr/>
                <p>© {new Date().getFullYear()}
                    sebis. All rights reserved.</p>
            </div>
        );
    }
}
```

Write regular component code, but renamed to allow style to be added for the actual Footer component.

Export component as a JS module with attached CSS styles.

```
export const Footer = Styled(PlainFooter)`
    max-height: 35px;
    bottom: 0;
    left: 0;
    right: 0;
    position: fixed;
    background: white;
    > p {
        text-align: center;
        margin-top: 4px;
    }
`;
```

Footer.js

Define component css styles.

https://www.styled-components.com/