

React Domain Specific Language



Simplify something simple

15. 12. 2016

Written by

Simon Meyer
simon.meyer@stud.hs-hannover.de

Haminou Mohammed
haminou.mohammed@stud.hs-hannover.de

Sascha Becker
s.becker@wertarbyte.com

Contents

1	Introduction	1
1.1	Structure	1
1.2	UI	2
2	Domain Specific Model	3
3	Grammar	4
4	Generator	6

1 Introduction

Javascript is growing rapid fast in mobile web development, server architectures, rest services, desktop cross platform apps and microservices with the help of nodejs. On top of that every developer can choose which frontend platform is the best choice. React is developed from facebook and provides dynamic DOM manipulation without the need of reloading the complete DOM on every change.

Beside that react has a huge community bas for additional custom components and active issue tracking. Definitely something you should keep an eye on.

1.1 Structure

Every react component follows the same structure. Define your needed imports of react base code and additional components you want your component to use.

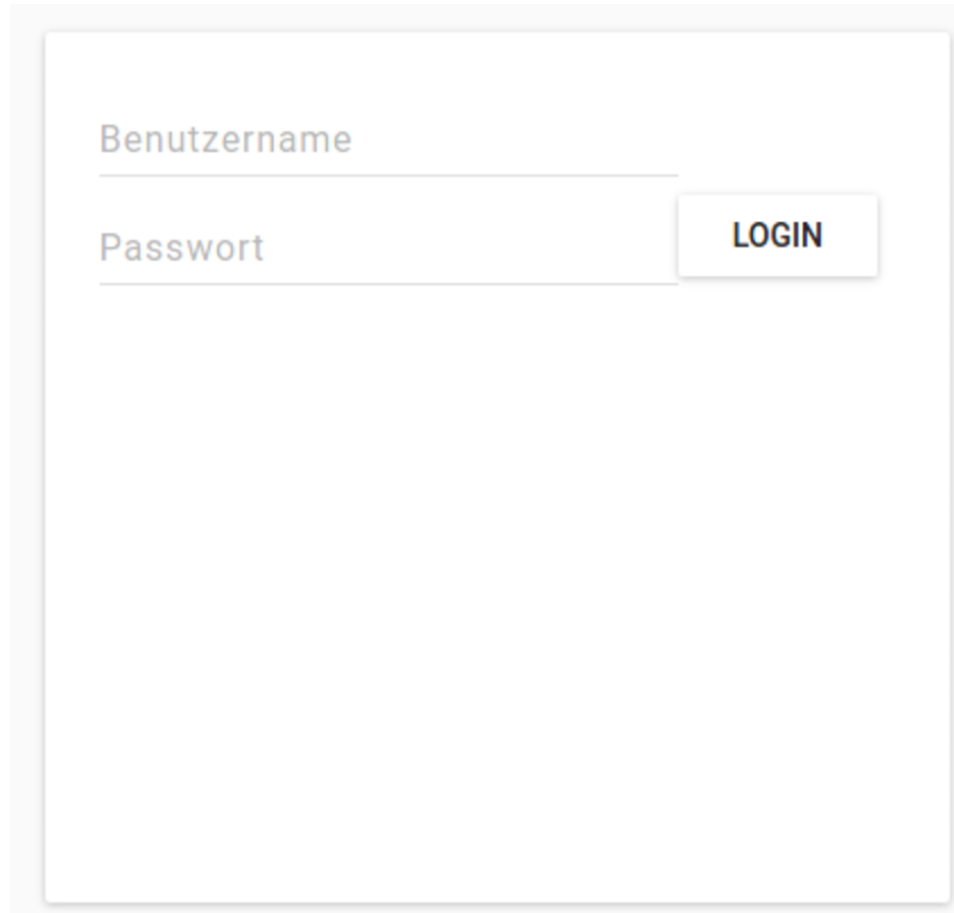
Define the default export name and what exactly should be exported. Normally it is your component for sure. A simple code example could look like this:

```
import React, { Component, PropTypes } from 'react'
import { Paper, RaisedButton, TextField } from 'material-ui'

export default class TestName extends Component {
  render() {
    return (
      <Paper style={{ width: 300 }}>
        <TextField
          hintText="Benutzername"
          style={{ float: 'left' }}
        />
        <TextField
          hintText="Passwort"
          style={{ float: 'middle' }}
        />
        <RaisedButton
          label="Login"
          style={{ float: 'right' }}
        />
      </Paper>
    )
  }
}
```

1.2 UI

The rendered code above will look like this:



A login form with two input fields and a button. The first input field is labeled "Benutzername" and the second is labeled "Passwort". A button labeled "LOGIN" is positioned to the right of the "Passwort" field. The form is enclosed in a light gray border.

Benutzername	
Passwort	LOGIN

2 Domain Specific Model

Our model focuses on generating one react component class object. As an example we take a simple paper element which holds two text fields and a button. The needed code looks like this.

```
Class TestName

Component Paper

TextField t1
.hintText="Benutzername"
.style=" float: 'left '

TextField t2
.hintText="Passwort"
.style=" float: 'middle '

Button b1
.label="Login"
.style=" float: 'right '
```

First we'll define our react component class name 'TestName'. This identifies our object in a global webapp and provides an import name for incode use. Then we define the root component of this react component. A react component can only return exactly one component. In this case we chose a paper element with 'Component Paper'.

After that we define various other components which get embedded in our root element, the paper component. In this case our textfields get some additional attributes for hinttext and css styling via the '.style' snippet.

3 Grammar

Our DSM starts with the Model, which contains many Classes. These Classes have a name and null to many materials. The Material can be a Component, a Button or a TextField. If the Material is a Component, it is embedding other Materials. To differentiate between the Attributes of the Materials there is a Property for each Material (ButtonProperty, TextFieldProperty).

```
Model:
root+=Class*
;

Class:
'Class' name=ID properties+=Material*
;

Material:
Component | Button | TextField
;

Component:
'Component' name=ID
;

Button:
'Button' name=ID properties+=ButtonProperty*
;

ButtonProperty:
Label | ButtonStyle
;

Label:
'.label=' attribute=STRING
;

ButtonStyle:
'.style=' buttonStyle=STRING
;

TextFieldStyle:
'.style=' textfieldStyle=STRING
```

```
;

TextField:
'TextField' name=ID properties+=TextFieldProperty*
;

TextFieldProperty:
HintText | TextFieldStyle
;

HintText:
'.hintText=' attribute=STRING
;
```


4 Generator

In the main template the doGenerate Method is creating a new class with ".java" ending for each 'Class' in the Model and will therefore execute the generateComponent Method. In the generateComponent() we iterate through all Materials and creating the Components. The determineElement() in the Component Generation specify the Material and creates the Material with its Attributes.

```
/*
 * generated by Xtext 2.10.0
 */
package org.xtext.example.mydsl.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.AbstractGenerator
import org.eclipse.xtext.generator.IFileSystemAccess2
import org.eclipse.xtext.generator.IGeneratorContext
import org.xtext.example.mydsl.myDsl.Component
import org.xtext.example.mydsl.myDsl.Button
import org.xtext.example.mydsl.myDsl.TextField
import org.xtext.example.mydsl.myDsl.Label
import org.xtext.example.mydsl.myDsl.ButtonStyle
import org.xtext.example.mydsl.myDsl.TextFieldStyle
import org.xtext.example.mydsl.myDsl.HintText
import org.xtext.example.mydsl.myDsl.Material
//import org.xtext.example.mydsl.myDsl.Paper
//import org.xtext.example.mydsl.myDsl.PaperStyle
import org.xtext.example.mydsl.myDsl.Class

/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#code-generation
 */
class MyDslGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa
        , IGeneratorContext context) {
        for(e: resource.allContents.toIterable().filter(Class)){
            fsa.generateFile('' e.name .js'',

```

```

        generateComponent(e,resource));
    }
}

def generateComponent(Class t, Resource res) '''
    doImports()

    FOR s: t.properties
    IF s instanceof Component
    FOR q: res.allContents.toIterable.filter(Class)
    export default class t.name extends Component{
        render() {
            return(
                < s.name style={{ }}
                    FOR m: res.allContents.toIterable.filter(Material)
                    determineElement(m,res) ENDFOR
                </ s.name >
            )
        }
    }
    ENDFOR
    ENDIF
    ENDFOR
'''

def doImports()'''
    import React, { Component, PropTypes } from 'react'
    import { Paper, RaisedButton, TextField } from 'material-ui'
'''

def determineElement(Material e, Resource res)'''
    IF e instanceof Paper<PaperFOR f: e.properties IF f
    instanceof Label label=" f.attribute" ENDFIF IF f instanceof
    PaperStyle style={{ " f.paperStyle" }} ENDFIF ENDFOR '''/> ENDFIF
    IF e instanceof Button<RaisedButtonFOR f: e.properties IF f
    instanceof Label label=" f.attribute" ENDFIF IF f instanceof
    ButtonStyle style={{ " f.buttonStyle" }} ENDFIF ENDFOR /> ENDFIF
    IF e instanceof TextField<TextFieldFOR f: e.properties IF f
    instanceof HintText hintText=" f.attribute" ENDFIF IF f
    instanceof TextFieldStyle style={{ " f.textfieldStyle" }}
    ENDFIF ENDFOR /> ENDFIF
'''
}

```