2018

VLC PHOTONICS

Documentation

CONTENTS

1- DESIGN PATTERN

Overview

The web application follows an MVC (Model-view-controller) object-oriented architecture bridging scopes between client and server-side: the model defines data logic (model schemas, constraints, paths), the view interprets the outcome of the UI scripts while the controller provides factories projecting the model into the view.

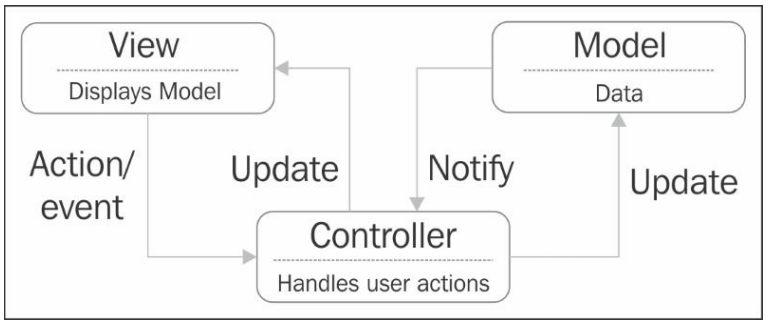

Figure 1 MVC architecture. source: https://www.oreilly.com/library/view/spring-mvc-designing/9781787126398/graphics/B03070_02_01.jpg

Code distribution

- **Server-side**
  - **.\app.js**

    The gateway of the web application, partaking in the runtime environment by means of dependencies deployment. Mainly establishing the API by allocating data variables defined in the model and setting methods upon which the controller can operate the model.

  - **.\models\main.js**

    Introduces data structures in model schemas containing paths for attributes, constraints and data types that meet requirements set by eventual manipulations.

- **Client-side**
  - **.\client\app.js**

    The primary node of the UI that provides routing by binding routing services to view templates and their respective controller.

  - **.\client\index.html**

    The frontend gateway of the UI: The main -constant- view, managing UI scripts and pointing out to cascading views.

  - **.\client\controllers\mains.js**

    Where factories operate services out of pre-established methods. CRUD (Create, Read, Update, Delete) services are defined on different levels on with **.\app.js**, **.\models\main.js** and **.\client\app.js** and deployed from here.

- **.\client\controllers\views*.html**

  These are adjacent views cascading out of **.\client\index.html** where every collection/document's display is formatted

Development technological set

- **MEAN stack**
  MongoDB as data store, Express as a web framework, AngularJS as a frontend focused framework and Nodejs as a runtime environment.

- **Dependencies**
  - Body-parser: An Express middleware module, here used for object/JSON conversion involved in handling http requests.
  - Mongoose: Interface for modeling MongoDB based data.
  - jQuery: A JavaScript library used by the API especially for CRUD operations handling events.

2- IMPLEMENTATION

- .\app.js

```
1.  //create an Express application
2.  const express = require('express');
3.  const app = express();
4.
5.  //import middleware modules
6.  const bodyParser = require('body-parser');
7.
8.  //import mongoose module
9.  const mongoose = require('mongoose');
10.
11. //point out client-side to middleware
12. app.use(express.static(__dirname+'/client'));
13.
14. //import json middleware parser
15. app.use(bodyParser.json());
16.
17. //creating a module for documents in each collection
18. Foundry =require('./models/main');
19. Block =require('./models/main');
20. Definition =require('./models/main');
21. Component =require('./models/main');
22.
23. // Connect to Mongoose
24. mongoose.connect('mongodb://localhost:27017/photonicsdb', { useNewUrlParser: true });
25. var db = mongoose.connection;
26.
27. // Routing
28.
29. // Setting a GET method to the index
30. app.get('/', (req, res) => {
31.     res.send('Photonicsdb: please use /api/...');
32. });
33.
34. // GET method route for a collection of foundry documents
35. app.get('/api/foundries', (req, res) => {
36.     Foundry.getFoundries((err, foundries) => {
37.         if(err){
38.             throw err;
39.         }
40.         res.json(foundries);
41.     });
42. });
43.
44. // GET method route for a foundry document
45. app.get('/api/foundries/:_id', (req, res) => {
46.     Foundry.getFoundryById(req.params._id, (err, foundry) => {
47.         if(err){
48.             throw err;
49.         }
50.         res.json(foundry);
51.     });
52. });
53.
54. // POST method route for a foundry document
55. app.post('/api/foundries', (req, res) => {
56.     var foundry = req.body;
57.     Foundry.addFoundry(foundry, (err, foundry) => {
58.         if(err){
59.             throw err;
60.         }
61.         res.json(foundry);
```

```
    });
});

// PUT method route for a foundry document
app.put('/api/foundries/:_id', (req, res) => {
    var id = req.params._id;
    var foundry = req.body;
    Foundry.updateFoundry(id, foundry, {}, (err, foundry) => {
        if(err){
            throw err;
        }
        res.json(foundry);
    });
});

// DELETE method route for a  foundry document
app.delete('/api/foundries/:_id', (req, res) => {
    var id = req.params._id;
    Foundry.removeFoundry(id, (err, foundry) => {
        if(err){
            throw err;
        }
        res.json(foundry);
    });
});

//setting 3000 as the port the app uses to listen to requests
app.listen(3000);
console.log('Running on port 3000...');
```

- **.\models\main.js**

```
//import mongoose module
const mongoose = require('mongoose');
//defining ObjectId as a data type
var ObjectId = mongoose.Schema.Types.ObjectId;
//components collection schema
const componentSchema = mongoose.Schema({
    name: String, type: {
    type: String,
    enum: ['parameter', 'function','status']}
});
//definitions collection schema
const definitionSchema = mongoose.Schema({
    name: String,
    function: String,
    specifications: [String]
});
//blocks collection schema
const blockSchema = mongoose.Schema({
    name: String,
    status:{
        type: String,
        enum: ['active', 'obsolete']},
    reference: {
        type: String,
        enum: ['PDK', 'paper', 'VLC']},
    foundry: String,
    class: String,
    parameters: [String],
    valuec: [String],
    valueo: [String]
});
//foundries collection schema
```

```
const foundrySchema = mongoose.Schema({
    name: String,
    contact:String,
    phone: String,
    mail: String,
    url:String
});

// creating module for schema by passing document as argument
const Definition = module.exports = mongoose.model('Definition', definitionSchema);
const Component = module.exports = mongoose.model('Component', componentSchema);


//create a module assigining find function to GET method by passing callback argument
module.exports.getFoundries = (callback, limit) => {
    Foundry.find(callback).limit(limit);
}

//create a module assigining findById function to GET method by passing id and callback arguments
module.exports.getFoundryById = (id, callback) => {
    Foundry.findById(id, callback);
}

//create a module assigining create function to ADD method
module.exports.addFoundry = (foundry, callback) => {
    Foundry.create(foundry, callback);
}

//create a module assigning findOneAndUpdate function as an object and passing (document attributes: attributes paths) arguments
module.exports.updateFoundry = (id, foundry, options, callback) => {
    var query = {_id: id};
    var update = {
        name: foundry.name,
        contact: foundry.contact,
        phone: foundry.phone,
        mail: foundry.mail,
        url: foundry.url
    }
    Foundry.findOneAndUpdate(query, update, options, callback);
}

//create a module assigining remove function by passing id argument
module.exports.removeFoundry = (id, callback) => {
    var query = {_id: id};
    Foundry.remove(query, callback);
}
```

- **Client-side**

  - **.\client\app.js**

```
// importing ngRoute directive to provide routing services
var myApp = angular.module('myApp',['ngRoute']);
//configuring routes by specifying bound controller and view
myApp.config(function($routeProvider){
    $routeProvider.when('/', {
        controller:'MainsController',
        templateUrl: 'views/mains.html'
    })
.otherwise({
        redirectTo: '/'
    });
});
```

- **.\client\index.html**

```
<!doctype html>
  <!-- Bootstrap core CSS -->
  <link href="views/css/bootstrap.min.css" rel="stylesheet">
  <!-- Material Design Bootstrap -->
  <link href="views/css/mdb.min.css" rel="stylesheet">
</head>
<body style="background-color:#E4E4E4;">
  <!-- Navbar and navbar links -->
  <nav class="navbar navbar-expand-lg bg-dark text-warning">
    <a class="navbar-brand bg-dark " href="#/mains" style="width: 320px;">
      <img src="views/logo/logo.svg" alt="">
    </a>
    <!-- Toggler for smaller screen sizes -->
    <button class="navbar-toggler text-light" type="button" data-toggle="collapse" data-target="#navbarPh" aria-controls="navbarPh" aria-expanded="false" aria-label="Toggle navigation">
      <span data-feather="more-vertical" style="width: 23px; height: 23px;">
      </span>
    </button>
…
    <!-- container for collapsing views by means of ngView that complements the ngRoute directive  -->
    <div class="container">
      <div class="row">
        <div class="col-md-12 text-monospace">
          <div ng-view></div>
        </div>
      </div>
    </div>
  </div>
  <!-- Bootstrap core JavaScript-->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script>window.jQuery || document.write('<script src="views/scripts/jquery-slim.min.js"><\/script>')</script>
<script src="views/scripts/popper.min.js"></script>
<script src="views/scripts/bootstrap.min.js"></script>
<!-- Angular core JavaScript -->
<script src="views/scripts/angular.js"></script>
<script src="views/scripts/angular-route.js"></script>
<!-- MDB core JavaScript -->
<script type="text/javascript" src="views/scripts/js/mdb.min.js"></script>
<!-- Routing provider and controllers -->
<script src="app.js"></script>
<script src="controllers/mains.js"></script>
<!-- ClipboardJS -->
<script src="views/scripts/clipboard.min.js"></script>
<script>
var clipboard = new ClipboardJS('.btn');
clipboard.on('success', function(e) {
    console.log(e);
});
clipboard.on('error', function(e) {
    console.log(e);
});
</script>
<!-- Icons -->
<script src="https://unpkg.com/feather-icons/dist/feather.min.js"></script>
<script>
  feather.replace()
</script>
```

- **.\client\controllers\mains.js**

```
var myApp = angular.module('myApp');
    //Load services required by the core module
myApp.controller('MainsController', ['$scope', '$http', '$location', '$routeParams', function($scope, $http, $location, $routeParams){
    console.log('MainsController loaded...');
    //Reference to application model (with scope) to create CRUD operations factory using http service to generate HTTP requests
    //Read all foundries
    $scope.getFoundries = function(){
        $http.get('/api/foundries').success(function(response){
            $scope.foundries = response;
        });
    }
    //Read foundry by id
    $scope.getFoundry = function(){
        var id = $routeParams.id;
        $http.get('/api/foundries/'+id).success(function(response){
            $scope.foundry = response;
        });
    }
    //Create a foundry
    $scope.addFoundry = function(){
        console.log($scope.foundry);
        $http.post('/api/foundries/', $scope.foundry).success(function(response){
            window.location.href='#/foundries';
        });
    }
    //Read a foundry by id and update it
    $scope.updateFoundry = function(){
        var id = $routeParams.id;
        $http.put('/api/foundries/'+id, $scope.foundry).success(function(response){
            window.location.href='#/foundry/'+id;
        });
    }
    //Read a foundry by id and delete it
    $scope.removeFoundry = function(id){
        $http.delete('/api/foundries/'+id).success(function(response){
            window.location.href='#/foundries';
        });
    }
}]);
```

- **.\client\views*.html**
    - **mains.html**

```
  <!-- Call reading all foundries service from the controller-->
<div ng-init="getFoundries()">
          <!-- Search within the read foundries documents-->
          <input type="text" class="form-control" id="inlineFormInputGroupAll" ng-model="q" placeholder="Keyword" aria-label="filter blocks" placeholder="Keyword">
…
<!-- Display the read documents by foundry scope | filter with reference to searched keywords-->
        <div ng-repeat="foundry in foundries | filter : q">
…
                <!-- Link to redirect to the foundry by id -->
                <a href="#/foundry/{{foundry._id}}" class="btn btn-sm btn-lg btn-block btn-outline-elegant btn-rounded waves-effect" role="button" aria-pressed="true">
                More info
                </a>
```

- **foundry.html**

```
1.          <!-- Call reading a foundry service from the controller-->
2.          <div ng-init="getFoundry()">
3.  …
4.                      <!-- Call reading all the blocks service from the controller-->
5.                      <div class="collapse show" id="collapseBlock" ng-init="getBlocks()">
6.                          <!-
    - Display the read blocks' documents by block scope | condition: match if the block foundry (from blo
    cks collection) attribute with the block name attribute (from foundries collection)-->
7.                          <div ng-repeat="block in blocks" ng-if="block.foundry==foundry.name">
8.                            <!-- Link to redirect to the block by id -->
9.                            <h5>
10.                     <a href="#/block/{{block._id}}" class="badge badge-dark">{{block.name}}
11.                     </a>
12. …
13.                     <!-- Link to redirect to foundry editing view by id -->
14.                     <a href="#/foundry/edit/{{foundry._id}}" class="btn btn-warning btn-sm">
15.                       Edit
16.                     </a>
17. …
18.                     <!-- Link to display confirmation window for foundry deleting -->
19.                     <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-
    target="#fModal" data-backdrop="false">
20.                       Delete
21.                     </button>
22. …
23.                             <!-- Call deleting a foundry service from the controller-->
24.                             <button type="button" class="btn btn-warning" ng-
    click="removeFoundry(foundry._id)">
25.                               Yes, delete !
26.                             </button>
```

- blocks.html, definitions.html and components.html follow the same pattern as mains.html
- block.html definition.html and component.html follow the same pattern as foundry.html

- **addB.html**

- Create a component:

```
1.                      <!-- Call create a component service upon submition -->
2.                      <form ng-submit="addComponent()">
3.  …
4.                      <!-- Condition : when the component
    name form is triggered, check box appears, if checked, display read components' documents | filter by
     the form's content-->
5.                              <input type="checkbox" class="custom-control-input" id="customCheck1" ng-
    model="checked">
6.                              <label class="custom-control-label" for="customCheck1" ng-
    if="component.name">check matching components
7.                              </label>
8.                              <br>
9.                              <span ng-if="checked">
10.                               <select class="form-control mdb-color white-text border-0">
11.                                 <option ng-
    repeat="component in components | filter : component.name">{{component.name}}
12.                                 </option>
13.                               </select>
14.                             </span>
```

- Define a class:

```
                        <!-
- Call read components service and display them in a selection form | filter by type attribute 'funct
ion' in the first form and 'parameter' in the second-->
                        <div ng-init="getComponents()">
                          <select class="form-control" ng-model="definition.function">
                            <option value="" disabled selected>
                              Choose a function
                            </option>
                            <option ng-
repeat="component in components | filter : {type: 'function'}" value="{{component.name}}">
                              {{component.name}}
                            </option>
                          </select>
                        </div>
…
                      <!-- Display the selected parameters in a table -->
                      <label class="justify-content-center align-items-center font-weight-bold">
                         Selected parameters
                      </label>
                      <table class="table table-sm">
                        <tbody>
                          <tr ng-repeat="specifications in definition.specifications">
                            <th scope="row">
                              {{$index+1}}
                            </th>
                            <td>
                              {{definition.specifications[$index]}}
                            </td>
                          </tr>
                        </tbody>
                      </table>
```

- The remaining of addB.html follows the same pattern as the mentioned
- addC.html and addF.html have the same content as addB.html with with the exception of collapsing different tabs ("DEPHINE A CLASS" is collapsed instead of "ADD A COMPONENT" in addC.html and "CREATE A BLOCK" in addB.html)

    - **edit_foundry.html**

```
<!-
- Call update a foundry service upon submition, Call reading a foundry service from the controller --
>
            <form ng-submit="updateFoundry()" ng-init="getFoundry()">
…
```

- Edit_block.html, edit_definition.html, edit_component.html and fill_block.html follow the same pattern as mains.html

Installation

1. Install MongoDB (https://www.mongodb.com/download-center/community)
   Install nodejs (https://nodejs.org/en/download/) and a bash command line (https://gitforwindows.org/)
2. Create database and log directories: create a new file "data" and then two files "data\db" and "data\log"
3. Create a data store and collections for Dephine in MongoDB:

- With Robo 3T
  i. Install ROBO 3T and run it
  ii. Create a database "photonicsdb"

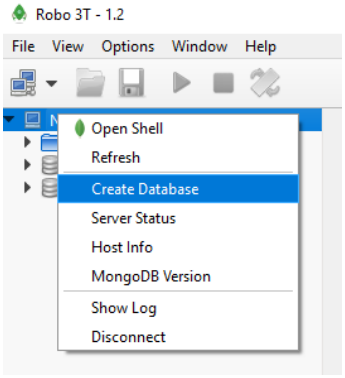

Figure 2 Create a database with ROBO 3T

iii. Create the collections "foundries", "blocks", "definitions", "components"

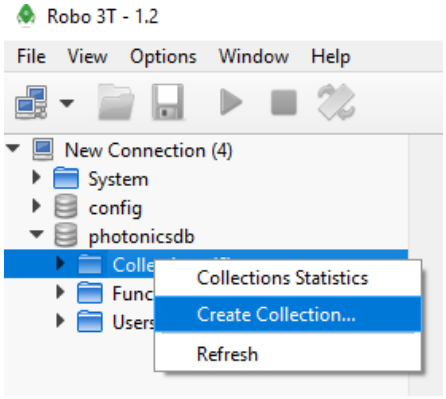

Figure 3 Create a collection with ROBO 3T

- With mongo shell
  i. Run mongod.exe with administrator priviliges (located by default in c:\program files\MongoDB\server\3.6\bin)

Create a Database:

If a database does not exist, MongoDB creates the database when you first store data for that database. As such, you can switch to a non-existent database and perform the following operation in the mongo shell:

ii. `use photonicsdb`

The insertOne() operation creates both the database myNewDB and the collection myNewCollection1 if they do not already exist.

Create a Collection:

If a collection does not exist, MongoDB creates the collection when you first store data for that collection.

iii. `db.foundries.insertOne( { x: 1 } )`

db.blocks.insertOne({ x: 1 })

db.definitions.insertOne({ x: 1 })

db.components.insertOne({ x: 1 })

4. Clone Dephine (copy/paste photonicsdb file with its content)
5. Git Bash in c:\photonicsdb
6. Run the command "node app"
7. Run Dephine on a browser on: http://localhost:3000

Notes:

- TypeError: $http.get(...). success is not a function: $http Success/Error methods (used in **.\client\controllers\mains.js**) were deprecated starting from AngularJS 1.6. Upgrading to any version following 1.5 will result in the mewntioned error.
- Supported browsers: Best viewed in Chrome (strongly recommended for mobile use), runs also on Firefox, IE10+, Edge, Opera and Safari
- Potential issues on iOS mobile browsers: Delete confirmation, add dropdown, zooming

3- UI/UX

➢ **The navbar**

The navbar has 4 links: foundries, blocks, library of classes, components and a collapsible add. These lead to views where foundries', blocks', definitions' and components collection documents are displayed. The navbar has a toggler for smaller screen sizes.

Figure 4 The navbar

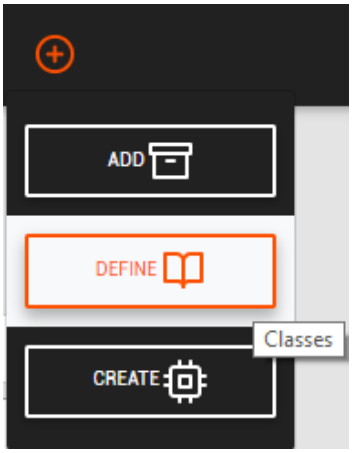

Figure 5 Collapsible nav link

Figure 6 navbar toggler

➢ **The views**

- **Home (Foundries)**
  Foundries' collection documents are returned on cards, every card has foundries' attributes and a "more info" button leading to the single **foundry view**. This view has a clipboard copy button and a search form.

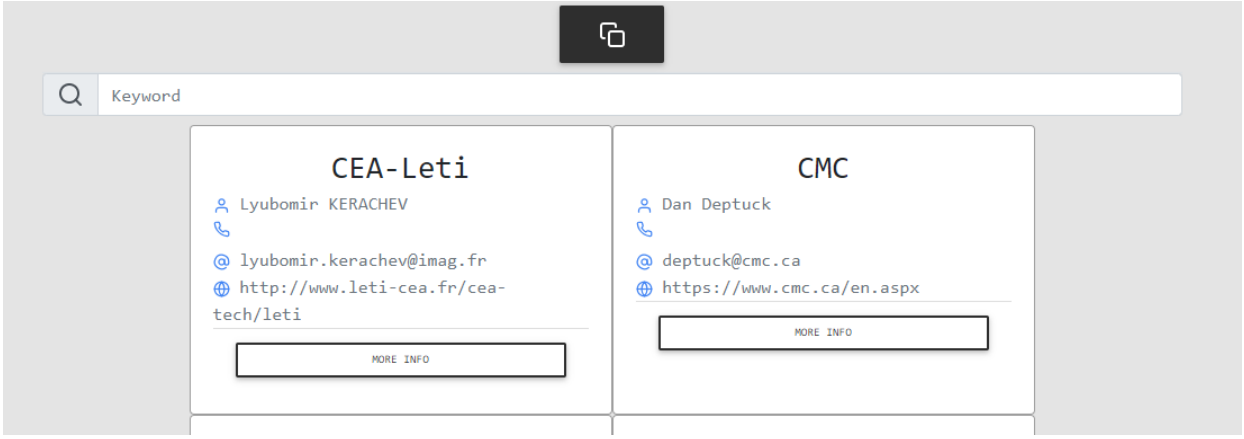

Figure 7 Foundries view

- **Foundry**
  Accessed foundry document is returned on a card with the foundry attributes, also, documents from blocks collection belonging having the accessed foundry name as an attribute will be displayed. This view has an Edit button leading to **edit_foundry view**, a delete button to a delete confirmation window and a clipboard copy button.

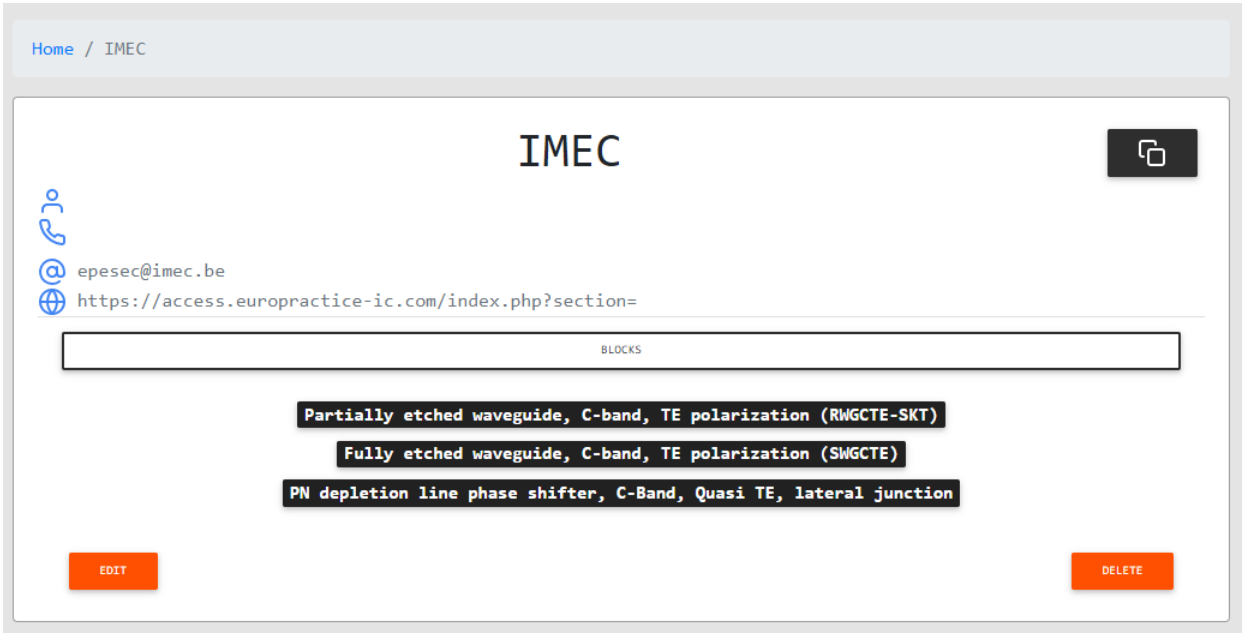

Figure 8 Foundry view

- **edit_foundry**
  Returns a form prefilled with the foundry attributes that can be updated through the submit button.

Figure 9 Edit foundry view

- **Blocks**
  Blocks' collection documents are returned on cards, every card has blocks' attributes and a "more info" button leading to the single **block view**. This view has a clipboard copy button, a search form, filtering blocks by foundry and by class forms, the filtering by function form is meant to filter the classes form. Under the forms, a count for the number of returned blocks, a show/hide classification attributes on the cards and a switch slider (ON: show only "active" blocks, OFF: show all).

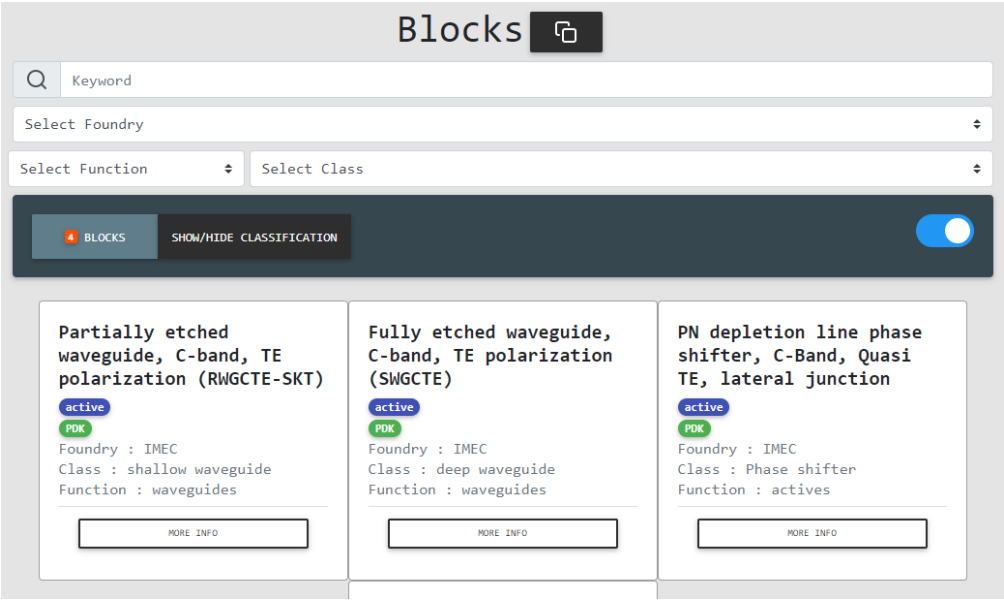

Figure 10 Blocks view

- **Block**
  Returns accessed block document on a card with the block attributes and the assigned class document from definitions collection. The view has a clipboard copy button, an Edit button leading to **edit_block view**, a Fill button to **fill_block view** and a delete button to a delete confirmation window. (n.b. "Class bound parameters" is an attribute of the class (definitions collection) while "Outstanding parameters" is an attribute of the block itself (see **.\models\main.js** for the complete model schemas).

Figure 11 Block view

- **edit_block**
  Returns a form prefilled with the block attributes that can be updated through the submit button. Hold <ctrl> for multiple selection of outstanding parameters, <shift> for sequential selection and click back to deselect. Selected parameters are shown under the submit button.

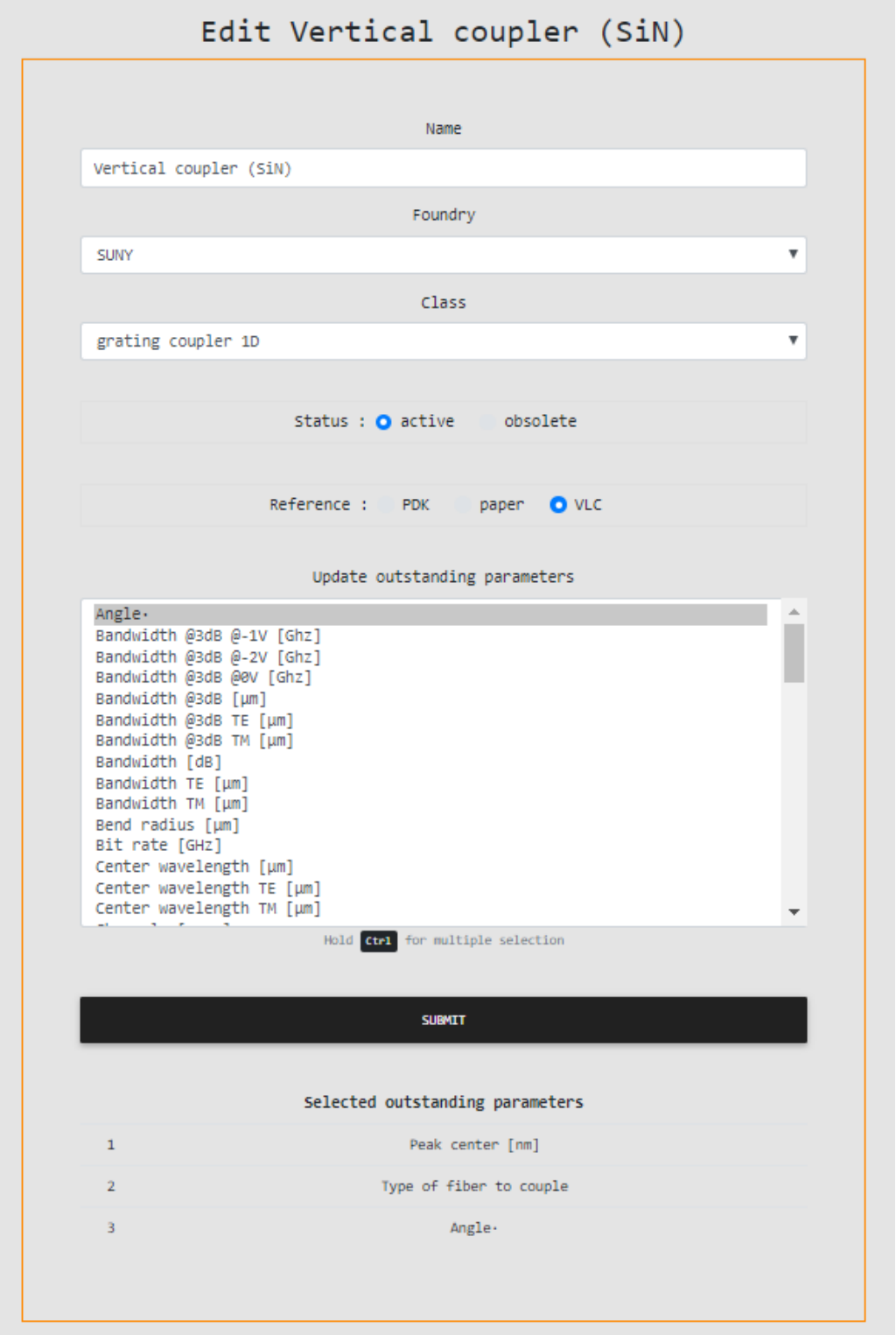

Figure 12 Edit block view

- **fill_block**
  Displays the block view with forms to update values of parameters by index.

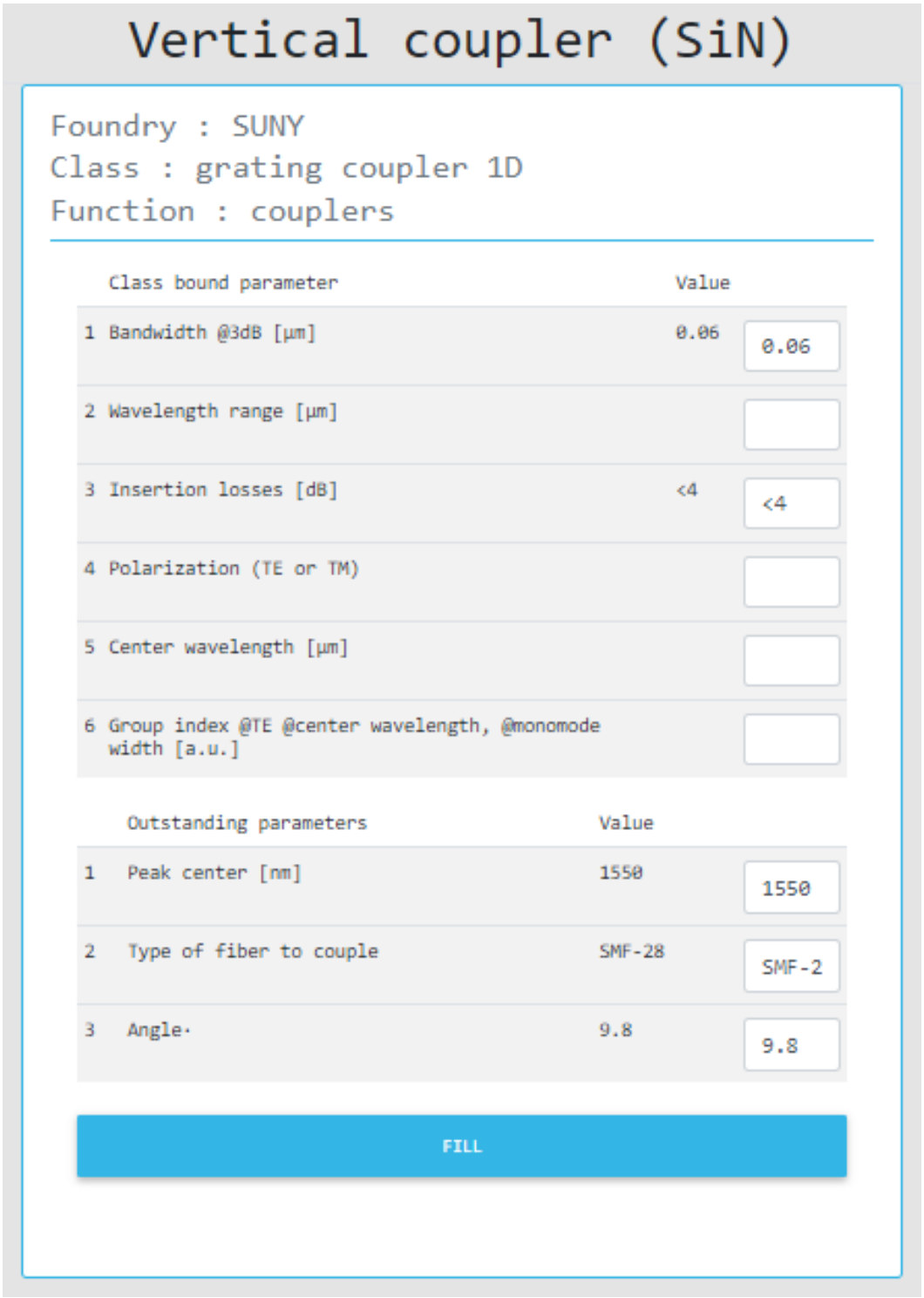

Figure 13 Fill block view

- **Library of classes** 🕮
  Returns definitions' collection documents on cards, every card has definitions' attributes and a "more info" button leading to the single **definition view.** This view has a search form and a filtering classes by function form.

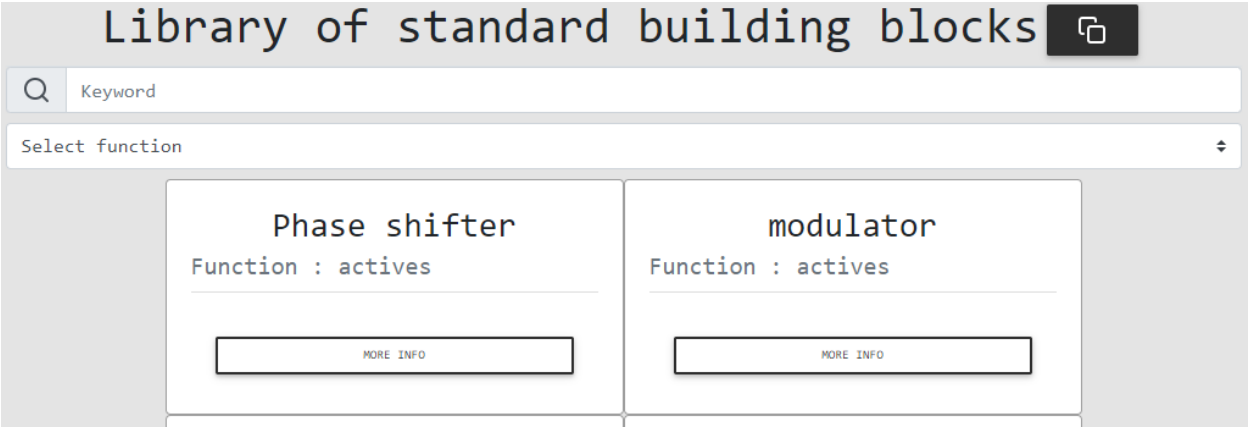

Figure 14 Library of classes view

- **Definition**
  Returns accessed class document displaying its attributes. This view has a clipboard copy button, an Edit button leading to **edit_definition view** and a delete button to a delete confirmation window

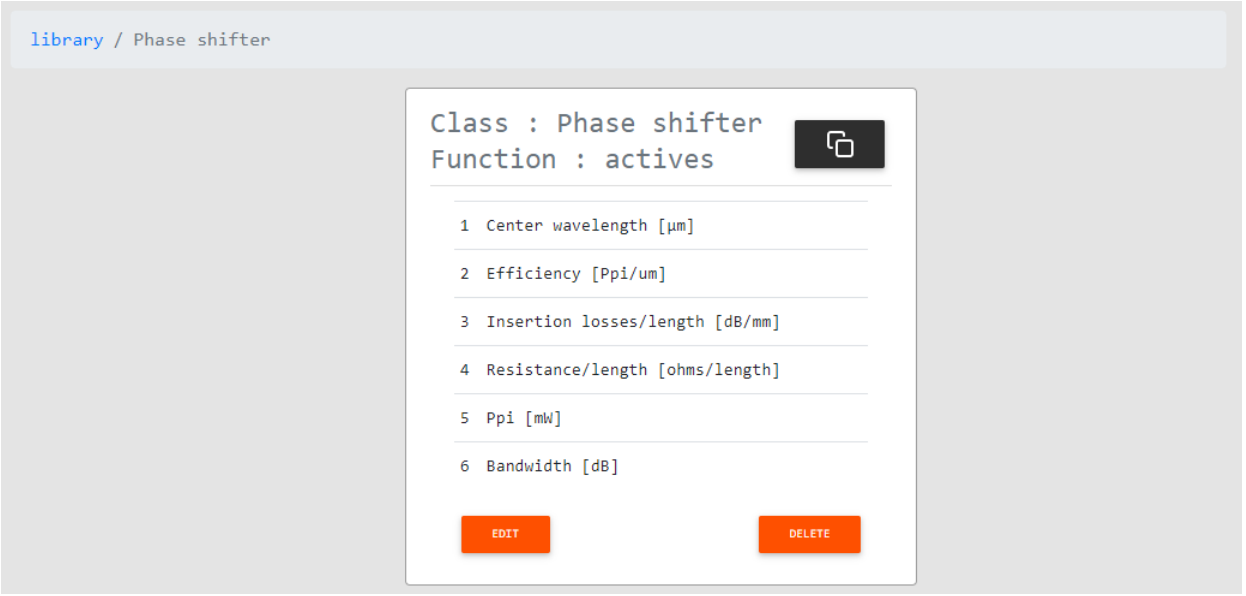

Figure 15 Definition view

- **edit_definition**
  Returns a form prefilled with the definition attributes that can be updated through the submit button. Hold <ctrl> for multiple selection of outstanding parameters, <shift> for sequential selection and click back to deselect. Selected parameters are shown under the submit button.

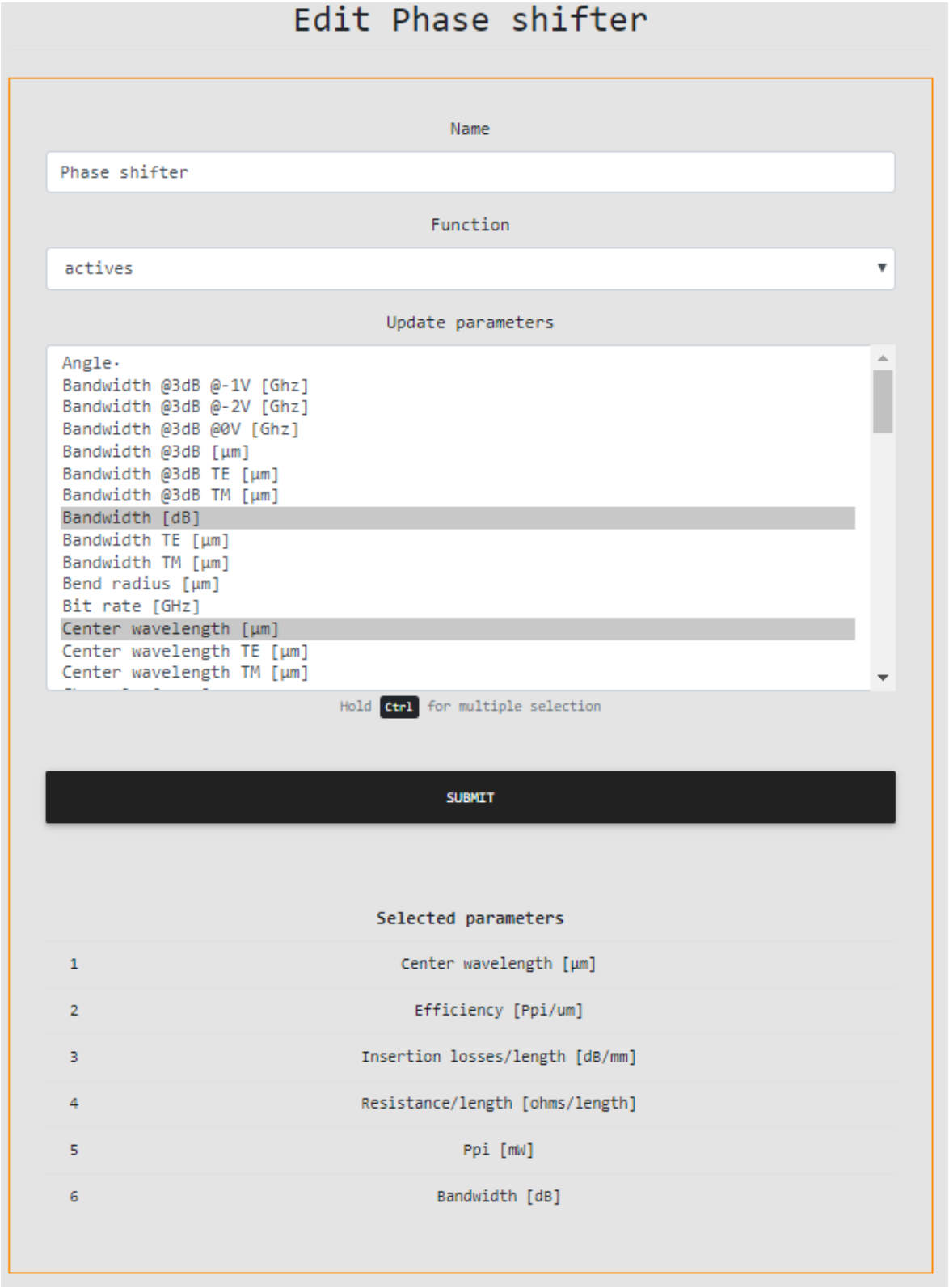

Figure 16 Edit definition

- **Components**
  Returns components' collection documents on cards. These are distinguished by their function or parameter "type" attribute. Function components are used to create (and filter) classes and parameters are used to create classes and blocks. Foundries has component role in **add view**.

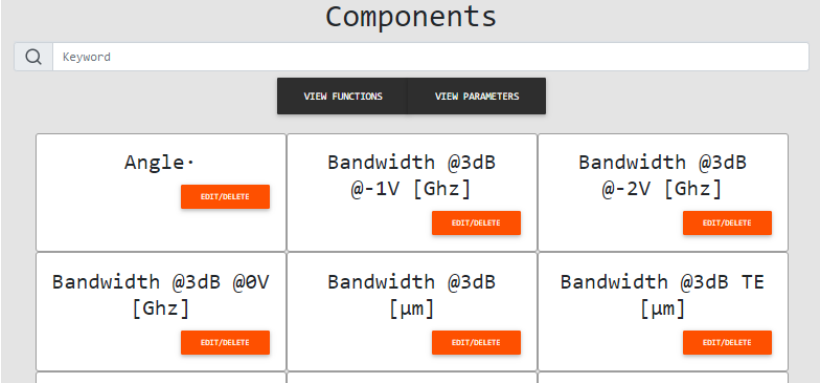

Figure 17 Components view

- **Add/addf/addc/addb**
  Returns forms to create components documents on collection. A name for the document is the only required attribute for creating the document. By starting to type the name a checkbox appears, when checked a list of documents' names appears allowing to search within existing collection to avoid duplicates.

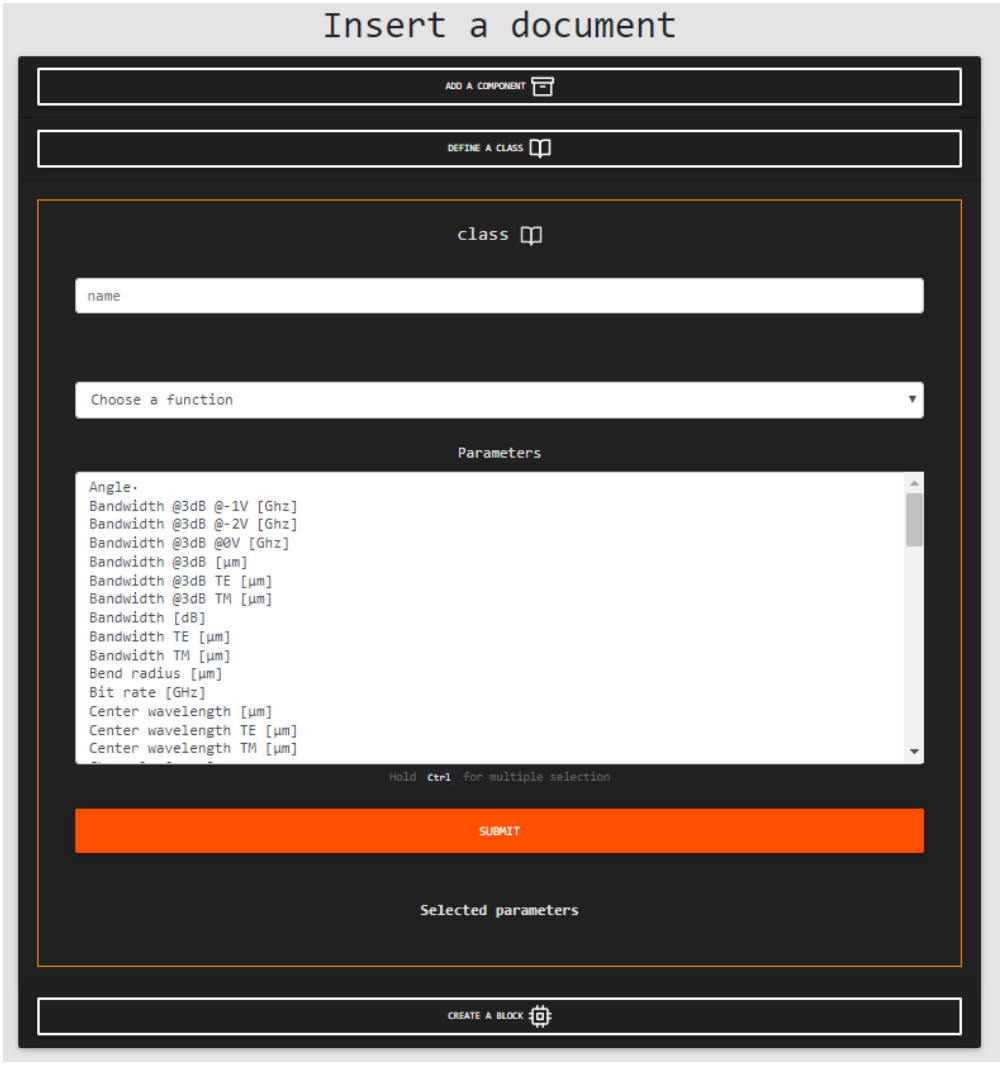

Figure 18 addc view