

20191120 - TECHNICAL DOCUMENTATION - V1.1

Alternative show off web application

Table of Contents

Document follow-up	3
Modifications	3
Validation remarks	3
Introduction	4
Minimal requirements	4
Project final structure	5
Getting started: Angular	6
Application structure	6
Routing	6
Data model	7
Data service	8
Display data	9
Compilation	12
Getting started: Express.js	14
Installation	14
Retrieving request body	16
Serving Angular files	18
Launch server	18
Download template	19
Required documentation	20
Angular	20
Node.js	20
Express.js	20

Document follow-up

Modifications

Date	Author	Description
20/11/2019	DF-OB	Creation
05/12/2019	DF-OB	Update retrieving request body

Validation remarks

Date	Author	Description

Introduction

This document will describe how to achieve the result you can preview on <https://alternativeshowoff.azurewebsites.net> via the Alternative Show Off step in Winner Bizz. It will review the technical requirements and steps to reproduce this web application.

This process uses a Node.js server framework, Express.js, and the famous JavaScript framework, Angular.

Therefore, we will only talk about those two frameworks without going over Winner Bizz part (which is the section below the black rectangle). This document will begin with creating an Angular application, then a server that will serve the web application.

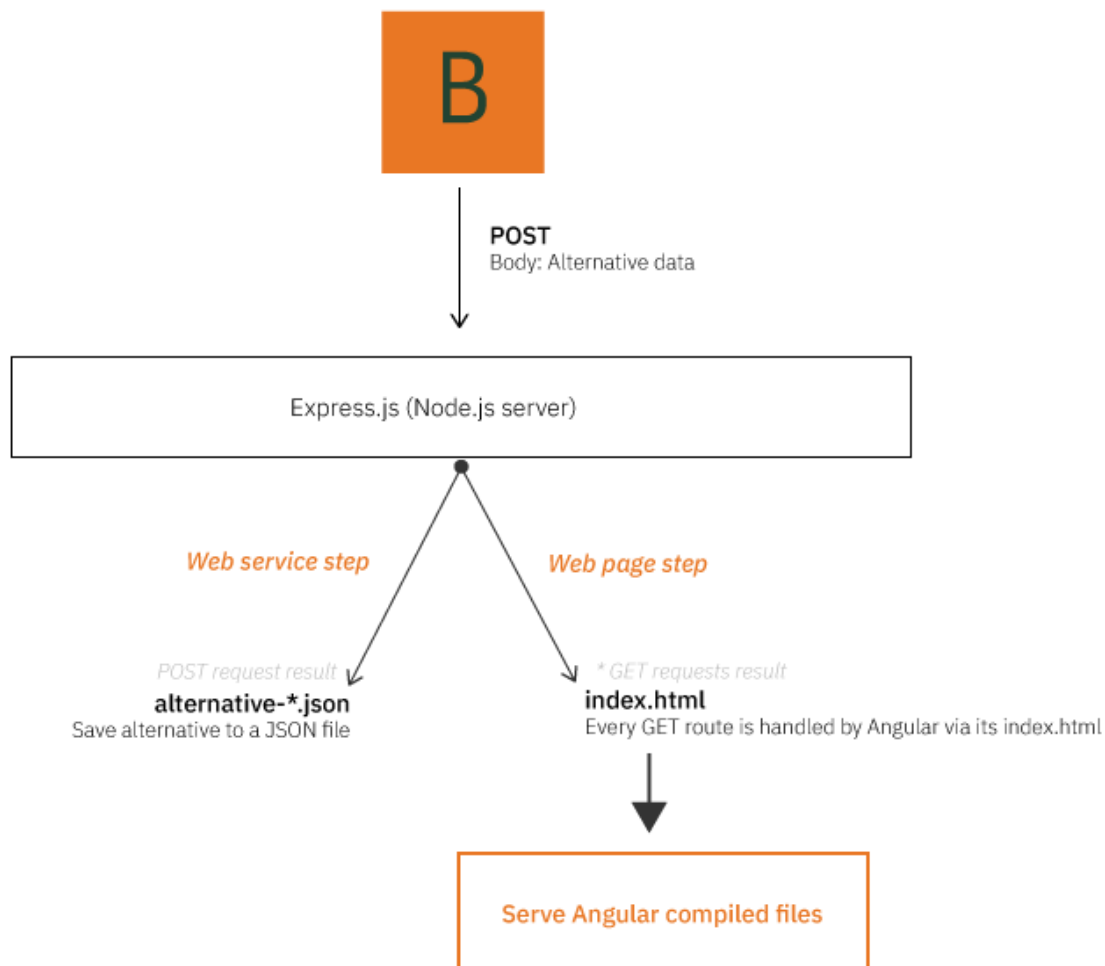


Figure 1 - Application global process

A functional template is available to download at the end of this document.

Minimal requirements

Node: ^10.16.0

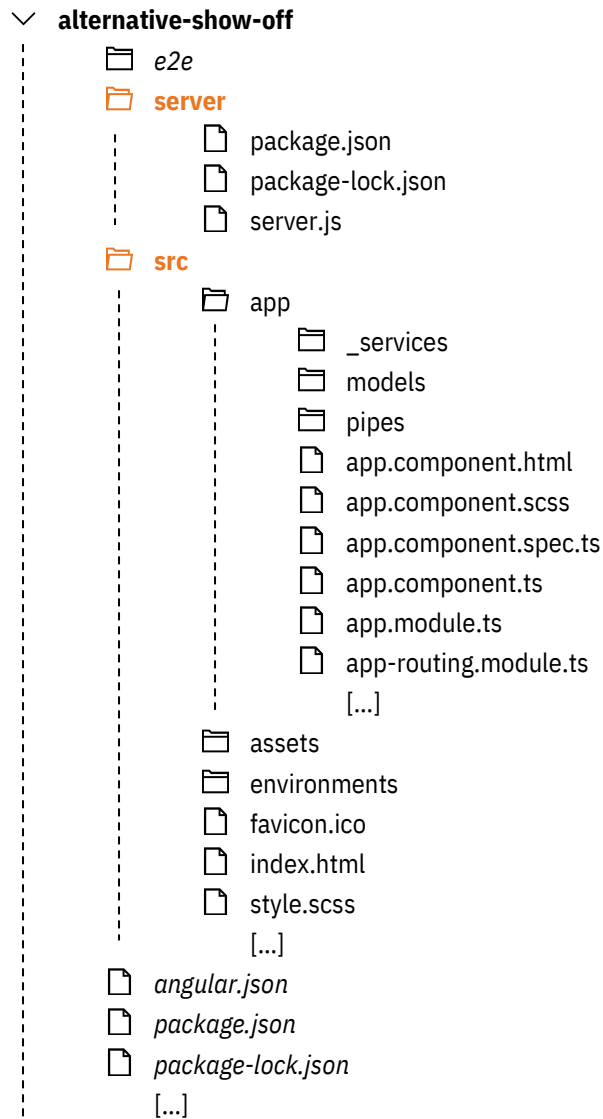
Npm: ^6.11.1

Express.js: ^4.17.1

@angular/cli: ^8.3.18

Project final structure

The project will have the following final structure:



Only the colored folders and their content will be discussed over this documentation.

If you'd like to use the **Web service** step in Winner Bizz, go straight to Getting started: Express.js.

Getting started: Angular

Application structure

Angular is a platform for building mobile and desktop web applications. If you're not familiar with Angular, please go through [Getting Started tutorial](#) before pursuing this documentation.

Begin with create a new application wherever you like on your computer:

```
$ ng new alternative-show-off
```

Answer yes to question about Angular router. Choose the style processor of your choice.

To style the application, we used SCSS and PrimeNG by PrimeFaces. Install it by running the following commands at the root of your application directory:

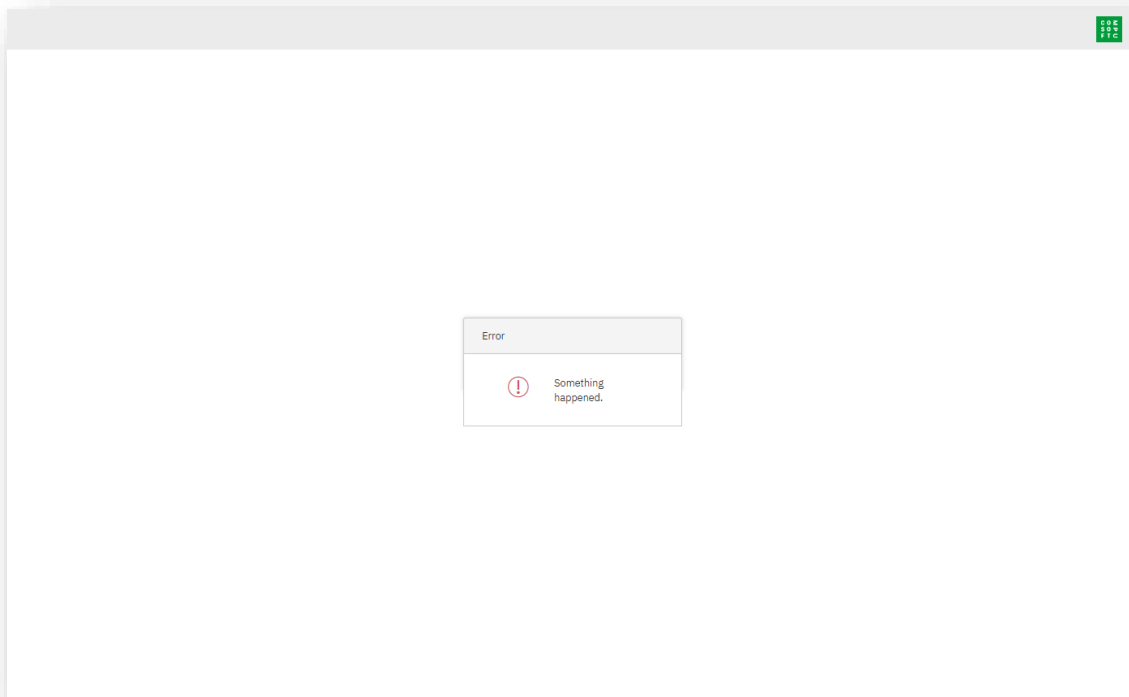
```
$ npm install primeng --save
$ npm install primeicons -save
$ npm install @angular/animations -save
```

Routing

Our application has three routes: one that shows off the alternative, a download page for the zip archive and a fallback route, to make sure every HTTP error is visually handled, as simple as it might be with just a message. The double stars “**” mean that for every item of the Routes array, if the router didn't hit any of these, or if the entered URL has no match, it will automatically show the FallbackComponent.

```
7  const routes: Routes = [
8      { path: 'alternative', component: AlternativeViewComponent },
9      { path: 'download', component: DownloadZipComponent },
10     { path: '**', component: FallbackComponent },
11 ];
```

The fallback component was very simple for this project, see below.



Data model

Because the alternative data is a very large dataset, the best approach is to create models according to the json nodes. The Angular command to do that is:

```
$ ng generate class models/class-name
```

This will create a “models” folder for you to tidy all your files.

For this project, we create a model named JsonData that will contain all the JSON data we’re receiving from Winner Bizz. As it has large information about invoicing, alternative or suppliers, we chose a generic name and will separate its children into more specific classes. It will act as our base class. To perform this, run the following command:

```
$ ng generate class models/json-data
```

```
1  import { Company } from './company';
2  import { Invoicing } from './invoicing';
3  import { AlternativeInfo } from './alternative-info/alternative-info';
4  import { Project } from './project';
5
6  export class JsonData {
7      Company: Company;
8      Customer: any;
9      Project: Project;
10     Invoicing: Invoicing;
11     AlternativeInfoCollection: AlternativeInfo[];
12     SupplierOrdersCollection: any;
13 }
```

This class is based on the json data structure, making sure each first level node gets its corresponding

class. We didn't create a model class for Customer and SupplierOrdersCollection as we don't want to display them in this project.

Here's the AlternativeInfo model we used.

```

1  import { Design } from '../design';
2  import { Address } from '../address';
3  import { VAT } from '../vat';
4  import { Currency } from '../currency';
5
6  export class AlternativeInfo {
7      ProjectNr: number;
8      SubProjectNumber: number;
9      SubAlternativeNumber: number;
10     AlternativeName: string;
11     AlternativeType: number;
12     Status: number;
13     AskToBeCancelled: string;
14     FinishedDate: Date;
15     AlternativeReference: any;
16     Offer: any;
17     DesignsCollection: Design[];
18     Delivery: {
19         DeliveryAddress: Address;
20     };
21     TotalVATsCollection: VAT[];
22     Currency: Currency;
23 }

```

Please refer to the Alternative Json structure documentation for more information on fields.

Data service

Creating a service, in a `_services` folder, to fetch data from the json file will help you follow the Angular style guide and the [Single Responsibility principle](#).

```
$ ng generate service _services/file
```

```

1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3  import { Observable } from 'rxjs';
4
5  @Injectable({
6      providedIn: 'root'
7  })
8  export class FileService {
9
10     constructor(private httpClient: HttpClient) { }
11
12     public getAlternativeJSON(alternativeId: string): Observable<string> {
13         return this.httpClient.get<string>(`../../assets/alternatives/alternative-${alternativeId}.json`);
14     }
15 }

```

This, via a HTTP GET request, fetches the content of the alternative JSON from a file created in a next step (See Getting started: Express.js).

On [line 14](#), we retrieve the same filename that will also be set at Retrieving request body section.

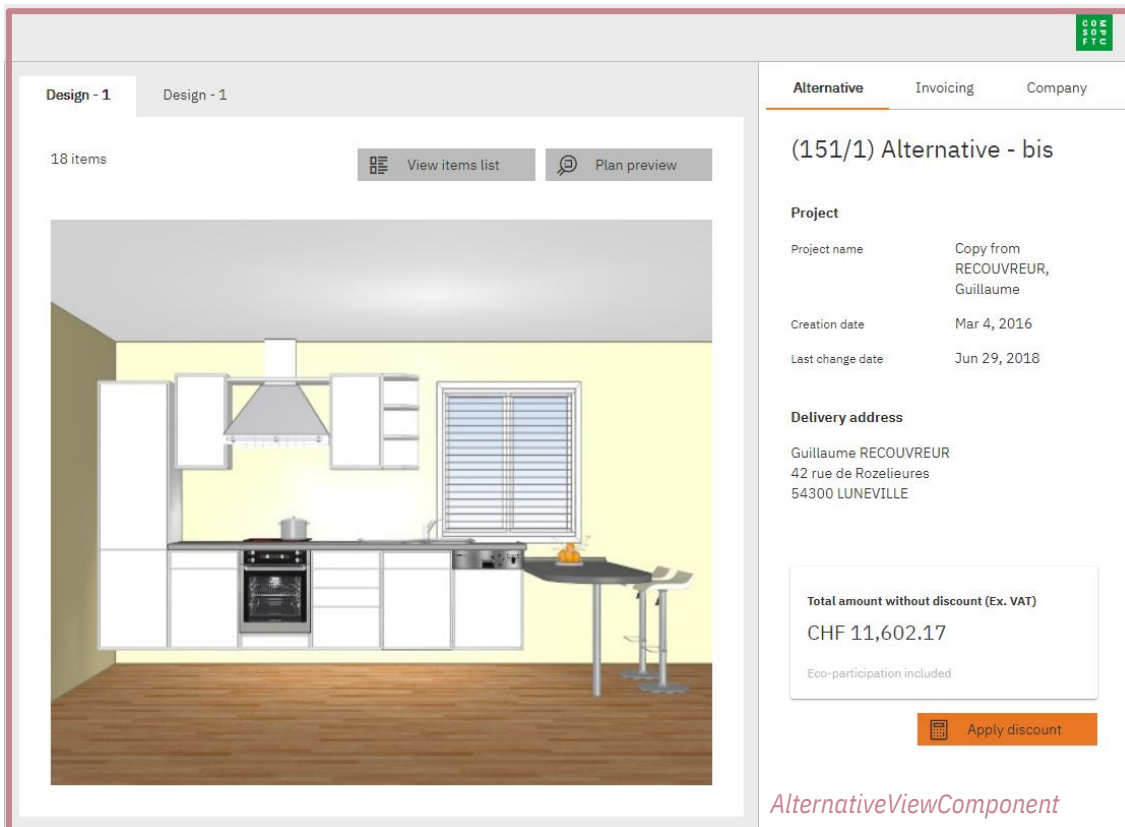
As specified, the result of this request is a string.

Display data

To display the alternative data, there are three required components: alternative-view, designs-view and items-list. The other components available in the template are here for factoring and styling purposes.

```
$ ng generate component alternative-view
$ ng generate component designs-view
$ ng generate component items-list
```

- AlternativeViewComponent



Displays the alternative global info. It is the base component that contains our web application. In the HTML part, we part the view in two sides – right, there is the alternative information (Alternative, Invoicing, Company) and left, there's the designs' plan and perspective previews. By default, it is the perspective view that's shown at the application launch.

A button at the top right corner of the left part allows you to switch to plan's preview. Another displays the design's items list.

In this component, we will call the FileService in order to get the alternative-*.json (See Getting started: Express.js - Retrieving request body).

```
59 this._files.getAlternativeJSON(this.alternativeId).subscribe(data =>{...},
```

_files: is the FileService variable.

We, then, convert the response to our JsonData class that will contain the information we want.

```

67     this._files.getAlternativeJSON(this.alternativeId).subscribe(data => {
68         if (data !== null) {
69             try {
70                 this.jsonData = JSON.parse(data)
71             } catch (err) {
72                 this.jsonData = <JsonData><unknown>data
73             }

```

In component's typescript file, we save the `jsonData.AlternativeInfoCollection[0]` variable into "alternativeInfo" to make it more readable. From the variable, we can retrieve all information needed as Angular uses brackets variables, as any templating engine would do.

```

69         // Get first level json nodes as entities
70         this.company = this.jsonData.Company;
71         this.invoicing = this.jsonData.Invoicing;
72         this.project = this.jsonData.Project;
73         this.alternativeInfo = this.jsonData.AlternativeInfoCollection[0];
74         this.designs = this.jsonData.AlternativeInfoCollection[0].DesignsCollection;

```

Here's an example from the first tab, displaying alternative's name, and project info (name, creation date, last change date).

```

19         <h2 class="inline-block ">{{alternativeInfo.AlternativeName}}</h2>

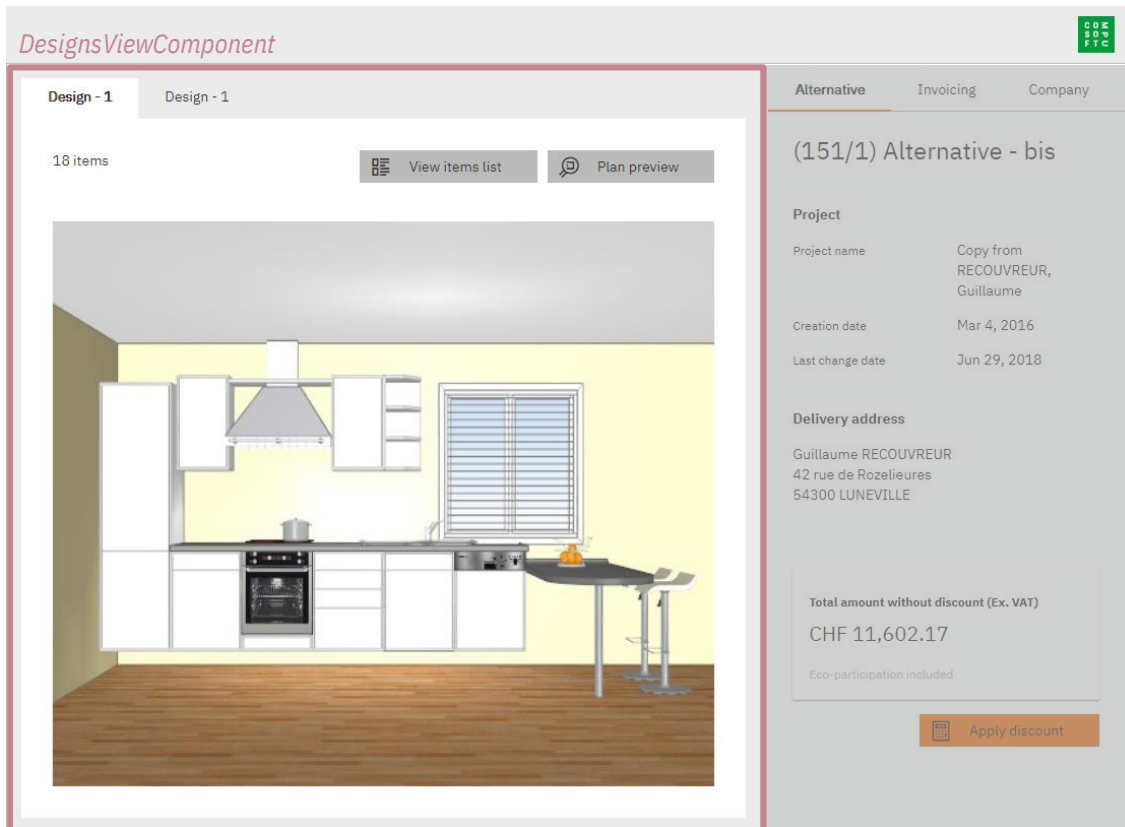
```

```

25         <div class="p-grid">
26             <div class="p-col-12 p-sm-6 no-left-padding">
27                 <span class="caption-label">Project name</span>
28             </div>
29             <div class="p-col-12 p-sm-6">
30                 {{project.Name}}
31             </div>
32         </div>
33         <div class="p-grid">
34             <div class="p-col-12 p-sm-6 no-left-padding">
35                 <span class="caption-label">Creation date</span>
36             </div>
37             <div class="p-col-12 p-sm-6">
38                 {{project.CreateDate | date}}
39             </div>
40         </div>
41         <div class="p-grid">
42             <div class="p-col-12 p-sm-6 no-left-padding">
43                 <span class="caption-label">Last change date</span>
44             </div>
45             <div class="p-col-12 p-sm-6">
46                 {{project.LastChangeDate | date}}
47             </div>
48         </div>

```

- DesignsViewComponent



DesignsViewComponent is a sub-component of AlternativeViewComponent. This component displays perspective and plan previews of each design as tabs for an easy navigation.

```

17 <div class="p-col-8 main">
18 <app-fallback [message]="errors.getDesigns.message"
19   [style]='center-dialog'
20   *ngIf="errors.getDesigns.message !== '' else showDesigns"></app-fallback>
21 <ng-template #showDesigns>
22   <app-designs-view [designs]="designs"></app-designs-view>
23 </ng-template>
24 </div>

```

For each design, we create a tab and displays its specific info such as items number and alternative and plan previews.

```

<div class="p-grid">
  <div class="p-col-12">
    <p-tabView [ngClass]='filled'>
      <p-tabPanel *ngFor="let design of designs; let i = index" [selected]='i == 0'
        [header]='design.DesignName'
        [ngClass]='filled'
        headerStyleClass="filled">
          <ng-template pTemplate="content">
            <div class="p-grid group-box">
              <div class="p-col-12 p-sm-2">
                <p>{{design.ItemLinesCollection.length}} items</p>
              </div>
            </div>
          </ng-template>
        </p-tabPanel>
      </p-tabView>
    </div>
  </div>

```

- ItemsListComponent:

The screenshot displays a software interface with a table of furniture items and a sidebar on the right. The table is titled 'Furniture' and contains 8 rows of data. The sidebar on the right shows project details for '(151/1) Alternative - bis'.

Code	Quotation quantity	Price Ex. VAT	VAT %	Description	
Furniture					
EV60-188	1 pce	CHF 773.28351	20 %	Storage unit, 1 door, 2 fixed shelves, 2 adjustable shelves D=60cm, W=60cm, H=18...	
U60	1 pce	CHF 226.05	20 %	Base unit, 1 drawer 1 door, 1 adjustable shelf W=60cm	
UH60	1 pce	CHF 217.70687	20 %	Oven unit for built in oven W=60cm, 1 panel	
US60	1 pce	CHF 631.91541	20 %	Drawer unit, W=60cm 3 drawers, 1 pull out	
SO60	1 pce	CHF 342.1108	20 %	Sink base unit, W=60cm 1 door, 1 panel	
GSB60	1 pce	CHF 178.12381	20 %	Door front for integrated Dishwasher in front finish, W=60cm	
H45-68	1 pce	CHF 268.59939	20 %	Wall unit, 1 door, 2 adjustable shelves, W=45cm, H=68cm	
KPG33-90	1 pce	CHF 56.54724	20 %	Gallery shelf KP 31, 2cm depth (Wall unit depth) 90cm, gloss, price plus cornic...	

Project Details:

- Project name:** Copy from RECOUVREUR, Guillaume
- Creation date:** Mar 4, 2016
- Last change date:** Jun 29, 2018
- Delivery address:** Guillaume RECOUVREUR, 42 rue de Rozelieures, 54300 LUNEVILLE
- Total amount without discount (Ex. VAT):** CHF 11,602.17
- Eco-participation included:**
- Apply discount:**

Like components above, you can access design's items list with brackets and JSON-like object fields:

```

16 <ng-template pTemplate="body" let-rowData let-rowIndex="rowIndex" let-columns="columns">
17 <tr class="ui-widget-header" *ngIf="rowGroupMetadata[rowData.Group].index === rowIndex">
18 <td colspan="8">
19 <span class="base-alt">{{rowData.Group}}</span>
20 </td>
21 </tr>
22 <tr>
23 <td class="ui-resizable-column">{{rowData.Code}}</td>
24 <td class="ui-resizable-column">{{rowData.QuotationQuantity}} {{rowData.Unit.Value}}</td>
25 <td class="ui-resizable-column priceExVat">{{design.Currency.CurrencyLocalCode}} {{rowData.Prices.PriceExVat}}</td>
26 <td class="ui-resizable-column">{{rowData.Prices.VATPercent}} %</td>
27 <td class="ui-resizable-column text-left">{{rowData.Description | stringTruncate}}</td>
28 </td>
29 <button pButton type="button"
30 [disabled]="rowData.Description == ''"
31 icon="cs-S-ZoomIn"
32 class="full-transparent"
33 (click)="showItemDetails(rowIndex)"></button>
34 </td>
35 </tr>
36 </ng-template>

```

Here's a sneak peek at how the items table is built. The style framework allows us to group items in a table. We then choose the information we want to display: ItemLineId, Code, QuotationQuantity with its Unit, the PriceExVAT, the VATPercent and the item Description. The last column generates a button to show the item entire description.

Compilation

When you're satisfied with the look of your application, you need to build it. Angular comes with integrated commands, the build one is also shipped with it. Open a terminal window, move to your application root folder and enter the following command:

```
$ ng build --prod="--true"
```

This will ensure that the build is in production mode. If there's no error in your code, the process stops automatically; if there is, the console will show you in which file it is located.

In the root folder of your application, a new folder has been created: the **dist** folder. It contains all files required for the application to work on an external server the same way it worked on your local computer: TypeScript files are turned into compiled Javascript, SCSS files are turned into CSS, ...

```
Generating ES5 bundles for differential loading...
ES5 bundle generation complete.

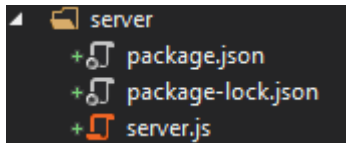
chunk {2} polyfills-es2015.2987770fde9daa1d8a2e.js (polyfills) 36.4 kB [initial] [rendered]
chunk {3} polyfills-es5.ef4b1e1fc703b3ff76e3.js (polyfills-es5) 122 kB [initial] [rendered]
chunk {0} runtime-es2015.edb2fcf2778e7bf1d426.js (runtime) 1.45 kB [entry] [rendered]
chunk {0} runtime-es5.edb2fcf2778e7bf1d426.js (runtime) 1.45 kB [entry] [rendered]
chunk {1} main-es2015.60f2891e251c0843ce6a.js (main) 993 kB [initial] [rendered]
chunk {1} main-es5.60f2891e251c0843ce6a.js (main) 1.05 MB [initial] [rendered]
chunk {4} styles.35f773614967aa822b84.css (styles) 247 kB [initial] [rendered]
Date: 2019-11-21T12:52:46.162Z - Hash: 74d58d486d716c7285d3 - Time: 38441ms
```

From now on, you could deploy your **dist** folder as it is and your application would work like a static website, but it wouldn't receive the alternative data (POST requests don't work on client side). That's why we now have to setup our Express.js server.

Getting started: Express.js

Installation

To perform a POST Request on client side, it is mandatory to create an instance of a node server. To do so, we use a node.js framework: Express. This server will have the following structure:



First, at the root of the Angular application, create a directory named server and then move the terminal into it. Then initialize a node directory into this folder.

```
$ mkdir server
$ cd server
$ npm init
```

This is what the console displays after the third command:

```
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (server)
version: (1.0.0)
description:
entry point: (index.js) server.js
test command:
git repository:
keywords:
author:
license: (ISC)
```

Enter the corresponding information when the prompt asks you to. We chose `server.js` as the entry-point name, but as you can see, it doesn't really matter what you choose as long as you stick to it.

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Is this OK? (yes)

This will create two files: `package.json` and `package-lock.json`. These files handle the node packages that the server requires in order to function.

The last command creates the server entry point. You can name also name it “app.js”, which is another frequent way to name it.

The next step installs libraries the server needs. Run the following commands. The `--save` option tells Node that these dependencies are required on development and production environments.

```
$ npm install express mime path body-parser --save
```

Then, open `server.js` in a code editor and add the following at the beginning:

```
1  // Initialize server dependencies
2  const express = require('express');
3  const mime = require('mime');
4  const path = require('path');
5  const bodyParser = require('body-parser');
6  const fs = require('fs');
7  const url = require('url');
8
9  const app = express();
10 const port = process.env.port || 3000
```

These lines use the packages we installed at the step before. We then initialize Express.js.

The last line sets up the port the application will be served on, setting 3000 as a fallback if default port isn't available.

```
12 // Set up static files directory
13 app.use(express.static(path.join(__dirname, '../dist/alternative-show-off')));
14
15 // Alternatives data can be very large, so we have to increase data size limit
16 app.use(express.json({ limit: '50mb' }));
17
18 // Use body-parser as a request middleware
19 app.use(bodyParser.urlencoded({
20   extended: true,
21   limit: '50mb'
22 }));
23 app.use(bodyParser.json());
```

Web Service: *Line 13 isn't necessary if you've chosen this step.*

Retrieving request body

Express allows you to easily create routes with request methods: GET, POST, PUT, DELETE... Our node server will only have two of them: one will receive the POST request initiated by Winner Bizz custom step and the other will redirect every route to the Angular files, letting the JavaScript framework displaying the information.

Web Service: *you don't have to use the GET route as you don't want to show a web page.*

Now, let's setup the POST route.

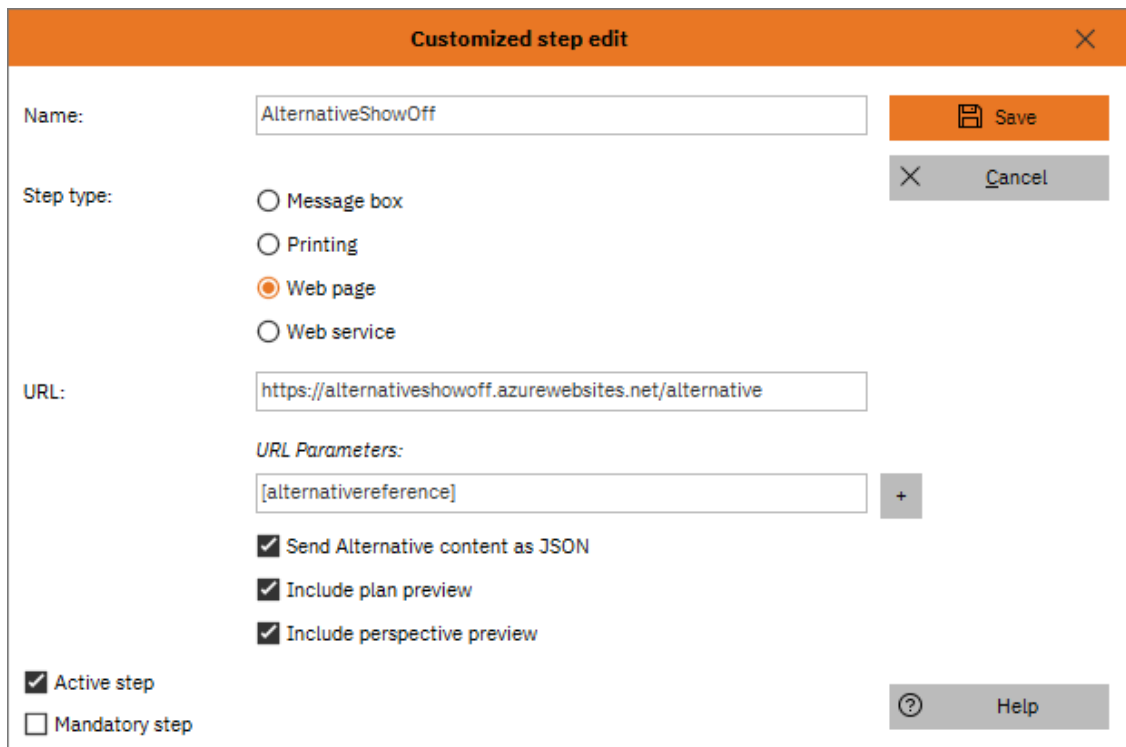
```
25 app.post("/alternative", (req, res) =>{...});
```

This creates a URL as follows : <https://alternativeshowoff.azurewebsites.net/alternative>, which only accepts POST requests and then executes a callback function with two arguments :

- **req:** identifies the incoming request
- **res:** identifies the response

Both variables detailed documentation are available at Express.js website (See Express.js)

First, if you ticked any parameter in Winner Bizz, you can retrieve them all with their identifier. An example for AlternativeReference:



Customized step edit

Name:

Step type:

- ☐ Message box
- ☐ Printing
- ☒ Web page
- ☐ Web service

URL:

URL Parameters:

+

☒ Send Alternative content as JSON

☒ Include plan preview

☒ Include perspective preview

☒ Active step

☐ Mandatory step

Save Cancel Help


```

33 // Retrieve [alternativereference] parameter
34 if (req.query.AlternativeReference !== null) {
35   alternativeID = req.query.AlternativeReference;
36 } else if (req.body !== "" && req.body.type !== undefined) {
37   // If it's not specified, retrieve [alternativereference] from request body
38   alternativeID = (req.body.data.AlternativeInfoCollection[0].AlternativeReference.AlternativeCode).replace('/', '_');
39 }

```

When you fetch the alternative, it comes as follows:

```

{" "data" ":"
"Company" ":" "{" ... "}"
  + "Customer" ":" "{" ... "}"
  + "Project" ":" "{" ... "}"
  + "Invoicing" ":" "{" ... "}"
  + "AlternativeInfoCollection" ":" "{" ... "}"
  + "SupplierOrdersCollection" ":" "{" ... "}"
}

```

Lots of extra quotes are added so before doing any processing on the data (req.body), we must clean it up.

```

42 // Format alternative JSON, as it's fetched as a string with root key named "data"
43 if (req.body !== null && alternativeID !== undefined) {
44   alternative = JSON.stringify(req.body.data, null, 2).replace(/\n|\r|\t/g, "");
45
46   fs.writeFile('../dist/alternative-show-off/assets/alternatives/${alternativeFilename}${alternativeID}.json', alternative, 'utf8', (err) => {
47     if (err) { console.log(err) }
48   });
49 }
50
51 if (!fs.existsSync('../dist/alternative-show-off/assets/alternatives/${alternativeFilename}${alternativeID}.json')) {
52   return res.status(200).send({ status: 404, id: `${alternativeFilename}${alternativeID}.json`, message: 'error' });
53 }

```

Web service: On [line 46](#) is where you deal with data contained in the **alternative** variable. To return a HTTP code, do as [line 52](#). If the server had trouble writing data to a .json file, it returns the following:

- HTTP Status: 200
- Request body:
 - o Actual HTTP Status: 404
 - o ID: "alternative-" + alternativeID
 - o Message: Error

After dealing with data (saving to a database, writing to a file, ...), go straight to Launch server.

Web page: If you'd like to display a web page, please go on.

After making sure the file is correctly written, redirect your POST route to any GET route, to get it to fall back into the next step.

```

53 res.setHeader("Content-Type", mime.getType(req.originalUrl));
54
55 res.redirect(url.format({
56   query: {
57     "AlternativeReference": alternativeID
58   }
59 }));
60 });

```

Line 53 is mandatory as it tells the server to distinguish static files by their extension. If we remove this line, it would consider all files as text/plain or as text/html, ignoring JavaScript or JSON types.

Serving Angular files

Now, we will tell the server that all (*) GET requests are sent to the index.html file generated by Angular.

```
62 // Serve angular dist files
63 app.get('*', (req, res) => {
64   res.setHeader("Content-Type", mime.getType(req.originalUrl));
65   res.sendFile("index.html", { root: '../dist/alternative-show-off' });
66 });
```

The framework will then deal with displaying the correct page on its own.

Launch server

```
68 app.listen(port, () => console.log(`AlternativeShowOff app listening on port ${port}!`));
```

Make sure you are in the server directory.

```
$ pwd
```

```
Path
----
.\server
```

If you're not, do the following:

```
$ cd server
```

And then launch the node server:

```
$ node server.js
```

```
AlternativeShowOff app listening on port 3000!
```

Download template

Download the application at <https://alternativeshowoff.azurewebsites.net/download>

Make sure every minimal requirement is fulfilled (See Minimal requirements section).

Unzip it wherever you like best on your computer and open a terminal console:

```
$ cd your-unzipped-folder-path
$ npm install
$ ng serve
```

A README.md is available at the root of the folder.

Required documentation

Angular

[Introduction to the Angular Docs](#)

[PrimeNG by PrimeFaces](#)

Node.js

[Documentation](#)

Express.js

[Documentation](#)