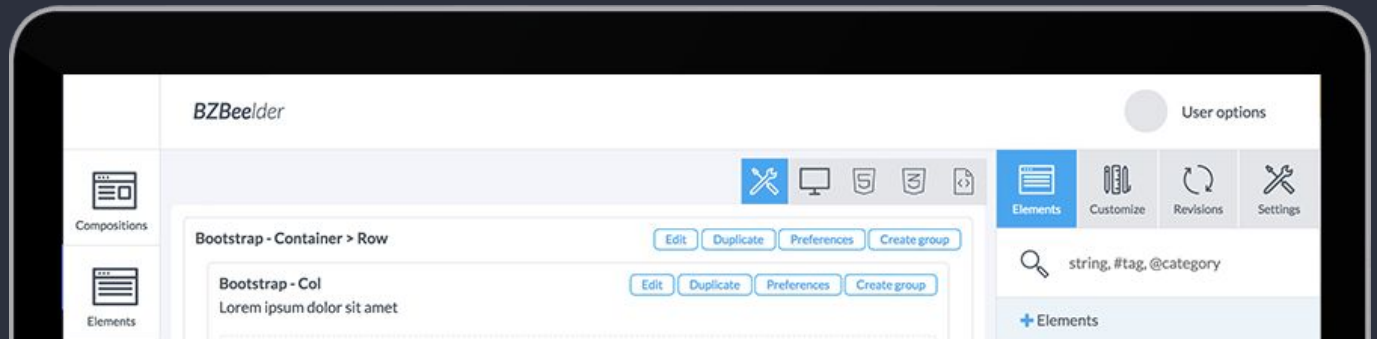

meet.js
kielce

CMS as a side project

Why it's good to have self side project



Who am I?

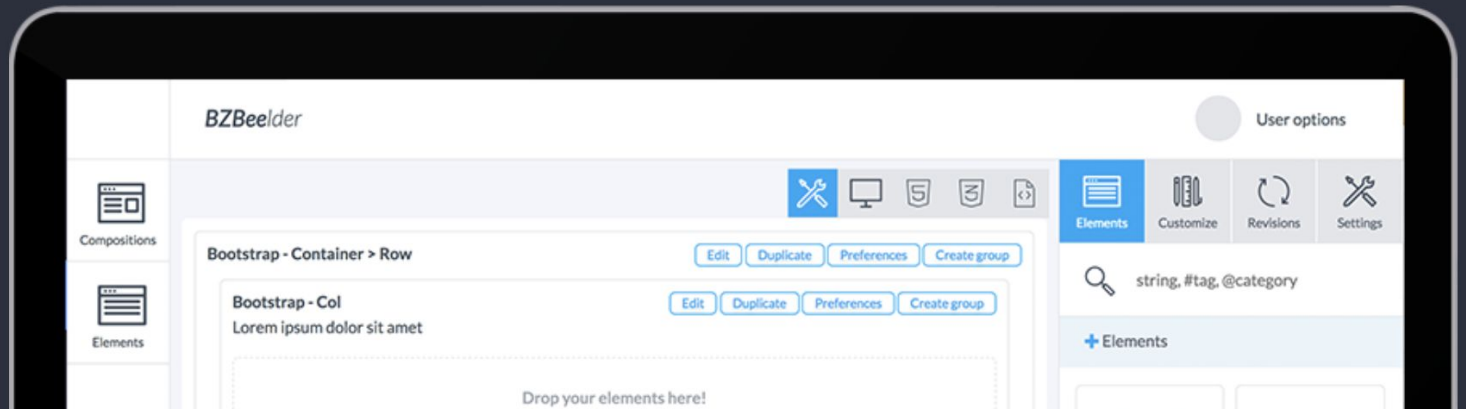
Blog: http://fedojo.com

GitHub: https://github.com/fedojo

VIRTUSLAB

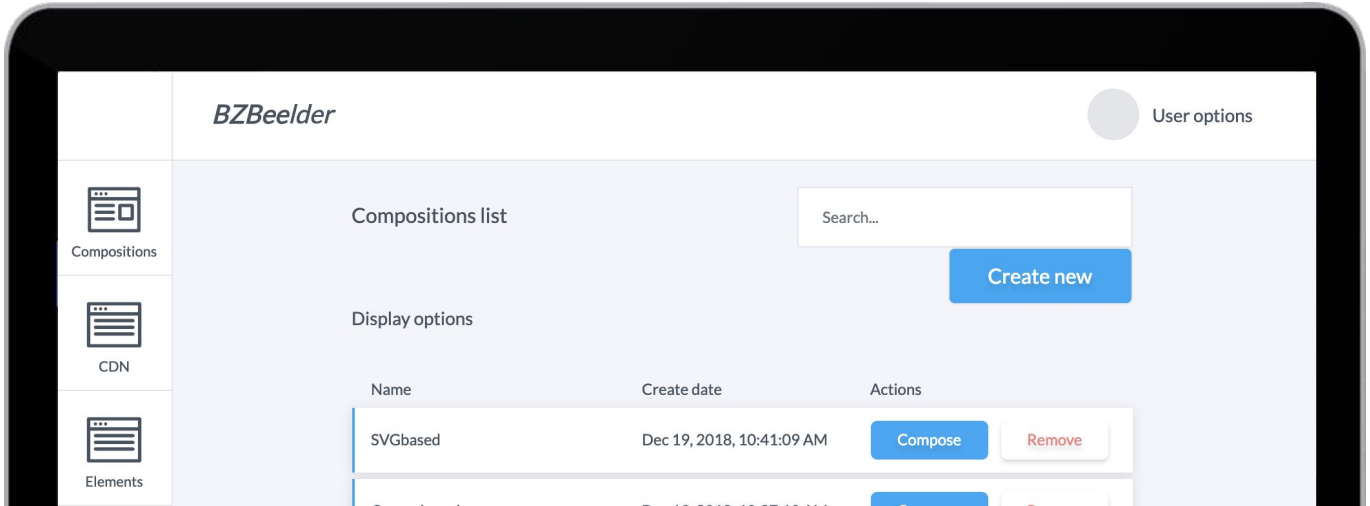
BZBeelder

Trendy approach done with Node.js and Angular

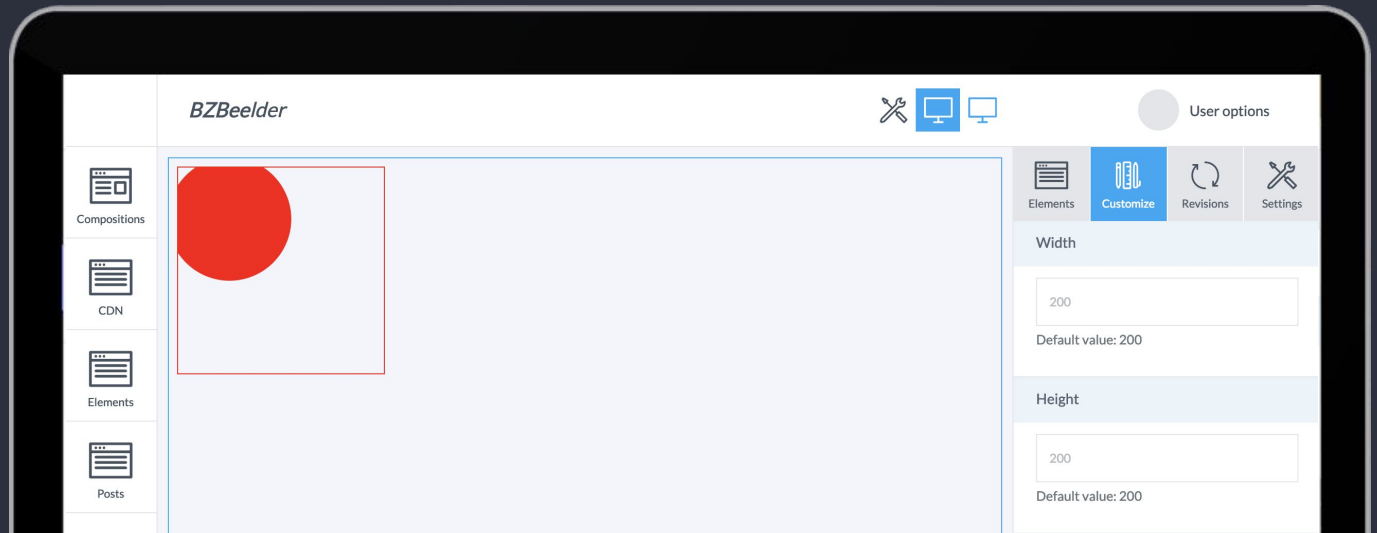


What is BZBeelder

- OpenSource CMS / framework (not yet fully published officially)
- HTML elements repository and website/template composer
- Blog engine
- Foundation for other purposes



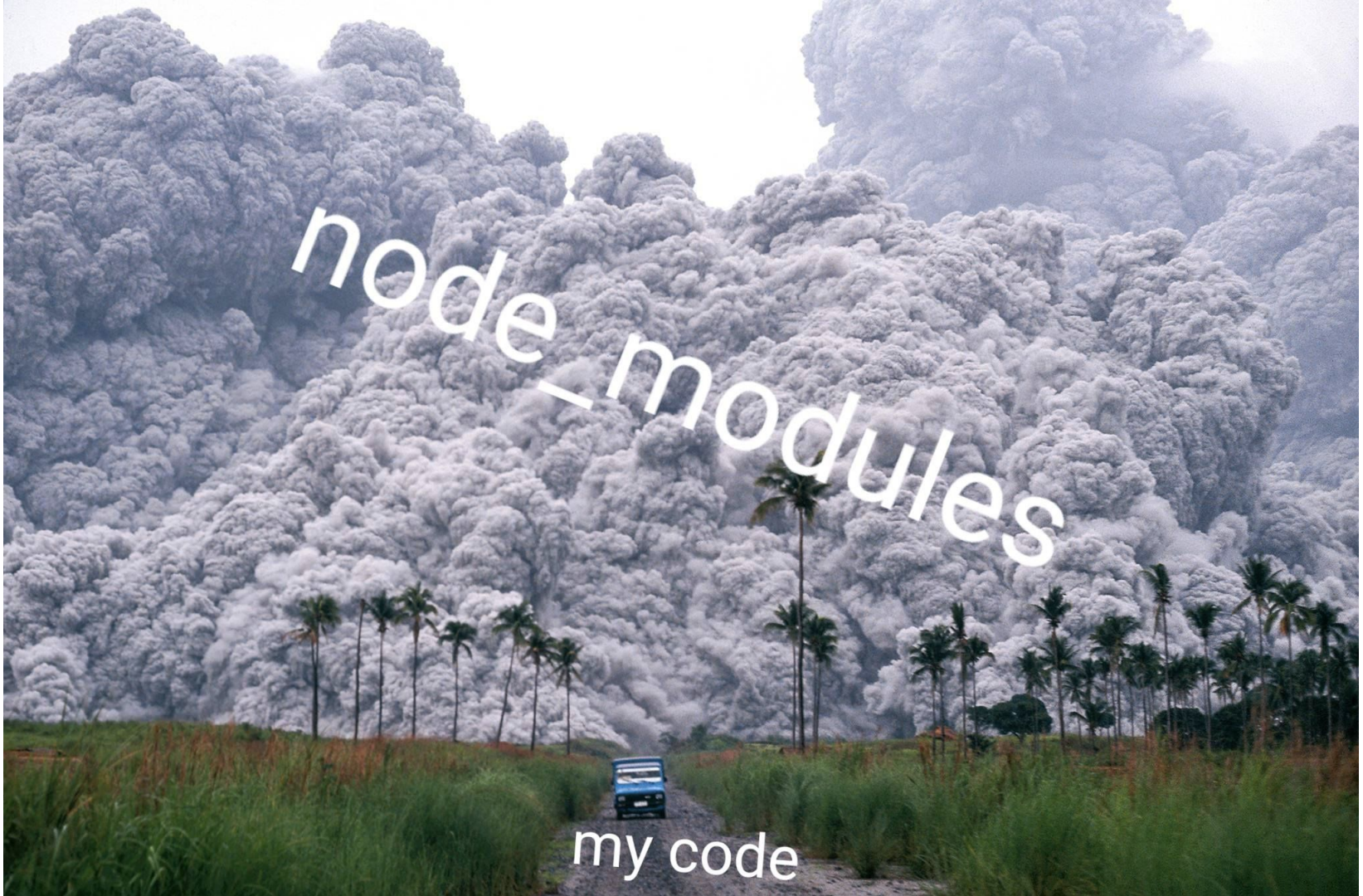
Tech stack


BZBeelder
User options
Compositions
CDN
Elements
Posts
Elements
Customize
Revisions
Settings
Width
200
Default value: 200
Height
200
Default value: 200

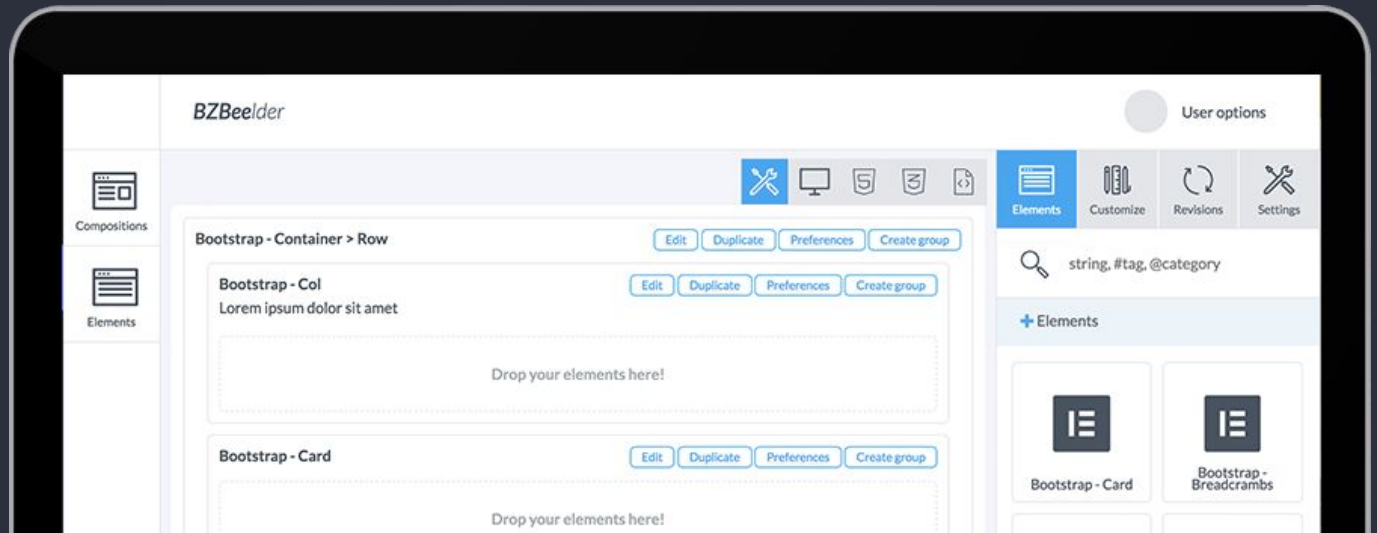
Tech stack

- NodeJS / ExpressJS / Mongoose
- MongoDB
- Angular 5

Another project based on MEAN stack...
BLEH!


node_modules
my code

CMS?


BZBeelder
User options
Compositions
Elements
Bootstrap - Container > Row
Edit
Duplicate
Preferences
Create group
Bootstrap - Col
Lorem ipsum dolor sit amet
Drop your elements here!
Bootstrap - Card
Drop your elements here!
Elements
Customize
Revisions
Settings
string, #tag, @category
+ Elements
Bootstrap - Card
Bootstrap - Breadcrambs

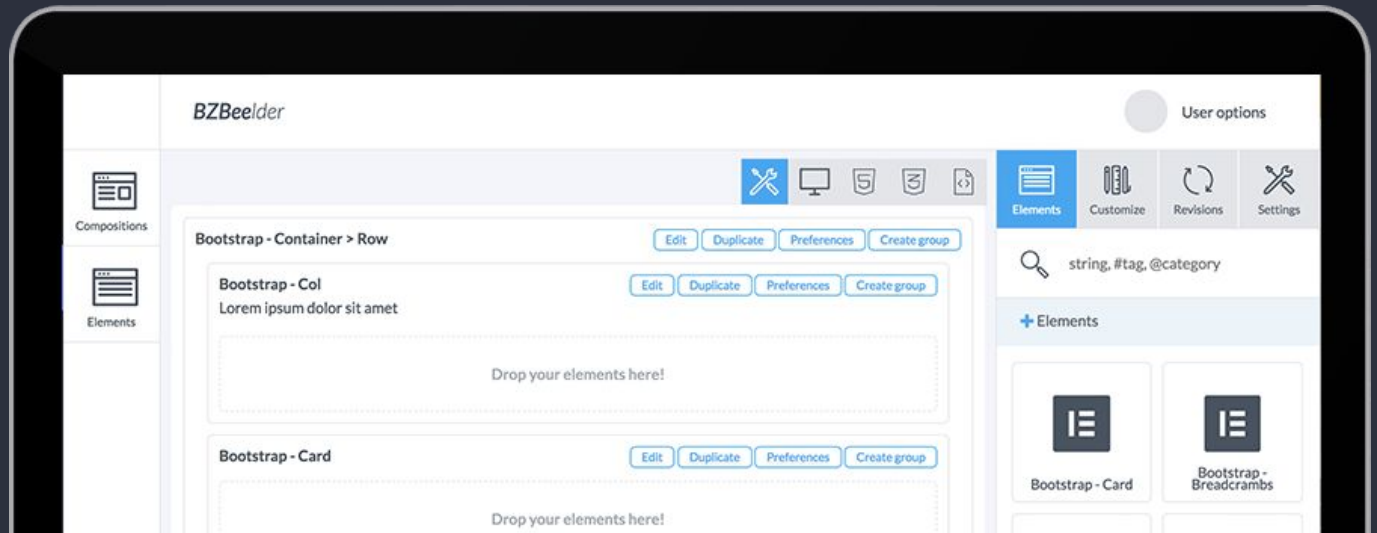
What is CMS?

Who is using CMS?

Client?

Developer?

Story


BZBeelder
User options
Compositions
Elements
Bootstrap - Container > Row
Edit
Duplicate
Preferences
Create group
Bootstrap - Col
Lorem ipsum dolor sit amet
Edit
Duplicate
Preferences
Create group
Drop your elements here!
Bootstrap - Card
Edit
Duplicate
Preferences
Create group
Drop your elements here!
Elements
Customize
Revisions
Settings
string, #tag, @category
+ Elements
Bootstrap - Card
Bootstrap - Breadcrambs

Why next CMS?

- Ive been using Wordpress / Typo3 / Drupal (a bit)
- I need something “own”
- I need something more elastic
- I need a tool which gives me opportunity to create projects based on previously created elements and repository of those elements

Why next CMS? ...

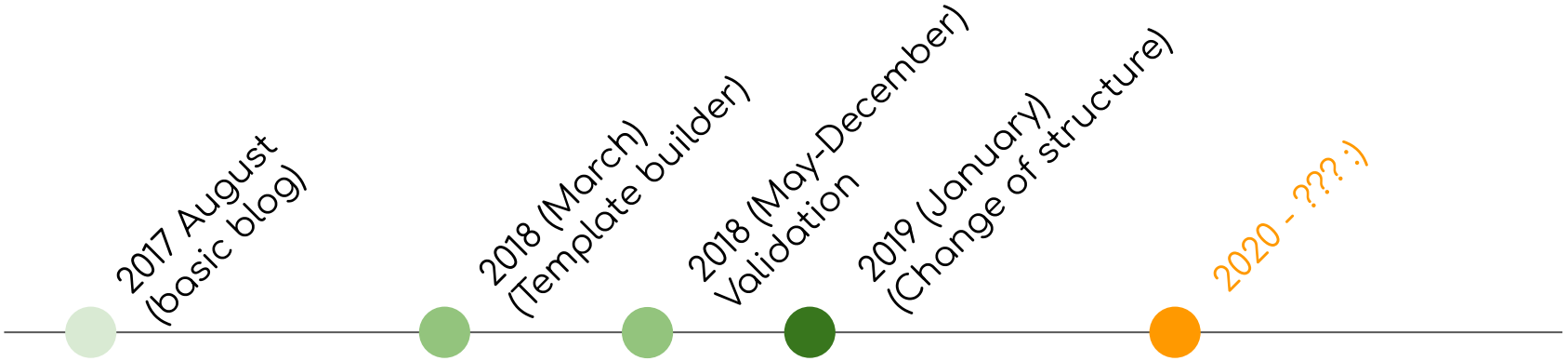
Pareto principle

80 / 20

90 / 10

From the beginning

- I've started from blog engine (about August 2017)
- Project was on hold for about 1 year
- I came back to project in march 2018



Main issues of building side projects

It's hard to spend your free time to build it

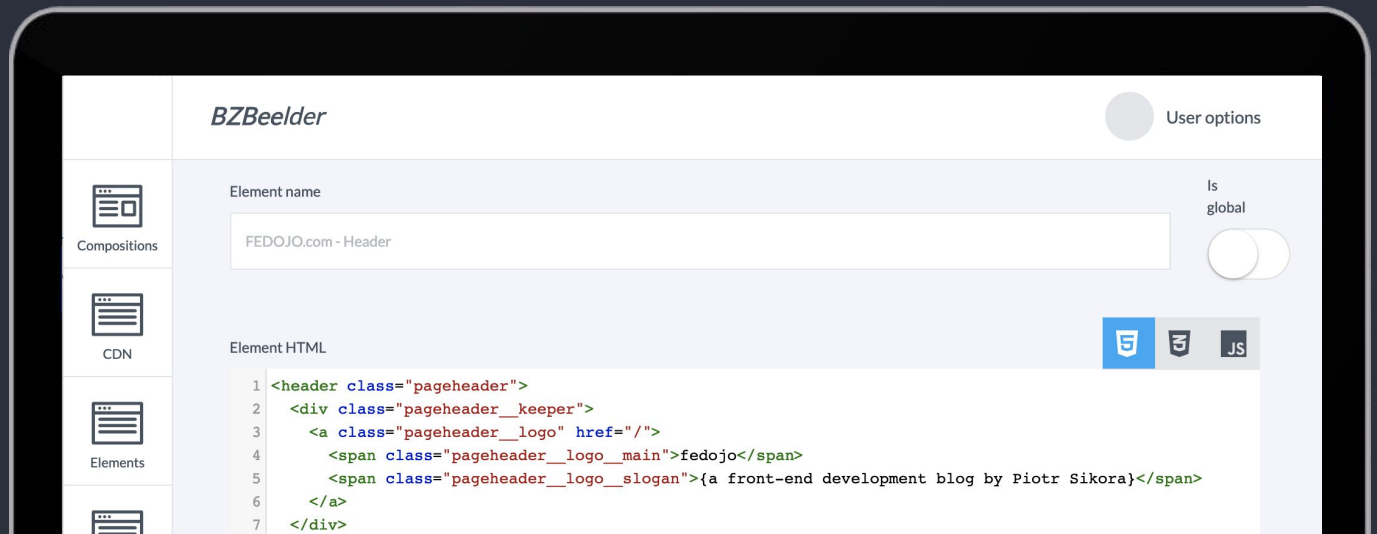
Main issues of building side projects

You still want to rebuild it - not release it

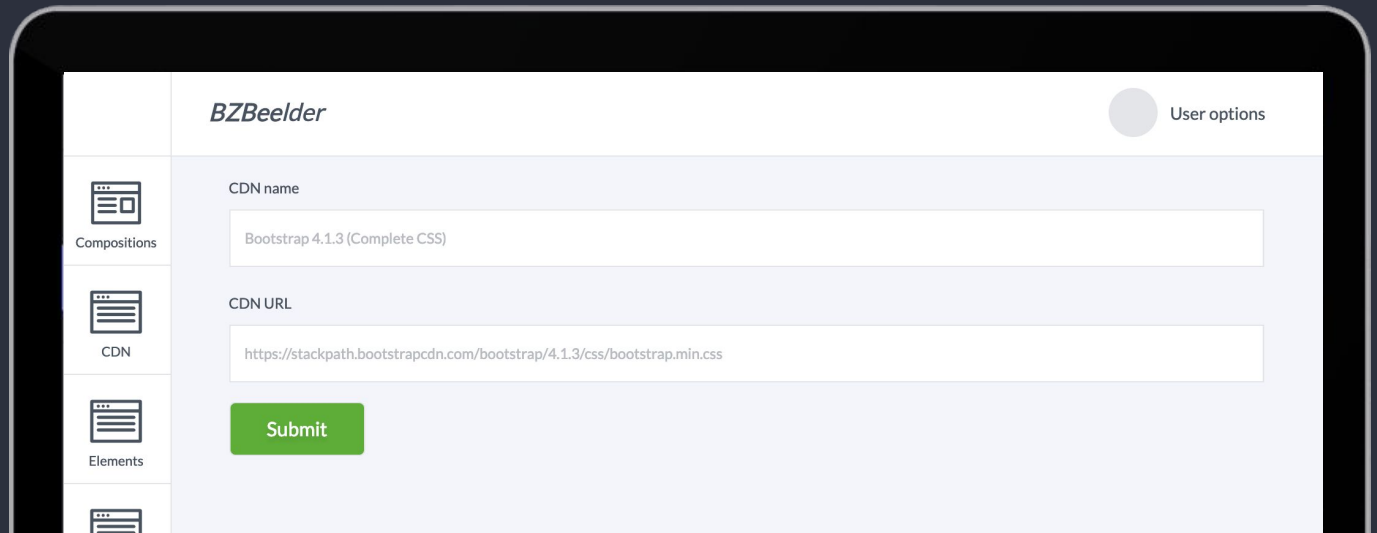
Why it’s good to have side project

- You can test new tech stacks without “naysayers“
- You can do what you want how you want and when you want
- You can easily customize your project as you know how to extend it
- You can test new dev tools

Process
Tools
Evolution



Step #1
Beginning




GREY GOOSE
IMPORTED

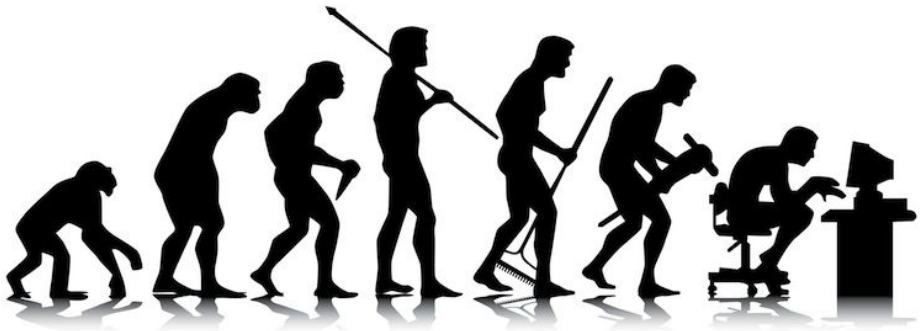
Step #1 - Beggining

1. **“Start small”** - Ive started with very small idea which I thought will take me max 2 days
2. It didn’t ... :) It took me 2 weeks
3. I had a road map in my mind - not on paper

- Listen, you're a developer. Tell me, why do you incorrectly estimate the time to create a software so often?
- Imagine you have to unload a truck. How much time will it take?
- A few hours.
- It's Kamaz.
- 8 hours.
- Kamaz is loaded with sand.
- 12 hours
- You do not have tools, just hands.
- 2 days.
- Outside is -40 degrees.
- 4 days.
- Kamaz stands under water.
- It does not make sense, you are constantly changing conditions!

Step #1 - Dry coding

- When I started I've been coding, coding, coding...
- No automated testing / No Unit testing
- Just manual testing



Issues #1

- I have no plan
- I have no deadline

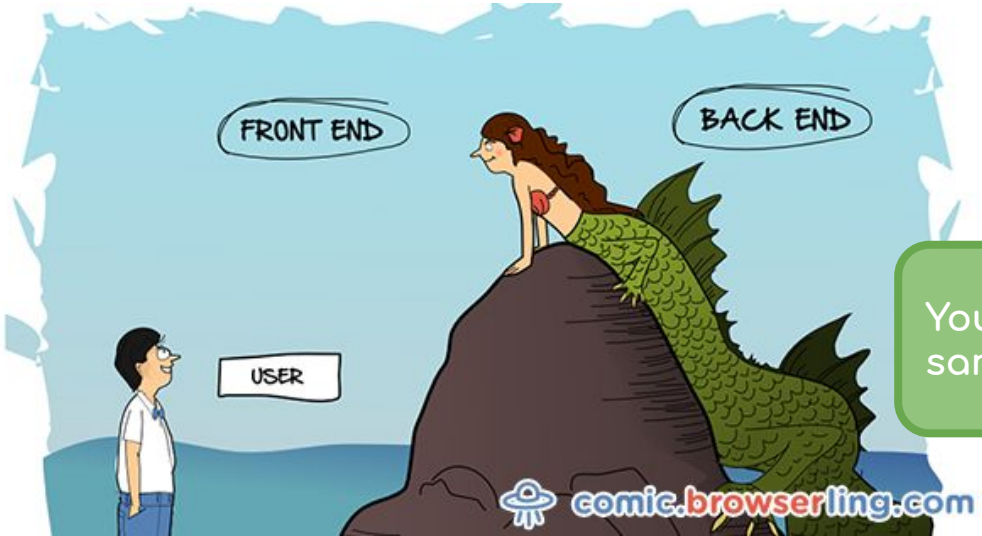
Lesson #1

- Start coding!
- Create roadmap
- Make a plan
- Create deadline / deadlines

Step #2
Dividing project

Step #2 - The process evolution

- I've divided project into two separate repositories (FE, BE)
- FE was a monolite without modules
- BE was a monolite without modules
- There were a lot of issues between FE and BE



You can be BA, FE , BE, QA, User same time :)

```
66  exports.findOne = (req, res) => {
67    Post.findById(req.params.id)
68      .then( resolve: post => {
69        if (!post) {
70          return res
71            .status(404)
72            .json({
73              message: 'Post not found with id ' + req.params.id,
74            });
75        }
76        res
77          .send(post);
78      }).catch( onReject: err => {
79      if (err.kind === 'ObjectId') {
80        return res
81          .status(404)
82          .json({
83            message: 'Post not found with id ' + req.params.id,
84          });
85      }
86      return res
87        .status(500)
88        .json({
89          message: 'Error retrieving post with id ' + req.params.id,
90        });
91    );
92  };
```

Lesson #2

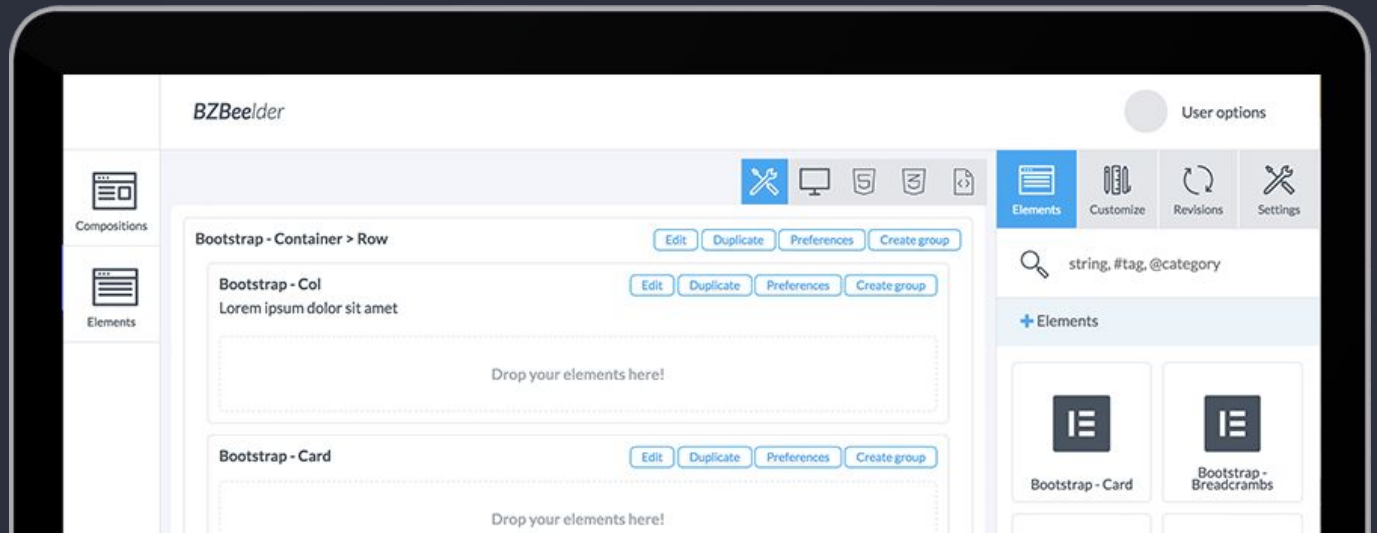
- Divide project into smaller modules (problems)

Issues #2

- I had one repo now two - it means that I had one issue now I have two :)
- Issues in communication between BE / FE (funny thing when you are full stack)

Step #3

Modules, testing, linters



Step #3 - Linting

```
✖ 1193 problems (1193 errors, 0 warnings)
  1097 errors, 0 warnings potentially fixable with the `--fix` option.
```

- Initially - pain in the ass...
- Linting helps you to keep standard in your code

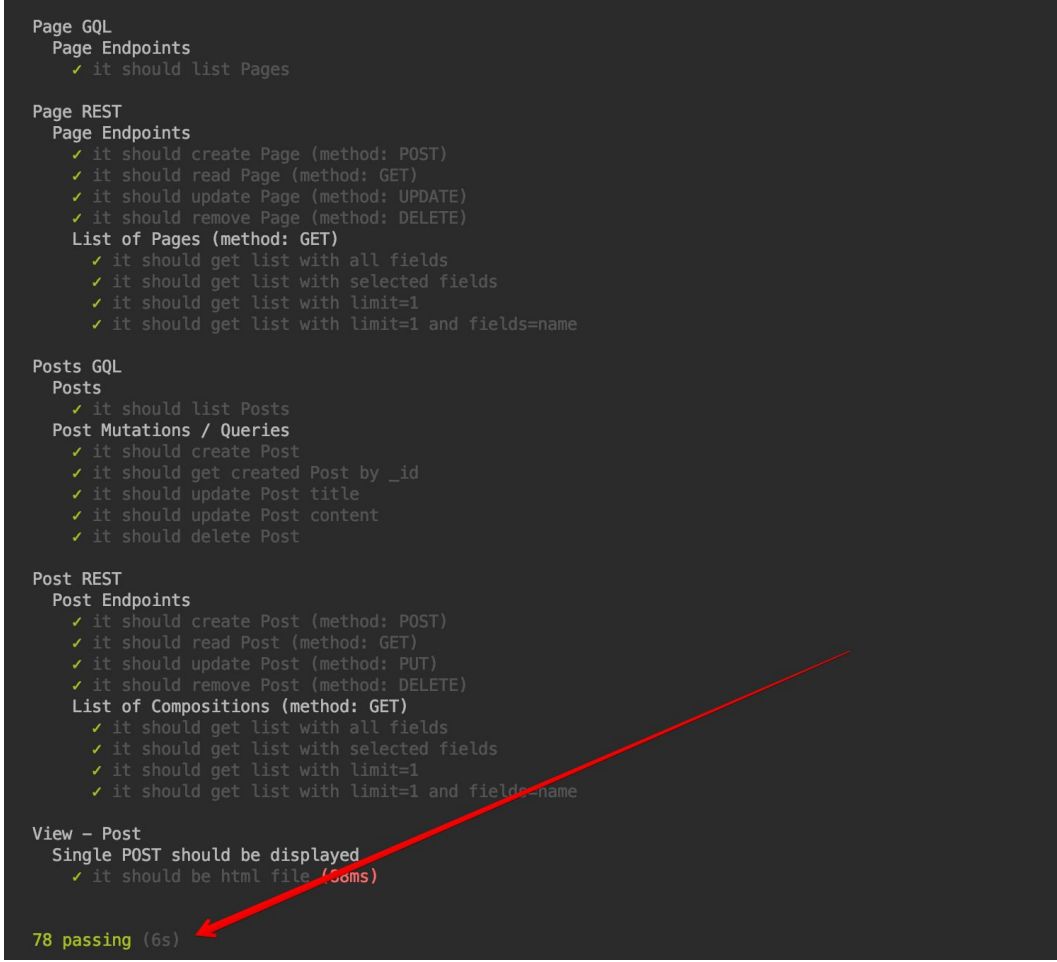
```
read: async (req, res) => {
  try {
    const post = await CRUDHelper.read(MODEL, req.params.id);

    res
      .status(200)
      .json(ResponseHelper.successResponse(post));
  } catch (e) {
    res
      .status(500)
      .json(ResponseHelper.errorResponse(err));
  }
},
```

Files changed (71)

+13	-5	R	eslintrc.json → .eslintrc.json (94% similar)
+2	-11	M	app.js
+0	-95	D	core/helpers/CRUDController.js
+25	-26	M	core/helpers/CRUDHelper.js
+6	-9	M	core/helpers/CacheHelper.js
+21	-18	M	core/helpers/ConditionHelper.js
+55	-55	M	core/helpers/ControllerHelper.js
+32	-28	M	core/helpers/FileHelper.js
+13	-13	M	core/helpers/QueryHelper.js
+21	-27	M	core/helpers/ResponseHelper.js
+0	-52	D	core/helpers/ThemeHelper.js
+29	-42	M	core/helpers/__test__/CRUDHelper.spec.js
+3	-3	M	core/helpers/__test__/QueryHelper.spec.js
+15	-15	M	core/helpers/__test__/ResponseHelper.spec.js
+6	-6	M	core/modules/assets-library/AssetModel.js
+86	-89	M	core/modules/assets-library/AssetsLibraryController.js
+0	-4	D	core/modules/assets-library/AssetsLibraryHelper.js
+8	-9	M	core/modules/assets-library/router.js
+8	-8	M	core/modules/auth/AuthCtrl.js
+19	-19	M	core/modules/auth/AuthHelper.js
+7	-8	M	core/modules/auth/AuthMiddleware.js
+58	-48	M	core/modules/auth/__test__/Auth.spec.js
+4	-3	M	core/modules/auth/router.js
+10	-13	M	core/modules/cdn/CdnController.js
+7	-2	M	core/modules/cdn/CdnHelper.js
+3	-3	M	core/modules/cdn/CdnModel.js
+9	-9	M	core/modules/cdn/CdnUtils.js
+166	-147	M	core/modules/cdn/__test__/Cdn.spec.js
+14	-16	M	core/modules/composition/CompositionController.js
+6	-6	M	core/modules/composition/CompositionModel.js
+5	-5	M	core/modules/composition/CompositionRevisionModel.js
+164	-140	M	core/modules/composition/__test__/Composition.spec.js
+1	-1	M	core/modules/composition/router.js
+2	-2	M	core/modules/element-free/router.js
+0	-102	D	core/modules/element-group/ElementGroupController.js
+0	-52	D	core/modules/element-group/ElementGroupModel.js
+0	-16	D	core/modules/element-group/router.js
+15	-20	M	core/modules/element/ElementController.js
+30	-30	M	core/modules/element/ElementHelper.js
+7	-8	M	core/modules/element/ElementModel.js
+10	-10	M	core/modules/element/ElementQuery.js
+7	-7	M	core/modules/element/ElementType.js
+144	-124	M	core/modules/element/__test__/Element.spec.js
+0	-2	M	core/modules/element/router.js
+15	-24	M	core/modules/page/PageController.js
+10	-11	M	core/modules/page/PageModel.js
+18	-18	M	core/modules/page/PageUtils.js
+1	-2	M	core/modules/page/router.js
+16	-3	M	core/modules/post/CacheController.js
+22	-22	M	core/modules/post/PostController.js
+28	-30	M	core/modules/post/PostHelper.js
+4	-4	M	core/modules/post/PostModel.js
+18	-18	M	core/modules/post/PostUtils.js
+134	-122	M	core/modules/post/__test__/Post.spec.js
+9	-9	M	core/modules/user/controllers/UserCtrl.js
+0	-38	M	core/modules/user/helpers/UserDummyHelper.js
+9	-13	M	core/modules/user/helpers/UserTokenHelper.js
+6	-6	M	core/modules/user/models/User.js
+4	-4	M	core/modules/user/models/UserGroup.js
+3	-3	M	core/modules/user/models/UserRole.js
+6	-6	M	core/modules/user/models/UserToken.js
+21	-22	M	core/modules/view/controllers/CompositionViewController.js
+21	-21	M	core/modules/view/controllers/PageViewController.js

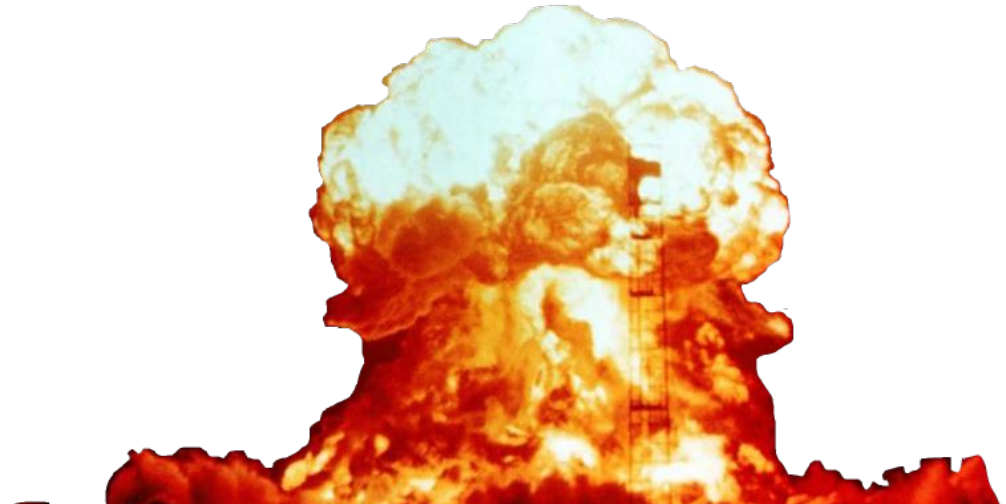
Step #3 - Testing

- Automatic testing of REST API
  - Chai
  - Chai-http
- Unit testing of basic functions
  - Chai



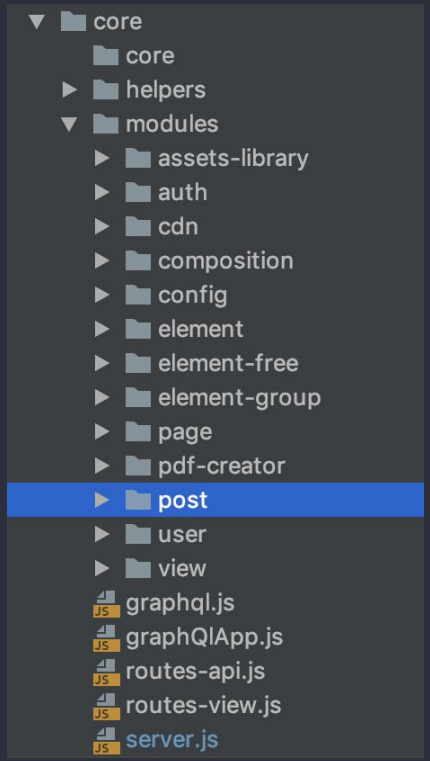
Step #3 - Unit testing

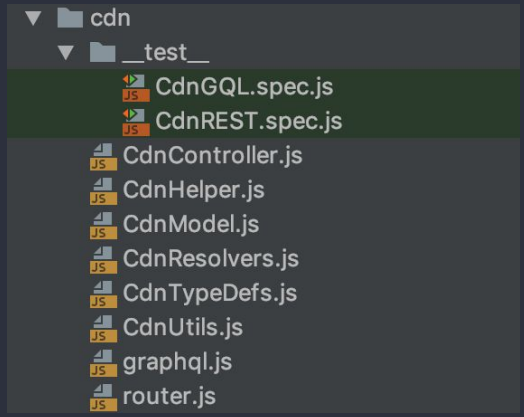
- Because of unit tests which are covering manual part of work I could start extending software easily.
- Tests are runned after each change
- Tests are runned before each commit and push



Step #3 - Modularisation

- BE was a divided into modules
- Each BE module has own
  - Model (MongoDB Schema)
  - Router (Express router)
  - Controller (Callbacks for router)
  - Tests (Chai tests)
- Optionally each module has
  - Helper
  - Utils
  - Graphql schema / Resolver / TypeDefs





Step #3 - Modularisation - router

```
app
  .use(cors(corsOptions))
  .use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument))
  .use('/api', ApiRouter)
  .use('/', ViewRouter);
```

```
const express = require( id: 'express');
const router = express.Router();

// CORE routes
const AuthRouter = require( id: "./modules/auth/router");

// CMS routes
const PostRouter = require( id: './modules/post/router');
const PageRouter = require( id: './modules/page/router');
const ConfigRouter = require( id: './modules/config/router

// BUILDER routes
const CdnRouter = require( id: "./modules/cdn/router");
const ElementsRouter = require( id: "./modules/element/router");
const ElementsFreeRouter = require( id: "./modules/element-free/router");
const CompositionRouter = require( id: "./modules/composition/router");

router.use('/auth', AuthRouter);

// CMS routes
router.use('/posts', PostRouter);
router.use('/page', PageRouter);
router.use('/config', ConfigRouter);

// BUILDER routes
router.use('/elements', ElementsRouter);
router.use('/elements-free', ElementsFreeRouter);
router.use('/compositions', CompositionRouter);
router.use('/cdns', CdnRouter);

module.exports = router;
```

```
const express = require( id: 'express');
const router = express.Router();
const elCtrl = require( id: './ElementController');
const AuthMiddleware = require( id: '../auth/AuthMiddleware');
const mids = AuthMiddleware.loginCheck;

router
  .get('/', mids, elCtrl.list)
  .get('/:id', mids, elCtrl.read)
  .post('/', mids, elCtrl.create)
  .put('/:id', mids, elCtrl.update)
  .delete('/:id', mids, elCtrl.delete);

module.exports = router;
```

Lesson #3

- Use linters / hinters - it helps you to find issues earlier
- Write tests - it makes your life easier
- Make smaller modules which are easier to maintain
- Create **‘core’** elements which will be used in your whole app

Issues #3

- Lot of code **(still)** in controllers which are looking almost the same

Step #4

Core helpers
Standardisation of Queries and Responses

Step #4 - Response helper

- Two functions
  - Success
  - Error
- Helps to keep response format same everywhere

```
const ERROR_MSG = {
    message: 'error'
}
const SUCCESS_MSG = {
    message: 'success',
    errors: null
}

const errorResponse = (errors) => {
    return {
        ...ERROR_MSG,
        errors: errors
    }
}

/**
 *
 * @param {array} rows
 * @param {number} limit
 * @param {number} total
 */
const successResponse = (data, limit = null, total = null) => {
    return {
        ...SUCCESS_MSG,
        errors: null,
        data,
        limit,
        total
    }
}

module.exports = {
    successResponse,
    errorResponse
}
```

But why haven't you used Classes? :)

Step #4 - CRUD Helper

- Functions which helps me to control CRUD actions (for project itself and Unit testing)

```
20
21    listCurried: (Model) => async (condition, { sort, fields, limit, offset }) => {
22      let query = Model.find(condition);
23
24      const itemsToCount = await new Promise((resolve, reject) => {
25        Model.count(condition, (err, length) => {
26          if (err) {
27            reject(err);
28          } else {
29            resolve(length);
30          }
31        });
32      });
33
34      (typeof sort !== 'undefined') ? query.sort(sort) : null;
35      (typeof fields === 'string') ? query.select(fields.split(',').join(' ')) : null;
36      (typeof limit !== 'undefined') ? query.limit(limit) : null;
37      (typeof offset !== 'undefined') ? query.skip(offset) : null;
38
39      return new Promise((resolve, reject) => {
40        query.exec((err, items) => {
41          if (err) {
42            reject(err);
43          } else {
44            resolve({ items, total: itemsToCount });
45          }
46        });
47      });
```

```
7   const CompositionController = {
8     list: async function(req, res) {
9       const token = req.headers["x-access-token"];
10
11      try {
12        const user = await UserTokenHelper.getUserIdByToken(token);
13        const condition = ConditionHelper.userCanUse(user.uid);
14        const compositions = await CRUDHelper.list(MODEL, condition);
15
16        res.status(200).json(compositions);
17      } catch (e) {
18        res.status(500).json(e);
19      }
20    },
21
```

nodecms / core / modules / composition / **CompositionController.js**

```
3
4   const CompositionController = {
5     /**
6      * List all elements with all properties
7      * @param req
8      * @param res
9      */
10    listAll(req, res) {
11      CompositionHelper
12        .list()
13        .then((arr) => {
14          res
15            .status(200)
16            .json(arr)
17        })
18        .catch((err) => {
19          res
20            .status(500)
21            .json(err)
22        })
23    },
24
```

nodecms / core / modules / composition / **CompositionHelper.js**

```
1   'use strict';
2
3   const Composition = require('./CompositionModel');
4   const CRUDHelper = require('../../helpers/CRUDHelper');
5
6   const CompositionHelper = {
7     list: function() {
8       return CRUDHelper.list(Composition, {});
9     },
10
11    create: function(obj) {
12      return CRUDHelper.create(Composition, obj);
13    },
14
15    read: function() {
16
17    },
18
19    update: function(id, obj) {
20      return CRUDHelper.update(Composition, id, obj);
21    },
22
```

Step #4 - Query helper

- Helps me to keep standard / consistent approach to queries

```
QueryHelper = {
    parse: (query) => {
        const parsedQueries = {
            limit: parseInt(query['limit']) || 0,
            skip: parseInt(query['skip']) || 0,
            sort: query['sort'] || null,
            fields: query['fields'] || null,
        }

        return parsedQueries;
    }
}

module.exports = QueryHelper;
```

localhost/api/endpoint?limit=20&skip=15&sort=_id&fields=_id,name

Lesson #4

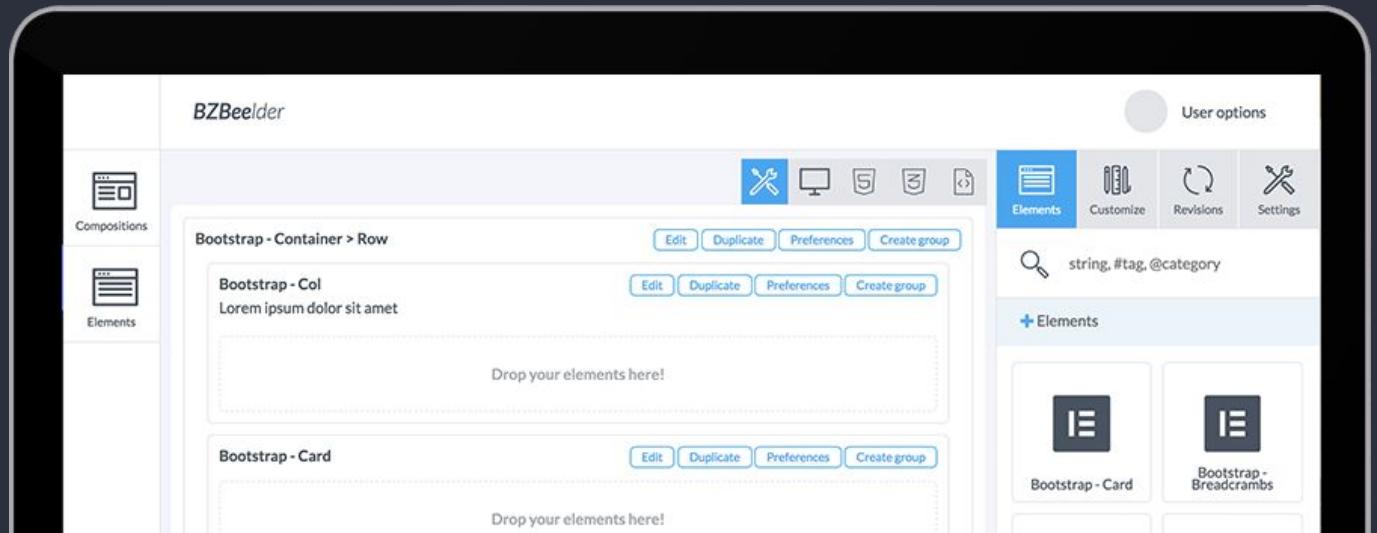
- Create **‘core’** code which will be used in your whole app and make your application consistent

Issues #4

- Tests are done but it's not connected with GIT so who knows what can be in repo
- Tests are done but have I tested everything? (happy paths vs corner cases)
- Lot of code still in controllers which are looking almost the same

Step #5

DRY, GIT



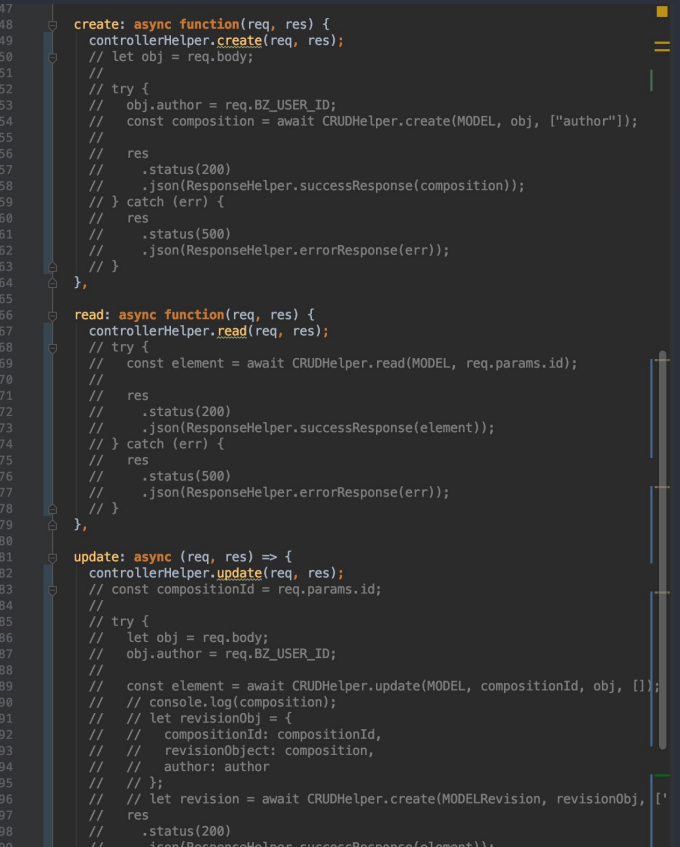
Step #5 - Controller Helper

- Universal controller for each MongoDB Schema
- It's based on
  - CRUD Helper,
  - Response Helper,
  - Query Helper

  mentioned before

```
const CRUDHelper = require( id: './CRUDHelper');
const ResponseHelper = require( id: './ResponseHelper');
const QueryHelper = require( id: './QueryHelper');

class CtrlHelper {
  constructor(model, config) {
    this.model = model;
    this.LIST_VALUES = config.LIST_VALUES || [];
    this.LIST_CONDITION = config.LIST_CONDITION || {};
    this.CREATE_FOREIGN_KEYS = config.CREATE_FOREIGN_KEYS || [];
  }

  async list({ query }, res) {
    const config = QueryHelper.parse(query);
    const condition = {};

    try {
      const { items, total } = await CRUDHelper.listCurried(this.model)(condition, config);

      res
        .status(200)
        .json(ResponseHelper.successResponse(items, config.limit, total));
    } catch (err) {
      res
        .status(500)
        .json(ResponseHelper.errorResponse(err));
    }
  }

  async create({ body }, res) {
    try {
      const element = await CRUDHelper.create(this.model, body, this.CREATE_FOREIGN_KEYS);
      res
        .status(200)
        .json(ResponseHelper.successResponse(element));
    } catch (err) {
      res
        .status(500)
        .json(ResponseHelper.errorResponse(err));
    }
  }
```

Step #5 - Controller Helper



```
  create: async function(req, res) {
    controllerHelper.create(req, res);
    // let obj = req.body;
    //
    // try {
    //   obj.author = req.BZ_USER_ID;
    //   const composition = await CRUDHelper.create(MODEL, obj, ["author"]);
    //
    //   res
    //     .status(200)
    //     .json(ResponseHelper.successResponse(composition));
    // } catch (err) {
    //   res
    //     .status(500)
    //     .json(ResponseHelper.errorResponse(err));
    // }
  },

  read: async function(req, res) {
    controllerHelper.read(req, res);
    // try {
    //   const element = await CRUDHelper.read(MODEL, req.params.id);
    //
    //   res
    //     .status(200)
    //     .json(ResponseHelper.successResponse(element));
    // } catch (err) {
    //   res
    //     .status(500)
    //     .json(ResponseHelper.errorResponse(err));
    // }
  },

  update: async (req, res) => {
    controllerHelper.update(req, res);
    // const compositionId = req.params.id;
    //
    // try {
    //   let obj = req.body;
    //   obj.author = req.BZ_USER_ID;
    //
    //   const element = await CRUDHelper.update(MODEL, compositionId, obj, []);
    //   // console.log(composition);
    //   // let revisionObj = {
    //   //   compositionId: compositionId,
    //   //   revisionObject: composition,
    //   //   author: author
    //   // };
    //   // let revision = await CRUDHelper.create(MODELRevision, revisionObj, ['
    //   res
    //     .status(200)
    //     .json(ResponseHelper.successResponse(element));
```

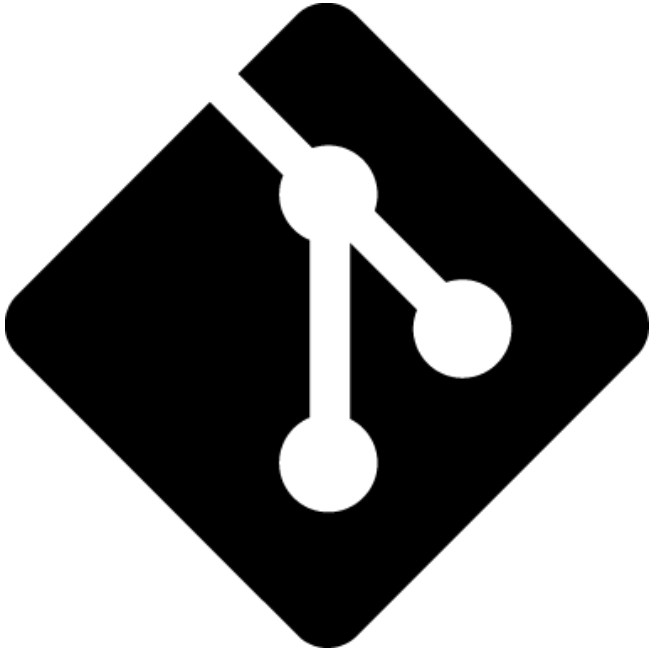
Controller Helper

1. CRUD operations for standard types
2. Keeps Query and Response standard
3. CRUD controller standart

Step #5 - Git

- Use GIT hooks
- It helps you make actions like linting / hinting / testing right before commiting and pushing code to repository
- Husky

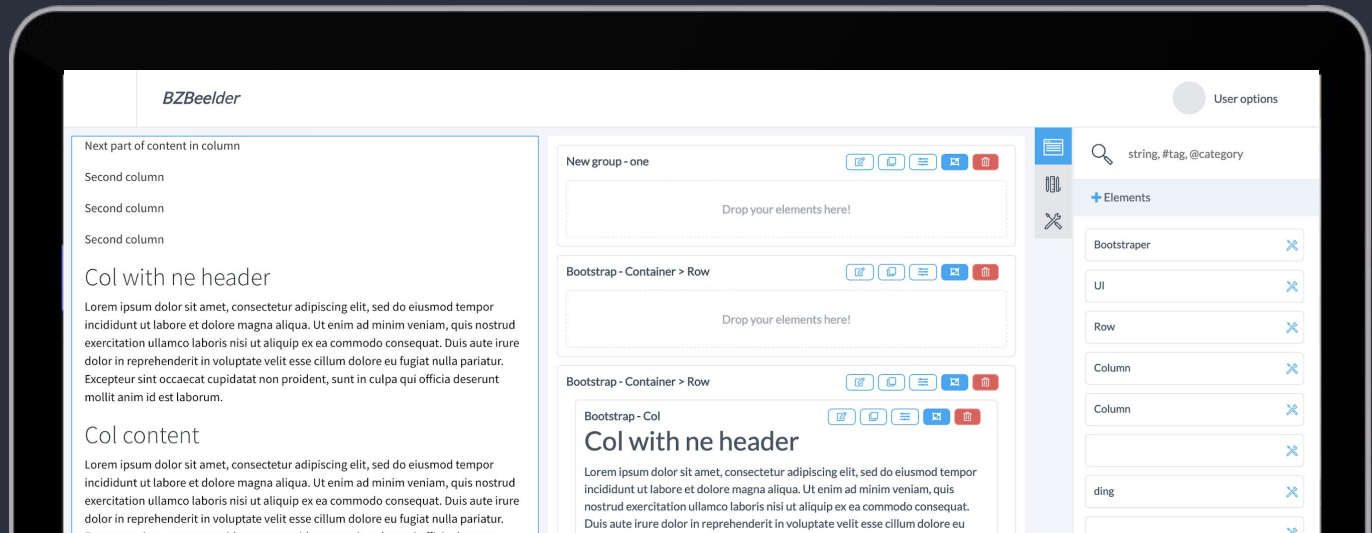
https://github.com/typicode/husky



Lesson #5

- DRY - watch your code and try to identify repeatable code
- Use GIT features (hooks)

Step #6
Templating



Step #6 - Template engine

- I've chosen PUG as a template engine
- PUG? WHY?

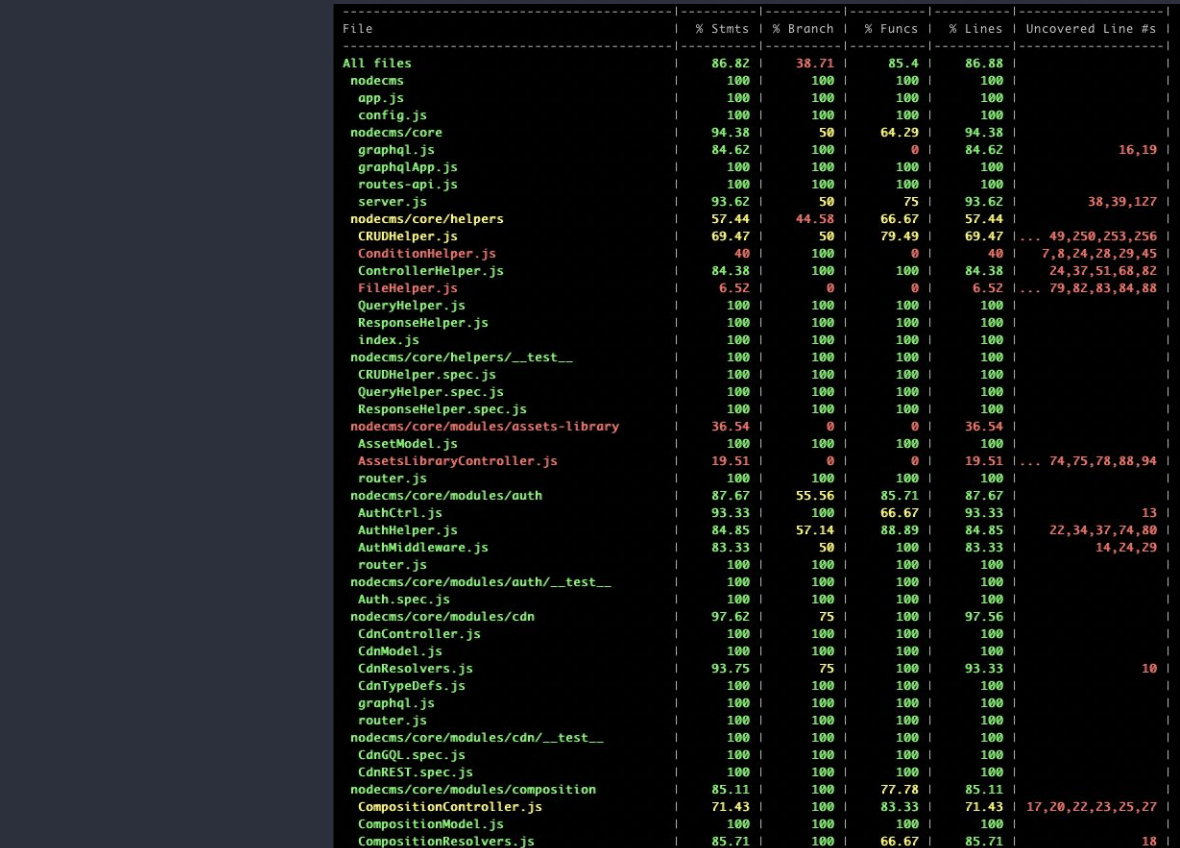
Step #6 - Template functions

- It helps to create template based on predefined functions
- Similar to WP functions like get_post(), get_posts(), get_header()

```
.youmightlike
  .youmightlike__content
    h2 Recent posts
    ul
      each post in relatedPosts
        li.youmightlike__listelement
          div.youmightlike__desccontainer
            span.youmightlike__date= dateToStandardFormat(post.create_date)
            a(href="/"+post.slug)
              span.h3= post.title
```

Step #6 - Code coverage

- Istanbul.js

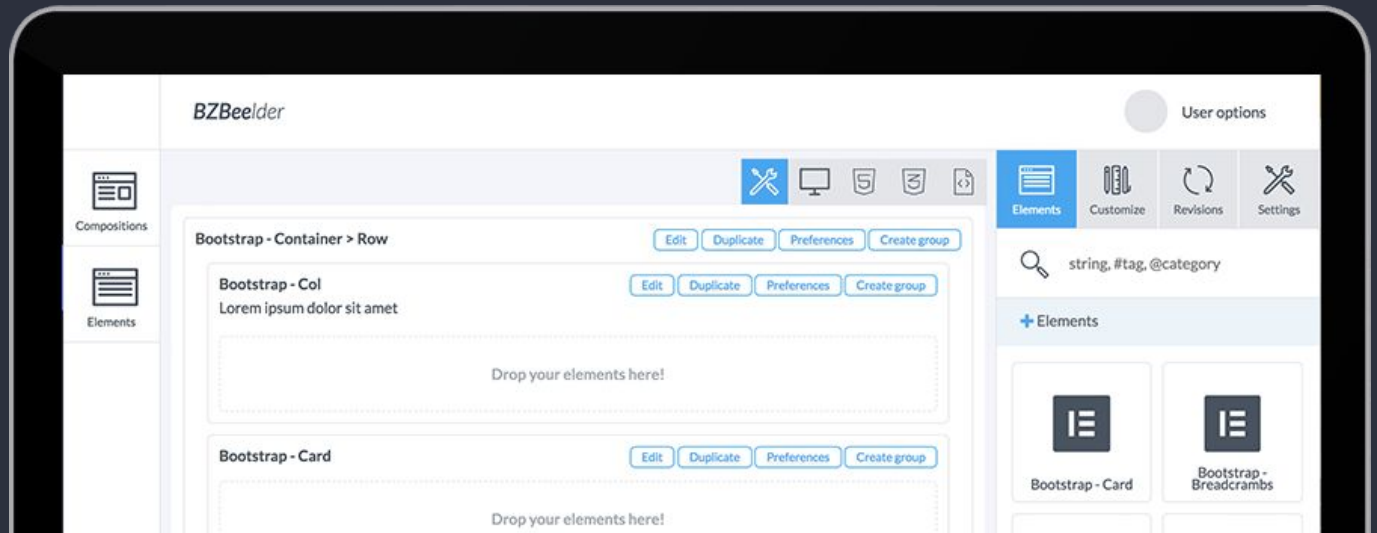


File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	86.82	38.71	85.4	86.88	
nodecms	100	100	100	100	
app.js	100	100	100	100	
config.js	100	100	100	100	
nodecms/core	94.38	50	64.29	94.38	
graphql.js	84.62	100	0	84.62	16,19
graphqlApp.js	100	100	100	100	
routes-api.js	100	100	100	100	
server.js	93.62	50	75	93.62	38,39,127
nodecms/core/helpers	57.44	44.58	66.67	57.44	
CRUDHelper.js	69.47	50	79.49	69.47	... 49,250,253,256
ConditionHelper.js	40	100	0	40	7,8,24,28,29,45
ControllerHelper.js	84.38	100	100	84.38	24,37,51,68,82
FileHelper.js	6.52	0	0	6.52	... 79,82,83,84,88
QueryHelper.js	100	100	100	100	
ResponseHelper.js	100	100	100	100	
index.js	100	100	100	100	
nodecms/core/helpers/__test__	100	100	100	100	
CRUDHelper.spec.js	100	100	100	100	
QueryHelper.spec.js	100	100	100	100	
ResponseHelper.spec.js	100	100	100	100	
nodecms/core/modules/assets-library	36.54	0	0	36.54	
AssetModel.js	100	100	100	100	
AssetsLibraryController.js	19.51	0	0	19.51	... 74,75,78,88,94
router.js	100	100	100	100	
nodecms/core/modules/auth	87.67	55.56	85.71	87.67	
AuthCtrl.js	93.33	100	66.67	93.33	13
AuthHelper.js	84.85	57.14	88.89	84.85	22,34,37,74,80
AuthMiddleware.js	83.33	50	100	83.33	14,24,29
router.js	100	100	100	100	
nodecms/core/modules/auth/__test__	100	100	100	100	
Auth.spec.js	100	100	100	100	
nodecms/core/modules/cdn	97.62	75	100	97.56	
CdnController.js	100	100	100	100	
CdnModel.js	100	100	100	100	
CdnResolvers.js	93.75	75	100	93.33	10
CdnTypeDefs.js	100	100	100	100	
graphql.js	100	100	100	100	
router.js	100	100	100	100	
nodecms/core/modules/cdn/__test__	100	100	100	100	
CdnGQL.spec.js	100	100	100	100	
CdnREST.spec.js	100	100	100	100	
nodecms/core/modules/composition	85.11	100	77.78	85.11	
CompositionController.js	71.43	100	83.33	71.43	17,20,22,23,25,27
CompositionModel.js	100	100	100	100	
CompositionResolvers.js	85.71	100	66.67	85.71	18

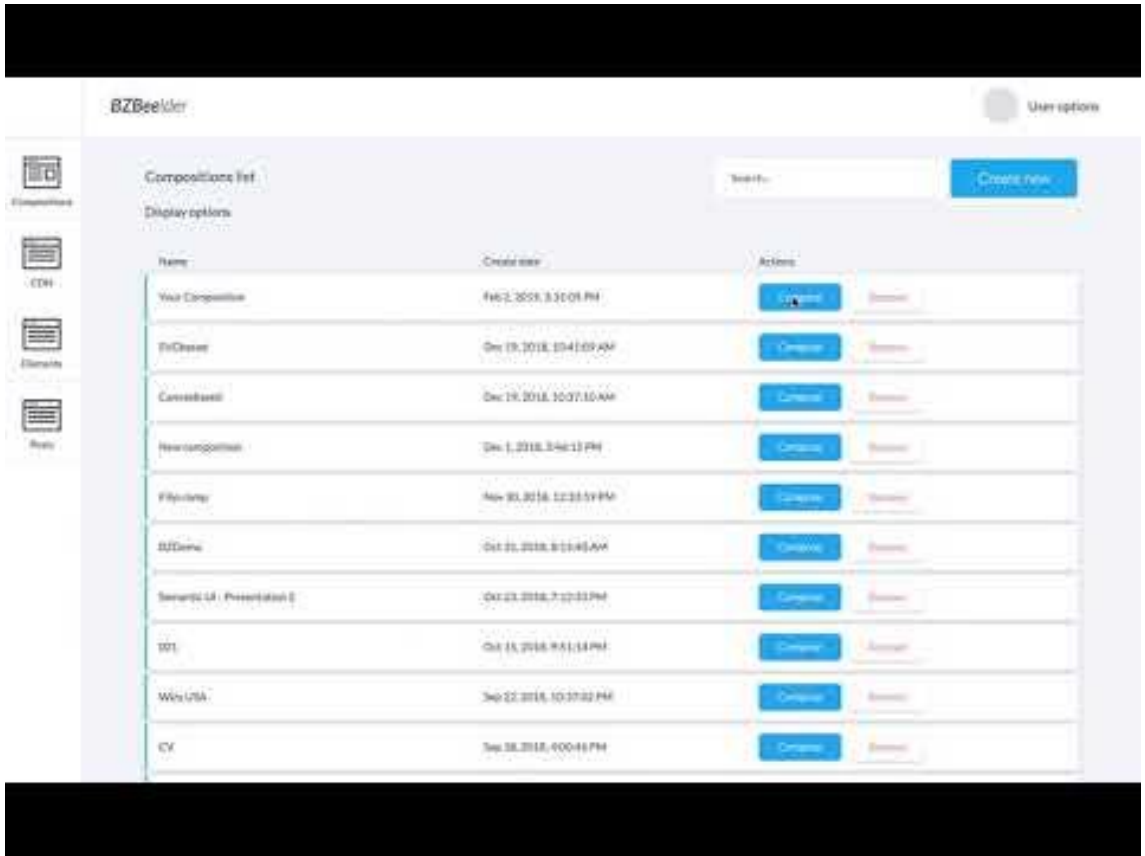
Lesson #6

- Template engine makes your life easier not only in CMS engine
- Extend template this language with your functions
- Caching!

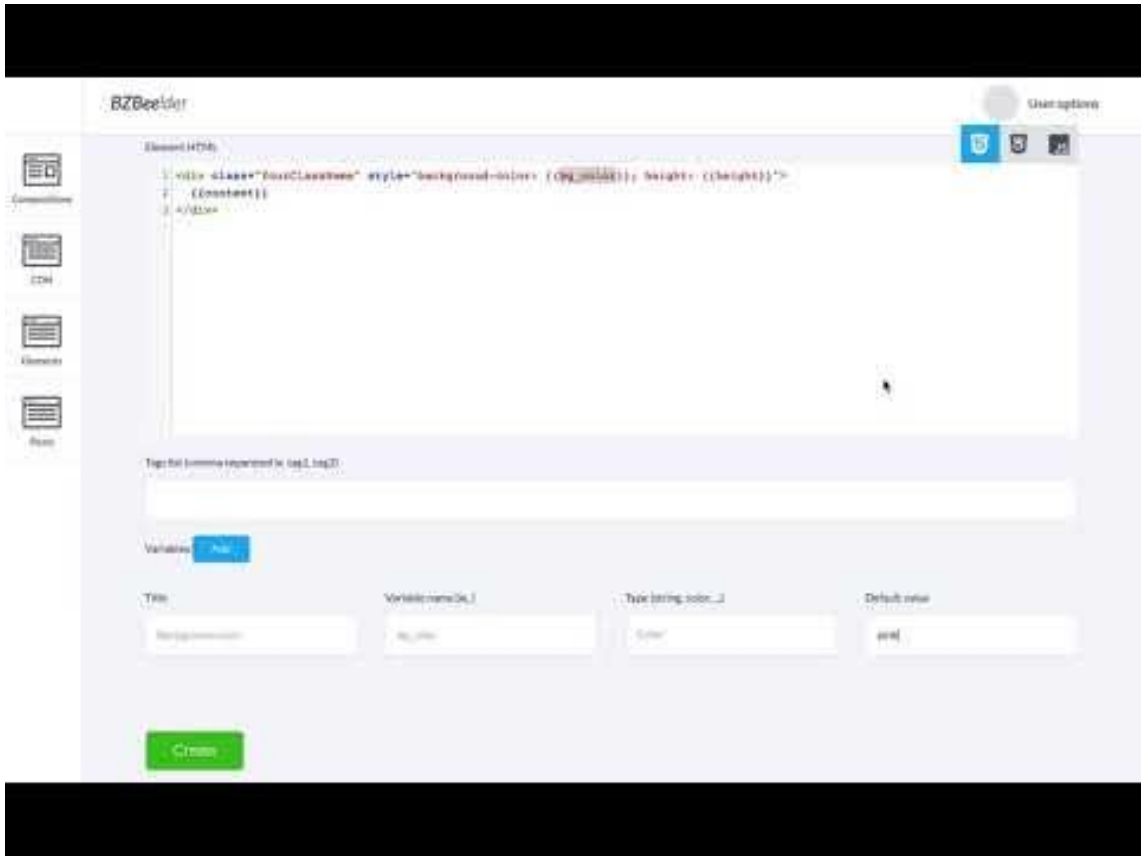
How it works!


BZBeelder
User options
Compositions
Elements
Bootstrap - Container > Row
Edit
Duplicate
Preferences
Create group
Bootstrap - Col
Lorem ipsum dolor sit amet
Edit
Duplicate
Preferences
Create group
Drop your elements here!
Bootstrap - Card
Edit
Duplicate
Preferences
Create group
Drop your elements here!
Elements
Customize
Revisions
Settings
string, #tag, @category
+ Elements
Bootstrap - Card
Bootstrap - Breadcrambs

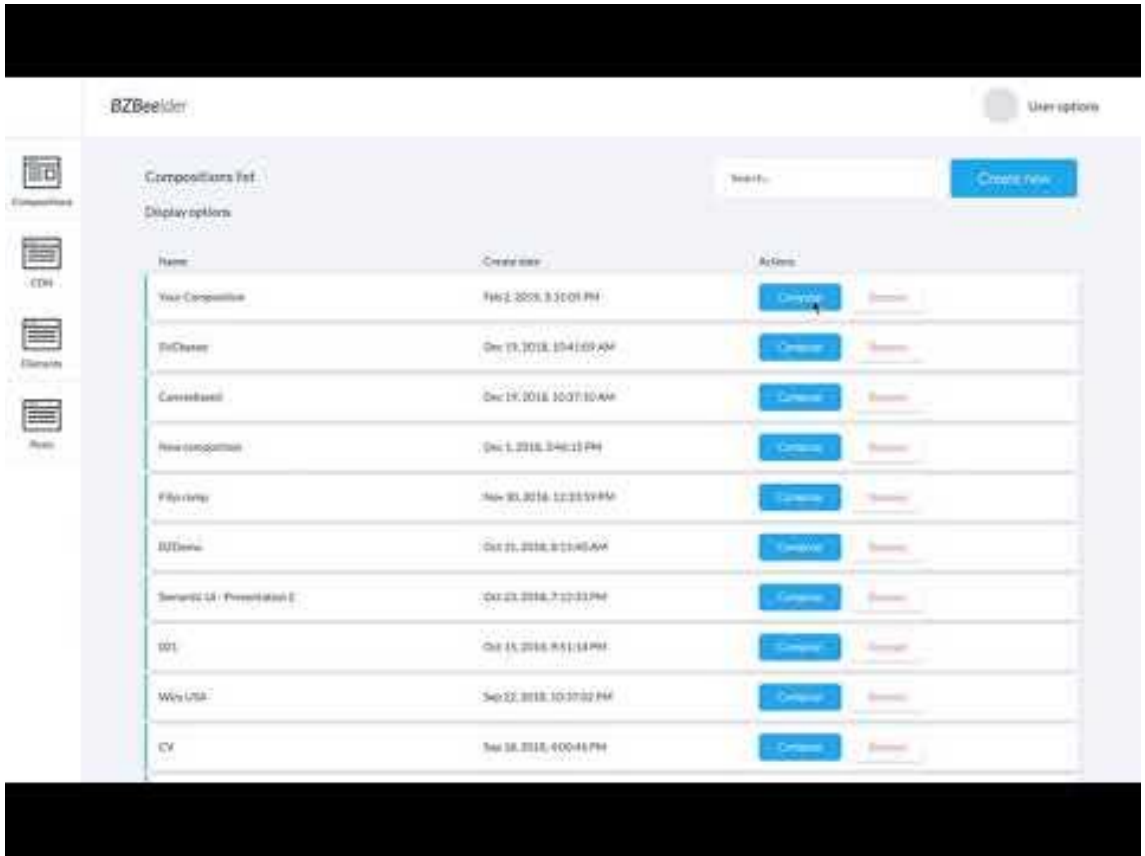
Front End - Define element



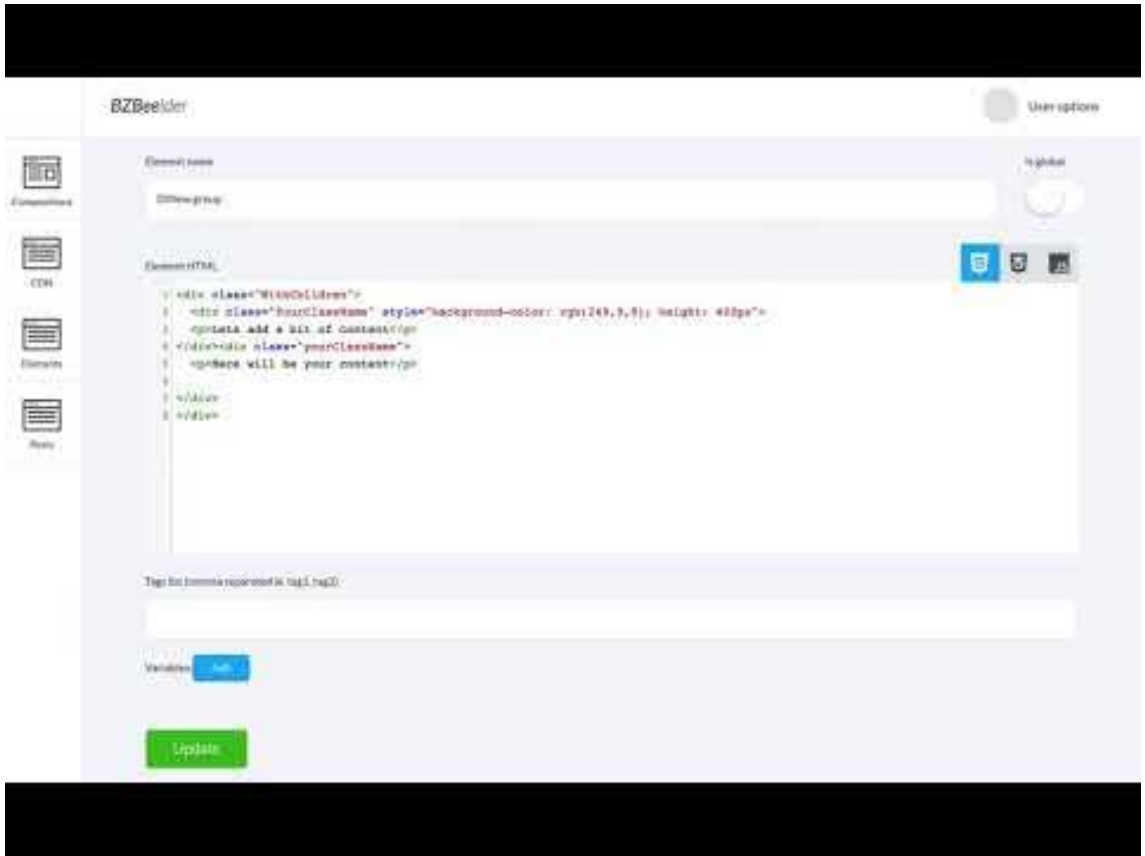
Front End - Define element with variables



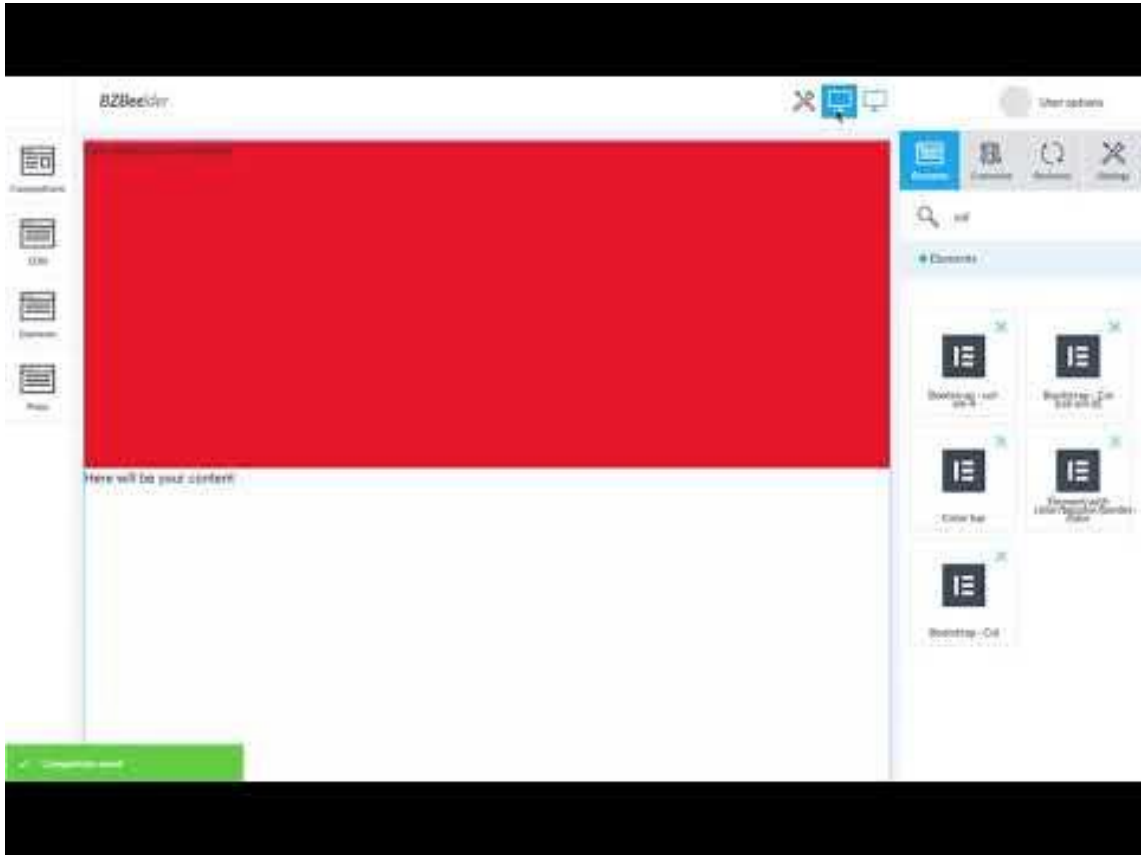
Front End - Define element with variables



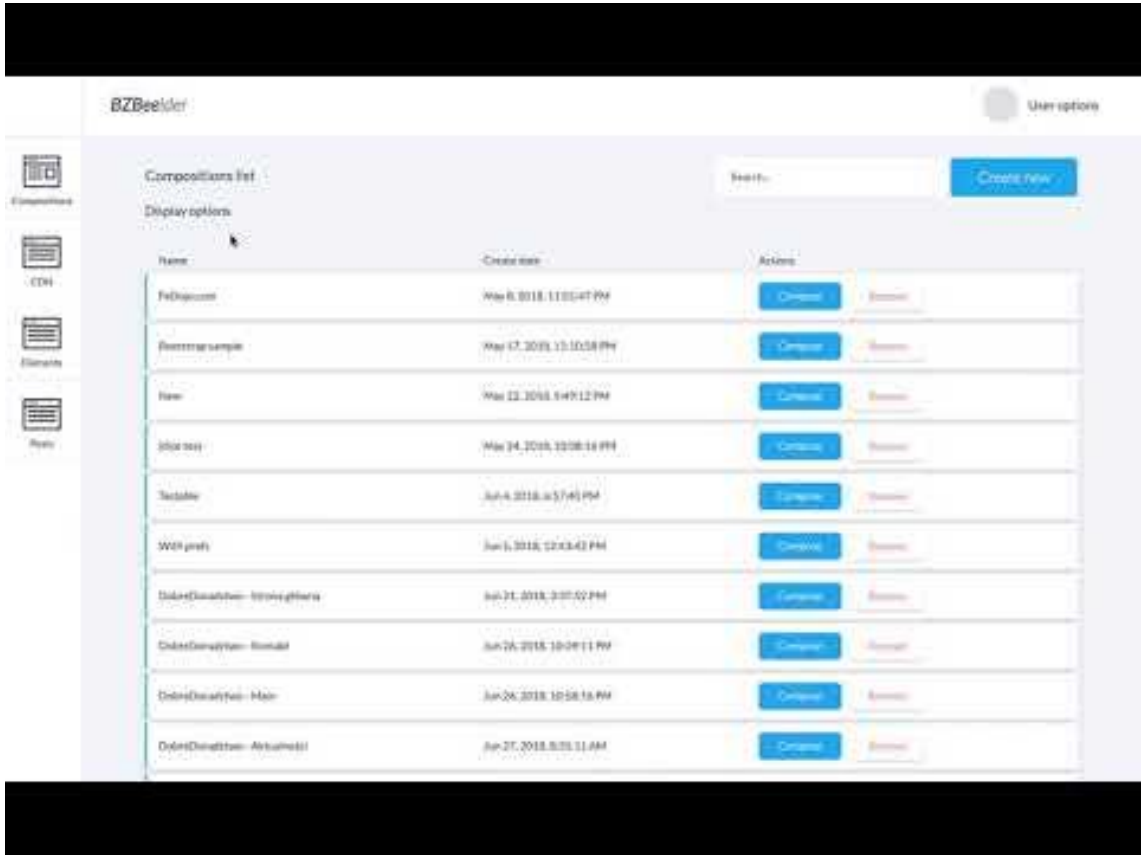
Front End - Define group of elements



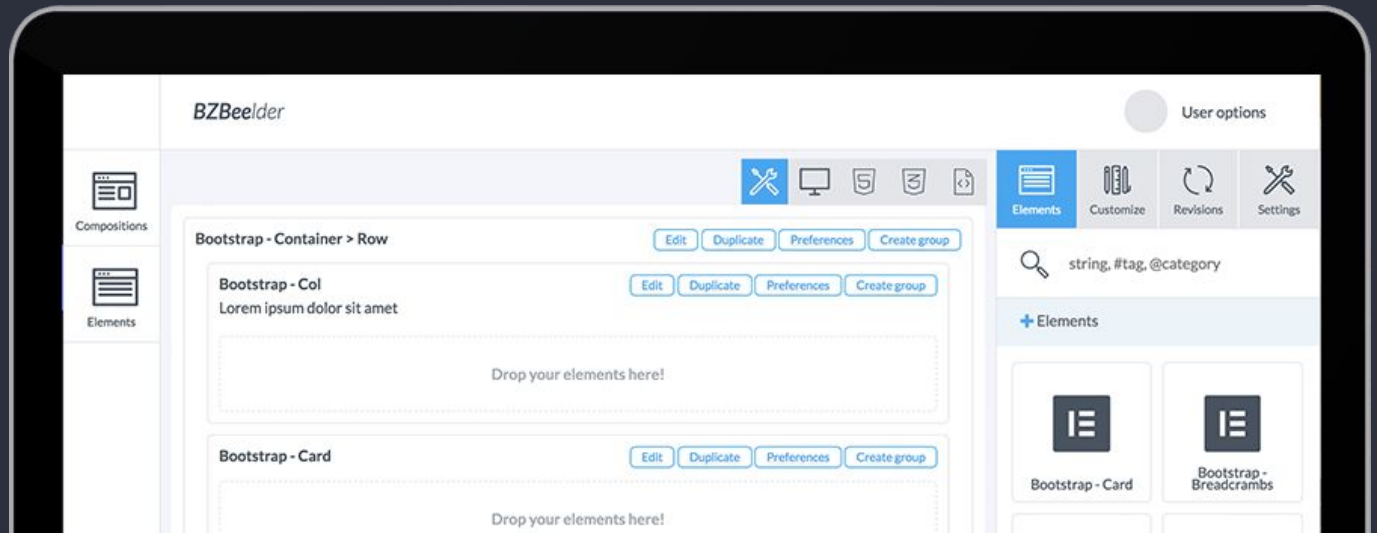
Front End - Using CDNs



Front End - Data grid

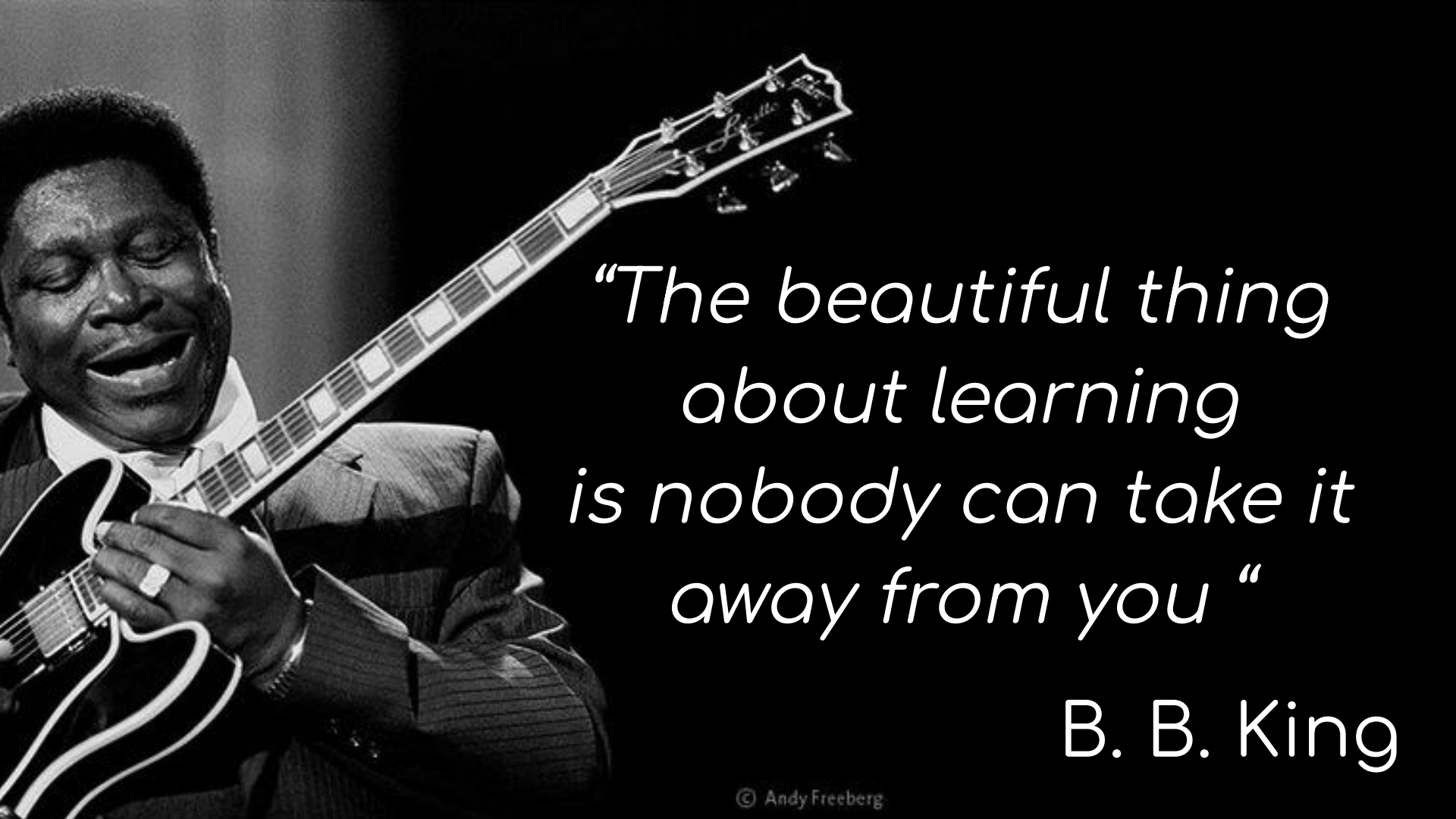


Why it's good to have own side project



Why it's good to have side project

- You can still test new libs and fresh language features (especially when it's such dynamic as **JavaScript**)
- Your visibility is rising and you can gather feedback about your code (even if it is bad feedback it's better to have it earlier)
- If you have a bit of time you can sit to it and add more features
- It's fully yours!


“The beautiful thing
about learning
is nobody can take it
away from you “
B. B. King
© Andy Freeberg

"The best time to plant a tree was 20 years ago. The second best time is now."

Chinese Proverb

Questions time!

Thank you!

Blog: http://fedojo.com
GitHub: https://github.com/fedojo
Project: https://github.com/fedojo/bzbeelder-be