Kaang: A RESTful API Generator for the Modern Web

Ricardo Queirós
CRACS & INESC-Porto LA & DI/ESMAD/P.PORTO, Porto, Portugal
ricardoqueiros@esmad.ipp.pt
https://orcid.org/0000-0002-1985-6285

Abstract

Technology is constantly evolving, as a result, users have become more demanding and the applications more complex. In the realm of Web development, JavaScript is growing in a surprising way, already leaving the boundaries of the browser, mainly due to the advent of Node.js. In fact, JavaScript is constantly being reinvented and, from the ES2015 version, began to include the OO concepts typically found in other programming languages.

With Web access being mostly made by mobile devices, developers face now performance challenges and need to perform a plethora of tasks that weren't necessary a decade ago, such as managing dependencies, bundling files, minifying code, optimizing images and others. Many of these tasks can be achieved by using the right tools for the job. However, developers not only have to know those tools, but they also must know how to access and operate them. This process can be tedious, confusing, time-consuming and error-prone.

In this paper, we present Kaang, an automatic generator of RESTFul Web applications. The ultimate goal of Kaang is to minimize the impact of creating a RESTFul service by automating all its workflow (e.g., files structuring, boilerplate code generation, dependencies management, and task building). This kind of generators will benefit two types of users: will help novice developers to decrease their learning curve while facing the new frameworks and libraries commonly found in the modern Web and speed up the work of expert developers avoiding all the repetitive and bureaucratic work. At the same time, Kaang promotes the good development principles by adding automatic testing and documentation generation.

For this accomplishment, Kaang generates the main API content based on the user's input and a set of templates which will help developers to manage and test routes, define resources, store data models and others. In order to provide an addition level of confidence to the generator's end-users, the generator will be integrated on Travis CI and published on both the npmjs and Yeoman registries.

2012 ACM Subject Classification Software and its engineering → Source code generation

Keywords and phrases web development, generators, web tooling, javascript

Digital Object Identifier 10.4230/OASIcs.SLATE.2018.1

Funding FourEyes is a Research Line within project "TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

1 Introduction

Nowadays we are witnessing a remarkable revolution in the Web frontend development. A decade ago we just needed to master the 3 pillars of the Web: the HyperText Markup Language (HTML) for structuring content, the Cascading Style Sheets (CSS) to style it and the JavaScript language to attach some behavior. With the evolution of the Web, users became more demanding and the applications became more complex. At the same time, browsers became more powerful and, consequently, the growing access to the Web through mobile devices increased. This made the frontend work more bureaucratic from the ad hoc management of few HTML/CSS/JS files to the automation of complex workflows. These new workflows are characterized by several tools and libraries organized into several categories which can be boil down to three main categories: scaffolding tools, dependency managers and build tools [3].

While all these tools help developers in their day-to-day lives, they also create a big learning curve for those that are starting in the Web development realm. Even for experts, this kind of tools and repetitive tasks become time-consuming, boring and their mechanical nature can produce undesirable errors either working alone or within a team. A typical scenario is the creation of RESTful APIs. In this context, we need to gather several libraries and frameworks to handle route management, resource definition, data persistence and unit tests. Also, we need to create the client-side, typically composed by a responsive and friendly GUI based on forms to easily communicate with the API endpoints and feed the respective models. As you can imagine, this process is time-consuming and error prone.

This paper presents **Kaang** as a RESTful API generator. In order to use the generator, the developer only has to open the command line and invoke the generator. During the application generation process, the API is customized based on the developers' input data. At the end, a tested and documented RESTful API is generated and can be consumed by a responsive GUI or using your favorite API tester (e.g., Postman). Obviously, the programmer may have to do some refinement, namely adding routes or new features. But the important thing is that the entire software development workflow had been set up and all the initial hard work mitigated, freeing the developer for more important coding aspects of the application.

The remainder of this paper is organized as follows: Section 2 reviews the current tools that nowadays gravitate in the Web development workflow and enumerates and compares the main existing generators based on three criteria: maturity, coverage and performance. In Section 3, we present Kaang and describe its architecture and main components, the input system, the routes management and the generation of tests and documentation. In Section 4, we validate Kaang enumerating the steps for the generation of a Movies API. Finally, we conclude with a summary of our main contributions and a perspective of future research.

2 Web development workflow

The Web development is going through an unprecedented phase of profound changes. Nowadays, a frontend Web developer should not only master the HTML/CSS/JavaScript triad, but also have a (basic) knowledge of what are preprocessors, module bundlers, task runners, scaffolding tools and other automation tools. In fact, these tools are mandatory in the so-called "new web development workflows" mitigating developers work and saving their time for others tasks.

Based on a typical software development workflow, you can easily translate it for the Web realm identifying three phases developers go through when coding (Figure 1).

The **setup phase** is the starting point where developers commonly set up project structure, apply reusable patterns/boilerplate code and install third-party libraries.

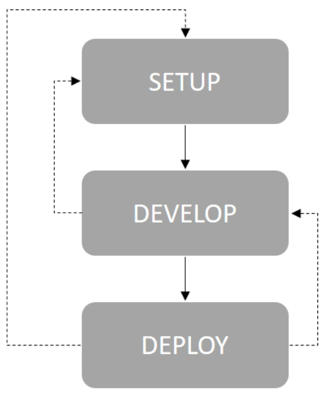


Figure 1 Phases of the Web development workflow.

The **development phase** is where developers write code using a programming language. If necessary, developers can go back to the setup phase, if they need to add a new module or refactor some code. Also here is the right place to perform quality control based on tests.

The **deployment phase** is where developers create an executable bundle from the code written in the previous phase and deploy it. In the Web context, this boils down to deploy the HTML/JavaScript/CSS to an web server. Obviously, developers can return to the development phase (fixing bugs or adding new features to existing code) or to the setup phase (creating new modules).

In order to meet all those expectations proper tooling is needed. In the next two sections Web tooling is depicted. In Section 2.1 the Web tooling is organized into three categories: scaffolding tools, dependency managers and build tools. In Section 2.2, we take a closer look for scaffolding tools and we compare those that generate RESTfull Web apps based on predefined criteria.

2.1 Frontend tooling

Nowadays, a typical Web development workflow [1] comprises three types of tooling which can be organized as follows:

- **Scaffolding tools**: generate the Web project structure, inject boilerplate code and add new modules;
- **Dependency managers**: take care of the self-contained modules, or packages and potential conflicts between them (e.g., dependencies versions) avoiding redundancy;
- **Build tools**: perform all the file-processing tasks, transforming your source code into something deployable. Typical examples are bundle and minify JavaScript and CSS files (e.g., MinifyCSS, UglifyJS), remove dead code (e.g., PurifyCSS), lint code (e.g., ESlint), optimize images (e.g., ImageMagick, ImageOptim), preprocess source code (e.g., CoffeeScript, LiveScript, Less, Sass, PostCSS), run tests (e.g., Jasmine, Mocha, Jest) and many others [2]. The configuration of those tools is stored in build files.

As depicted in Figure 2, the modern Web developer, in a typical scenario, developers start a scaffolding tool (e.g., Brunch, Yeoman) to set up the project, search and install components with a dependency manager (e.g., Bower, WebPack), and process files periodically through a build tool (e.g., Grunt, Gulp) [4]. The first two types of tools work mainly in the initialization phase, while the last one gravitates in the development and deployment phases.

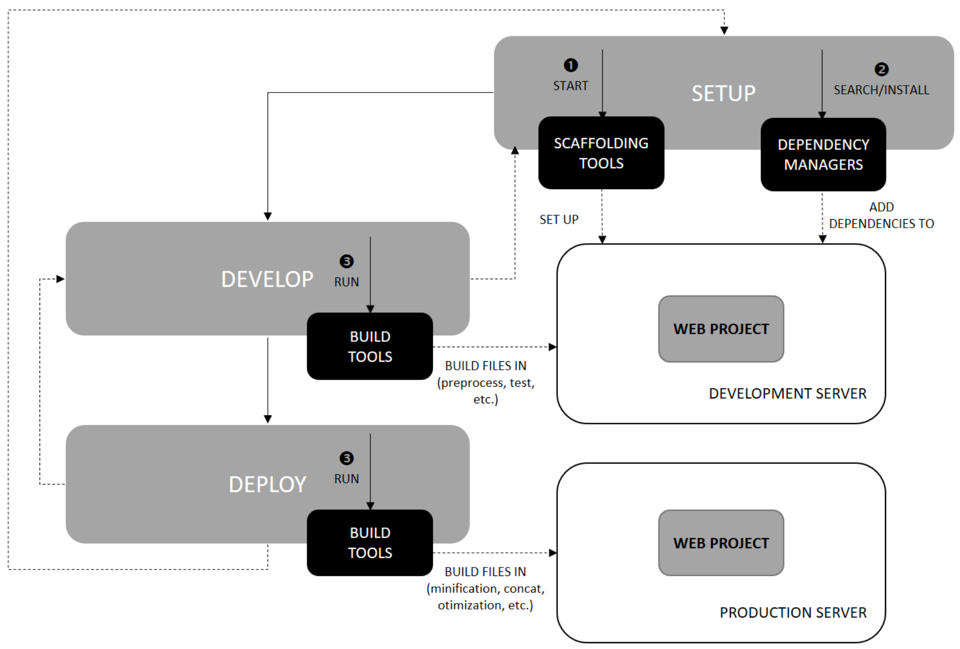


Figure 2 Web tooling.

Thus, the idea is straightforward: instead of using a huge number of tools individually for all the development tasks, developers should invest on automating the execution of tasks through these three different tool types. With these three easy-to-use interfaces all the complexity is hidden and developers can thereafter concentrate their efforts in the business logic.

2.2 REST API generators

One of the most important type of tools in the modern Web workflow is the scaffolding tools. As said before, these type of tools help developers to quickly build web applications by creating the necessary folders, copying initial files (like build scripts), applying boilerplate code, and triggering the installation of dependencies. Throughout development, these tools are also responsible for the creation of the base structure of new modules inside the Web project.

In this context, one of the most dominating tool is **Yeoman**, powered by Google. Despite, the earlier success of Loom and Brunch tools, Yeoman is currently the best way to kick-start new projects, prescribing best practices and tools. The success of yeoman is mainly due to its generator ecosystem. A generator can be defined as a plugin that can be run with the 'yo' command to scaffold complete projects. Currently, Yeoman supports more than 7000 plugins from basic web apps to complex generators for the popular frameworks of Angular, React, Polymer and others.

A generator comes with a folder full of templates that have to be transferred and updated by the generator script based on a simple prompting API that allows different parameters to change the output generated by the generator. Yeoman's only task is to run the generator.

Table 1 Generators maturity.

GitHub data	generator-rest	generator-sails-rest-api	generator-api
First release date	Sep/2016	Jan/2015	Sep/2016
Last release date	0.12.0 (Mar/2018)	1.3.13 (Dec/2017)	1.5.3 (Mar/2018)
# Open issues	11	12	3
# Pull requests	1	7	2
# Commits	130	1486	151
# Releases	22	63	9
# Contributors	13	10	11
# Stars	602	325	188
# Forks	117	63	33

In this paper we take a closer look for RESTful API generators. The selection of the generators was very simple. We access the yeoman generators repository[1] and search the generators using the keyword "rest". Then we sorted the results by popularity (stars of GitHub) and selected the top three, namely: generator-rest[2], generator-sails-rest-api[3] and generator-api[4].

In the next subsections the three generators are described and compared based on three facets: maturity, coverage and performance.

2.2.1 Maturity

It is difficult to determine which generator is most widely used or have more impact. Various methods of measuring tools popularity have been proposed, each subject to a different bias over what is measured. In this context, we will focus on comparing the activity in GitHub where all the selected generators are stored. In fact, the three generators chosen are pretty active on GitHub, as you can see in Table 1.

Although relatively recent, the generator-rest is the most popular, with a larger number of stars and forks. The number of forks is relevant. A fork is a copy of a repository. Forking a repository allows you to freely experiment a repo (with changes) without affecting the original project. Thus, this means that most people are using the Yeoman generator-rest base code to start their own projects. Regarding the generator-api, it presents the lower values of the three in almost all the indicators.

2.2.2 Coverage

For the coverage criterion we will make a comparison of the three Yeoman generators regarding the support for tasks typically found in build tools and REST specific features.

In Table 2 we present the comparison on generators build tools features.

As you can conclude, we have disappointing results, but there is an obvious reason. Web RESTful applications depend mostly in thin Web clients while most of the hard work is

[1] `http://yeoman.io/generators`
[2] `https://github.com/diegohaz/rest`
[3] `https://github.com/ghaiklor/generator-sails-rest-api`
[4] `https://github.com/ndelvalle/generator-api`

Table 2 Build tools features comparison.

Features	generator-rest	generator-sails-rest-api	generator-api
Bundler	-	-	-
Linter	ESLint	-	ESLint
Minifier	-	-	-
Optimizer	-	-	-
PreProcessor	-	Sass/Coffee	-
Reloader	-	-	-
Tester	jest	mocha	mocha

Table 3 REST features comparison.

Features	generator-rest	generator-sails-rest-api	generator-api
CRUD	yes	yes	yes
User login	yes	yes	no
Pagination	yes	no	no
Tester	yes	yes	yes
Documenter	yes	no	no
New models	yes	no	yes
Predefined services	no	yes	no
MVC	no	yes	no

server-side. Thus, you do not need to handle large and complex HTML/CSS files on the client. For that reason, is not crucial the presence of most of the tools included in this table.

In Table 3 we present the comparison on generators REST features.

One can conclude that all the generators have the basic ability to support CRUD functions in one (or more) endpoints. Also all the generators have an authentication layer based on JWT, Facebook, Twitter, GitHub, Instagram, Google Plus and other social networks. The pagination feature is only supported by the generator-rest through querymen, a Querystring parser middleware for MongoDB, Express and Nodejs (MEN). All the generators support tests. The generator-rest uses jest[5] and the others two use mocha[6]. Regarding documentation, only the generator-rest supports the creation of documentation from API annotations in the source code. The generation of documentation is based on apiDoc[7] – an inline documenter for RESTful web APIs. Also, all the generators foster the creation of new models and endpoints. However, the generator-api provides a quasi-automatic approach based on the concept of subgenerators. Once you have the generated project, if you want to add a new model you can simply run yo api:model. This will generate a new folder under model, and, then, you just need to import the route. Finally, the generator-sails-rest-api is the only generator that comprises several predefined services (cipher, image, payment, sms, etc.) and implements MVC through Sails.js[8] – a realtime MVC framework for Node.js.

[5] `https://facebook.github.io/jest/`
[6] `https://mochajs.org/`
[7] `http://apidocjs.com/`
[8] `https://sailsjs.com/`

Table 4 Performance benchmark.

Generators	generator-rest	generator-sails-rest-api	generator-api
Installation (npm)	10.73s	10.85s	10.91s
Generation (yo)	25.33s	4.31s	8.42s

2.2.3 Performance

In this subsection, the three generators are compared in terms of performance. This performance benchmark will be achieved by measuring the installation time of the generators through npm and the generation time of new projects through yeoman.

Obviously, these times are not very important in the production phase, but they are crucial in the development phase, since they allow to measure the impact in terms of time during the development process of the generator.

For the experiment, we started by installing Node.js v8.11.1 (includes npm 5.8.0) in a machine running Windows 10 (64 bits), Intel Core i7-6700K at 4.00GHz, 16 GB RAM and SSD. The results are presented in Table 4.

The first line of the table reflects the time spent by npm to install the generator. The npm tool is a package manager for JavaScript and is the world's largest software registry.

The npm tool is distributed with Node.js – which means that when you download Node.js, you automatically get npm installed on your computer. The command used for install the generators was the following:

```
npm install %%GENERATOR_NAME%% -g
```

The second line reflects the time spent by Yeoman to generate a new Web project based on the respective generator. For these you should firstly install yeoman, then create a folder for the new project and, finally, you can use yo to generate your project.

```
npm install -g yo
mkdir %%PROJECT_FOLDER%%
cd %%PROJECT_FOLDER%%
yo %%GENERATOR_NAME%%
```

Based on these results, the main conclusion is that all the generators take almost the same time to be installed through npm. However, when we look for the projects generation time, we see that generator-rest takes too much time when compared with the remainder generators. The reason for this discrepancy is due to the high number of libraries and tools provided by this generator.

3 Kaang

In this section we present Kaang, a RESTful API generator. The ultimate goal of Kaang is to help developers to quickly create basic REST API applications using the modern Web tooling. Kaang is based on Yeoman.

Yeoman is an open source client-side development stack, consisting of tools and frameworks intended to help developers build web applications. The most important part of Yeoman is the concept of generators. Yeoman generators are, at their core, Node.js modules and can be defined as building blocks on the Yeoman ecosystem. For the creation of Kaang, we didn't create it from scratch, instead we use a Yeoman generator (called generator-generator) to

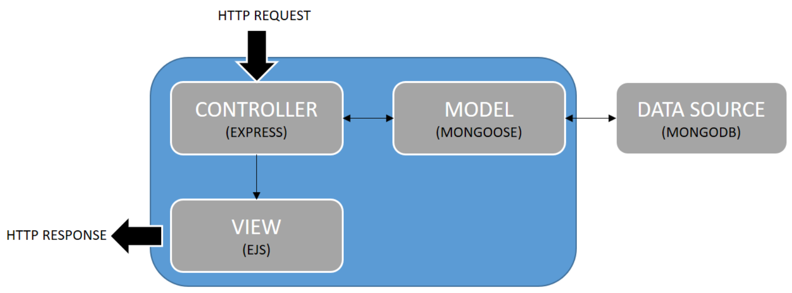


Figure 3 Kaang architecture.

automatically create a generator skeleton. Based on this skeleton we applied several changes which are depicted in the next subsections.

3.1 Architecture

Kaang's generated application architecture is based on a typical server-side API, running in a Node.js server and implementing MVC as depicted in Figure 3.

Kaang is composed by the following four main components:

1. Web framework
2. Database
3. Object document Mapper
4. Template engine

Web framework. For the Web framework we selected Express.js[9] as the REST framework which includes basic routing to determine how the generated application will respond to a client request to a particular endpoint. Each route can have one or more handler functions, which are executed when the route is matched.

Database. The database chosen was MongoDB[10]. Currently, MongoDB is the most popular NoSQL database since it integrates seamlessly in the Node.js realm.

Object document mapper. In order to map the model with the database, we selected a mediator responsible to define objects with a strongly-typed schema which is mapped to a MongoDB document. In this case, the selection was easy due to the popularity of Mongoose[11], an Object Document Mapper (ODM) for MongoDB documents.

Template engine. A template engine enables the use of static template files in a Web application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page. Express uses Pug (by default), but we can use others such as Mustache or EJS. For the Kaang generator, we opted for EJS[12] as the official template engine due to its simplicity and increasing popularity.

[9] `https://expressjs.com/`
[10] `https://www.mongodb.com/`
[11] `http://mongoosejs.com/`
[12] `http://ejs.co/`

3.2 Structure and Input

By default, Kaang presents a basic structure influenced by the structure generated by the express-generator tool, responsible to quickly create an application skeleton in Express. The following list shows the main folders and files generated by Kaang:

```
-- config/
   -- bower/
   -- gulp/
-- public/
   -- images/
   -- js/
   -- css/
   -- libs/
      -- bootstrap
      -- jquery
-- routes/
   -- all.js
-- models/
   -- movies.js
-- views/
    -- error.ejs
    -- index.ejs
    -- layout.ejs
-- tests/
    -- api.test.js
-- node_modules/
-- server.js
-- package.json
```

Once you have this structure in place, it's time to write the actual generator. Yeoman offers a base generator which you can extend to implement your own behavior.

The first task is receive input data from the user while generating the Web application. To accomplish this task, we use the Prompting API from Yeoman. The prompt module is provided by Inquirer.js and you should refer to its API for a list of available prompt options. The prompt method is asynchronous and returns a promise. You'll need to return the promise from your task in order to wait for its completion before running the next one. Listing 1 shows an excerpt from the Kaang generation main file.

The generator starts by asking some questions to the user. For our generator, the questions are:

- **Name of the application**: the name of the folder for the generated application. Also this name will be injected in several configuration files (bower, gulp, etc.) and in the title element of the main HTML file;
- **Use of a Web framework (Bootstrap)**: a yes|no question that will give the user the chance to install Bootstrap and use it to add responsiveness to the main HTML file and for more sophisticated GUI components to be included in the Web application;
- **Use of a JavaScript library (jQuery)**: a yes|no question that will give the user the chance to install jQuery and use it in the event management and AJAX calls in the main JavaScript file of the application;

Listing 1 Main generator file.

```
'use strict';
const Generator = require('yeoman-generator');
const chalk = require('chalk');
const yosay = require('yosay');

module.exports = class extends Generator {
  constructor(args, opts) {
    super(args, opts);
    this.log('Initializing KAANG generator!');
  }
  prompting() {
    this.log(yosay(`Welcome ${chalk.red('generator-kaang')}!`));
    const prompts = [
      {
        type: 'input',
        name: 'name',
        message: 'Your project name?',
        default: this.appname
      },
      ...
    ];
    return this.prompt(prompts).then(props => {
      // To access props later use this.props.name;
      this.props = props;
      ...
    }); }
};
```

- **Use of a test framework (jest)**: a yes|no question that will give the user the chance to use jest to test the API endpoints.
- **Use an API documenter (apiDocs)**: a yes|no question that will give the user the chance to have inline Documentation for the RESTful web API.

Based on the users' answers, Yeoman takes the predefined templates (based on the previous structure) and execute several tasks: injects the user data, made conditional copies, etc. Listing 2 shows the use of the copyTpl method in order to copy a template file with automatic data injection.

The template language used is EJS which aims to help render JavaScript code in the client side. In this case, the value of the name variable is injected in the `_index.ejs` template as shown in Listing 3.

Beyond data injection, the user's data can influence the inclusion of references in the dependency management file (`_bower.json`). That is the case of Bootstrap and jQuery installation which will depend on the users acceptance.

3.3 Routes and Models

Kaang generator creates, by default, an API for movies management. The API exposes the several endpoints. In Table 5, we present some of the available endpoints and a respective description.

Listing 2 Template generation.

```
// Scaffolding
writing() {
    // Copy application files
    this.fs.copyTpl(
      this.templatePath('_views/_index.ejs'),
      this.destinationPath('views/index.ejs'),
      { name: this.props.name }
    );
      ...
  }
```

Listing 3 Basic template file.

```
// _index.ejs
<html>
  <head>
    <title><%= name %></title>
  </head>
  ...
</html>
```

The reference API of the movie resource is composed by three functions. The GET function retrieves all the movies resources. The POST function inserts a new movie passed as a JSON string through a POST parameter. The DELETE function removes a specified movie (id included in the URI of the endpoint). Listing 4 shows the routing of these endpoints to specific function handlers

This file imports the module all.js which contains the concrete implementation of the routes, as presented in Listing 5.

These functions will interact with the Movies model represented as a Mongoose schema. This schema mediates the interaction with the respective MongoDB collection, as shown on Listing 6.

3.4 Testing

Kaang uses Jest, a testing framework written by Facebook, to test the API routes. The best way to organize the API tests is to separate each generator (and sub-generator) into its own describe block. Then, use a test block for each assertion. In code, this should end up with a structure similar to the presented in Listing 7.

Then, you can trigger the tests in the task runner workflow or invoke manually the npm command **npm run test** from the command line. The results are shown in Figure 4.

3.5 Documentation

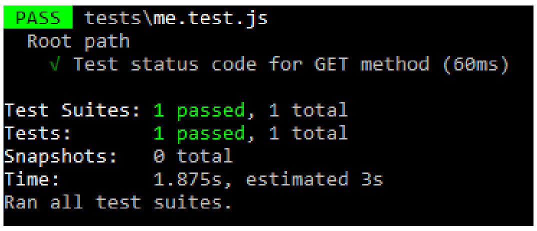
For the documentation, the Kaang generator uses the apiDoc tool that can be defined as an inline documentation generator for RESTful web APIs. This tool creates documentation based on API annotations included in the JavaScript source code. For the use of this tool we had to update the package.json file, namely the devDependencies section to include the

Table 5 Endpoints of the Movies API.

Endpoint	Description
GET /movies	Gets all the movie resources
POST /movie	Creates a new movie resource
DELETE /movie/:movieId	Remove a specified movie resource

Listing 4 Application routing.

```
...
let routes = require('./routes/all');
// Creation of the routes
app.get('/', routes.index);
app.get('/movies', routes.getMovies);
app.post('/movie', routes.postMovie);
app.delete('/movie/:id', routes.removeMovie);
...
```



```
 PASS  tests\me.test.js
  Root path
    √ Test status code for GET method (60ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.875s, estimated 3s
Ran all test suites.
```

Figure 4 Results of API test using Jest.

references to apiDoc and opn-cli (a cross-platform node-open which opens websites, files, executables, etc.).

```
"devDependencies": {
    "apidoc": "^0.17.6",
    "opn-cli": "^3.1.0"
}
```

After that we include a new script in the file:

```
"docs": "apidoc -i routes -o docs && opn docs/index.html"
```

Thus, all we need to do to generate the API docs and show the documentation is type the following command.

```
npm run docs
```

This command opens the respective file in the Web browser (Figure 5).

4 Validation

In order to validate Kaang we enumerate the necessary steps to generate and run a Web application. For that purpose, you just need to execute the following steps to see your Web app running:

Listing 5 Routes implementation.

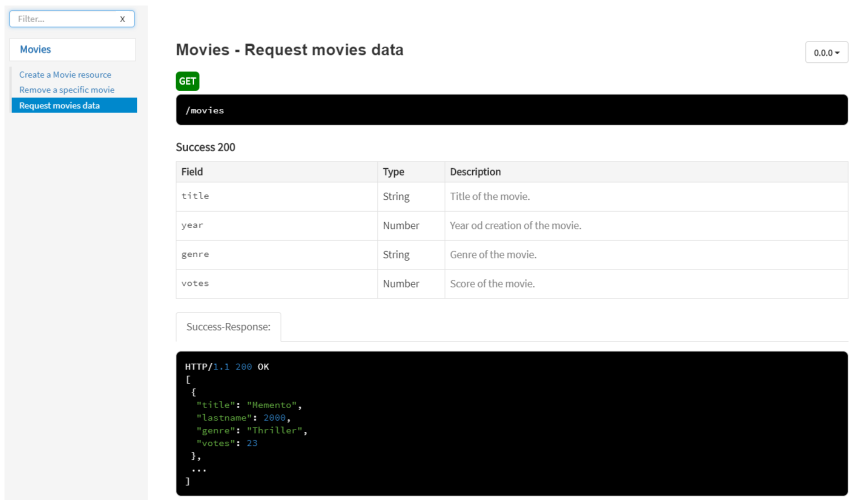
```
//_routes/_all.js
const Movie = require('../models/movie').movies
// Renders the main page (index.ejs)
module.exports.index = function(req, res) {
  res.render('index')
}
// Returns all movies
module.exports.getMovies = function(req, res) {
  Movie.find().exec(function(err, movie) {
    return err ? res.send(err) : res.json(movie)
  })
}
// Adds a new movie
module.exports.postMovie = function(req, res) {
  Movie.create(req.body, function(err, movie) {
    return err ? return res.send(err) : res.json(movie)
  })
}
// Remove an existent movie
module.exports.removeMovie = function(req, res) {
    let movieId = req.params.id
    Movie.find({'_id': movieId}).remove().exec(function(err, movie) {
      return err ? return res.send(err) : res.json(movie)
    })
  }
```

Listing 6 SOS schema for a task.

```
//_model/_movie.js
let mongoose = require('mongoose');
let Schema = mongoose.Schema;
let movieSchema = new Schema({
    title:  String,    year: Number,
    genre: String,     votes: Number
  });
module.exports.movies = mongoose.model('Movie', movieSchema);
```

Listing 7 SOS schema for a task.

```
//_tests/_me.test.js
const app = require('../app')
describe('Root path', () => {
    test('Test status code for GET method', () => {
        return request(app).get("/").then(response => {
            expect(response.statusCode).toBe(200)
        })
    });
})
```



Movies - Request movies data

```
/movies
```

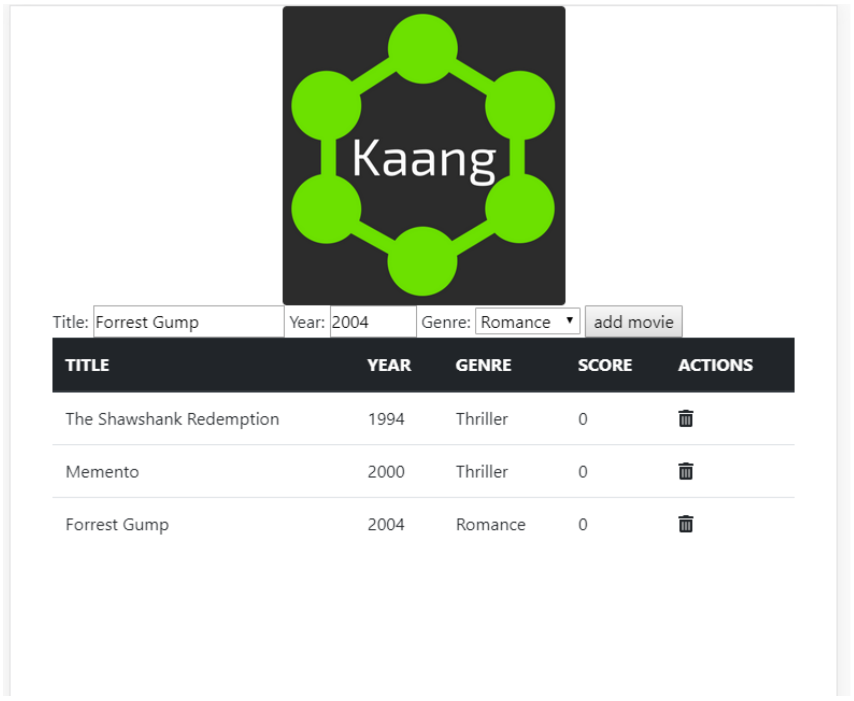
Success 200

Field	Type	Description
title	String	Title of the movie.
year	Number	Year od creation of the movie.
genre	String	Genre of the movie.
votes	Number	Score of the movie.

Success-Response:

```
HTTP/1.1 200 OK
[
 {
  "title": "Memento",
  "lastname": 2000,
  "genre": "Thriller",
  "votes": 23
 },
 ...
]
```

Figure 5 Documentation HTML file.



Title: Forrest Gump Year: 2004 Genre: Romance

TITLE	YEAR	GENRE	SCORE	ACTIONS
The Shawshank Redemption	1994	Thriller	0	🗑
Memento	2000	Thriller	0	🗑
Forrest Gump	2004	Romance	0	🗑

Figure 6 Kaang generator app frontend.

1. Call **mongod** to initiate MongoDB;
2. Generate Kaang's web app typing in the command line: **yo kaang**;
3. Answer the generator questions;
4. Execute **npm start** to start the server;
5. Open a browser and type **http://localhost/8080**.

The GUI of the generated Web application is depicted in Figure 6.

As you can see, it is a simple GUI, for testing purposes. Here, you can browse all the movies, add a new movie or delete an existent movie.

Kaang's source code is available at GitHub[13]. As part of being published to both the npmjs and Yeoman registries, the generator will be integrated on Travis CI. This should provide an addition level of confidence to the generator's end-users. Additionally, Travis CI feeds test results to Coveralls, which displays the generator's code coverage.

5 Conclusions

This paper describes Kaang as a RESTful API generator. The paper comprises several contributions, namely:

- It presents the state of art of REST generators using JavaScript;
- It can be used as a practical guide for those who aim to create their own generators;
- It abstracts the complexity of the creation of REST APIs;
- It fosters the use of generators decreasing the bureaucratic work and repetitive tasks;
- It decreases the learning curve for those who want to enter in the modern Web realm
- It fosters the use of good development practices such as testing and documenting.

As future work, the main idea is to extend this generator to cover other features such as the possibility to automatically generate new models/routes, the support for authentication based on JWT, the support for JavaScript and CSS preprocessors (e.g. CoffeeScript, LiveScript, Less, Sass) and the inclusion of a better module bundler and task runner with predefined runnable tasks (minification, image optimization, etc.). In this context, we expect that the new version of the Kaang generator supports Webpack[14].

References

1 Stefan Baumgartner. *Front-End Tooling with Gulp, Bower, and Yeoman*. Manning Publications, 2017.

2 Marijn Haverbeke. *Eloquent JavaScript*. No Starch Press, 2017.

3 Ricardo Queirós. CSS preprocessing: Tools and automation techniques. *Information*, 9(1), 2018. doi:10.3390/info9010017.

4 Alex Rauschmayer. *Speaking JavaScript: An In-Depth Guide for Programmers*. O'Reilly, 2017.

[13] https://github.com/rqueiros/Kaang
[14] https://webpack.js.org/