Web server generation - user documentation

Domen Vake

June 18, 2021

1 Brandybuck

Brandybuck is a command line tool, that allows developers to generate fully working web REST API web servers. Brandybuck generates a functioning NodeJS application running Express using Typescript. The user may define a JSON configuration file, upon which the application is generated. The features include:

- Model specification
- Database support
- Authentication
- Docker support
- Traefik v2 reverse proxy support

The generated web server may serve as a complete backend appliction for smaller projects, or can be expanded and provide a fast start on the way to the desired project.

- Installing
- Usage
- Init
  - Metadata configuration

  - Docker configuration
  - Trefik configuration

- Models
  - Name
  - CRUD
  - Fields

- Build
  - Folder structure
  - Authentication

- Running the generated server
  - Modifying environment

2 Installing

The tool may be installed from the snap store by running the command

```
snap install brandybuck --edge --devmode
```

3 Usage

The tool is used by running the `brandybuck` command.
It can work in one of two ways. Either you can run it with `init` or `build` command.

4 Init

Before building the web server, Brandybuck has to know what kind od server you want. Running the command

```
brandybuck init
```

will create two files within the directory from which it was run. The files will be called `brandybuck.config.json` and `brandybuck.models.json`. The files are used by the developer to specify the configurations of the web server that should be generated.

4.1 Metadata configuration

By default the `brandybuck.config.json` configuration file will look something like this:

```
{
  "project_name": "Unknown",
  "port": 3000,
  "auth": true,
  "model_source": "./brandybuck.models.json",
  "database": "SQLITE",
  "docker": {
    "port": 3000,
    "traefik2": false
  }
}
```

All the attributes have default values, within the tool, so you don't have to specify all the features within the .json, but only the modifications of the default configuration. Overview of the attributes can be seen in the tables below.

Attribute	Type	Default	Description
project_name	string	"Unknown"	Specifies the name of the generated project. Will resonate in folder structure, package.json and container names.
port	number	3000	Specifies the port the server should listen to.
auth	boolean	false	Specifies whether authentication should be generated.

Attribute	Type	Default	Description
model_source	string	”./brandybuck.models.json”	Specifies the the location of the configration file that describes the models.
database	string	”SQLITE”	Specifies the database that the server should use. (Currently only supports SQLITE)
docker	boolean / configuration object	{ ”port”: 3000, ”traefik2”: false }	Specifies whether and/or how dockerization files should be generated .
log	boolean	true	Adds some aditional logging to the generated server.
documentation	boolean	true	TODO: not yet implemented ;)

4.1.1 Docker configuration

The docker configuration can either be specified by a `boolean` or an `object`. The overview of the object configruation attributes can be seen in the table below.

Attribute	Type	Default	Description
port	number	3000	Specifies the port the docker should expose. Automatically binds to the exposed port to the server port.
traefik2	string / configuration object	false	Specifies whether and/or how traefik v2 configuration should be generated.

4.1.2 Trefik configuration

If the dockerization was enabled, there is an option to configure the container to run with https://docs.traefik.io/v2.0/Traefik v2 by specifying the `traefik2` attribute in docker. The overview of the configurations for traefik can be seen in the table below:

Attribute	Type	Default	Description
container_name	string	”brandybuck_traefik_container_”	Specifies the name of the container. Will resonate in the name of the router and service of traefik container.
proxy_network_name	string	”proxy”	Specifies the name of the external network, so the container is visible to traefik.
certresolver_name	string	”le”	Name of your traefik certresolver.
entrypoint_name	string	”websecure”	Name of your traefik entrypoint.
domain	string	”www.projectname.com”	Domain name of your project.

4.2 Models

Once the metadata of the project have been specified there is one more configuration to be cofigured. We must specify the resources that the server should store and serve. These models will define how your database is structured and how the server routes are generated. The data for this is strored in the `brandybuck.models.json` configuration file.
The model file will come pre-generated with one dummy model, so it is easier to get started. This model is to be modified/deleted in your final configuration.

```
{
  "models": [
    {
      "name": "table",
      "crud": {
        "create": true,
        "create_auth": false,
        "read": true,
        "read_auth": false,
        "update": true,
        "update_auth": false,
        "delete": true,
        "delete_auth": false
      },
      "fields": [
        {
          "name": "col1",
          "data_type": "VARCHAR",
          "null": false
        },
        {
          "name": "col2",
          "data_type": "VARCHAR",
          "null": false
        }
      ]
    }
  ]
}
```

The model configuration files contains a json which is nothing more than just an array of model objects. The model object is constructed from three main attributes: `name`, `crud` and `fields`. Important note is that the server application handles generation od `id` fields by itself so you should not specify the field named `id`. The server uses `uuid` to generate all ids.

4.2.1 Name

Name attribute specifies the name of the resource. It is recommended to name your models in a singular form, in consideration to readability of the generated source code since the generator appends the `s` to the end of the model names for database table names and such (i.e. `post -> posts`).

4.2.2 CRUD

The `crud` field describes which endpoints should be generated for the specified resource (`read`, `create`, `update`, `delete`). Each of the attributes defaults to `true`.
In the case you specified `auth` to `true` in the `brandybuck.config.json` you can specify if the route corresponding to the CRUD function needs to verify the user authentication. You can do that by specifying `read_auth`, `create_auth`, `update_auth` or `delete_auth` (they all default to `false` and are irrelevant if the `auth` value is `false`).

4.2.3 Fields

The `fields` attribute is an array of field object and describes the structure of the model. Each field is described by three main attributes seen in the table below:

Attribute	Type	Default	Description
name	string	”col”	Specifies the name of the model attribute / column.
data_type	string	”VARCHAR”	Specifies the data type of the column. Corresponds directly to the avalible types of the `database` that was specified in the `brandybuck.config.json`.
null	boolean	false	Specifies if the column may contain null values.

Each model may have any number of fields.

5 Build

Once the configuration has been specified it is time to build the project. You can do that by running the command below in a directory, that contains the `brandybuck.config.json` and `brandybuck.models.json` configuration files.

```
brandybuck build
```

5.1 Folder structure

The tool will read the configuration files and create the project. The full build folder structure looks something like this (may differ based on configuration):

```
root/
  |-- Unknown/
  |    |-- app/
  |    |    |-- db/
  |    |    |
  |    |    |-- migrations/
  |    |    |    |--  001-inital-schema.sql
  |    |    |
  |    |    |-- src/
  |    |    |    |-- auth/
  |    |    |    |-- db/
  |    |    |    |-- models/
  |    |    |    |    |-- core/
  |    |    |    |
  |    |    |    |-- routes/
  |    |    |    |-- server.ts
  |    |    |
  |    |    |-- package.json
  |    |    |-- tsconfig.json
  |    |
  |    |-- dockerfile
  |    |-- docker-compose.yml
  |    |-- server.entrypoint.sh
  |
  |-- brandybuck.configuration.json
  |-- brandybuck.models.json
```

5.2 Authentication

If the `auth` attribute was set to `true`, brandybuck will auto generate the `user` model so it is structured in a way that is compatible with the generated source code. You may modify the user later but it is recommended, that upon specifying the auth flag you should not generate own user models. Structure of the user model looks like this:

```
{
  id: string,
  name: string,
  password: string,
  role: 'USER' | 'ADMIN'
}
```

Passwords are hashed and salted and you get the control of setting the `hash salt rounds` in the generated `.env` file (default is 10). Authentication uses `JWT tokens` to authenticate the user.
In the generated `.env` file, there is also a field `ADMIN_EMAIL` where you may specify an email. If a user registers with that email the user will get an `ADMIN` value and others will get the `USER` value.

The user model does not have any CRUD methods and can not be retrieved from the server via an HTTP route. If a user wants to be registered to the server this can be accomplished by sending a `POST` request to to `<domain>/auth/local/register` with the body:

```
{
  "name": "User Name",
  "email": "user@email.com",
  "password": "usersecretpassword"
}
```

From this point the registered user can now login to the application by sending a `POST` request to to `<domain>/auth/local/login` with the body:

```
{
  "email": "user@email.com",
  "password": "usersecretpassword"
}
```

In the response the user will recieve a `Bearer token` that should be send in the header of all requests that require authentication:

```
"Authorization": "Bearer <token>"
```

6 Running the generated server

You can run the generated application directly by moving into the `app` folder and first installing the application dependencies with:

```
cd <name>/app
npm install
```

You can then start the application by running:

```
npm run start
```

or production version by running;

```
npm run prod
```

If you jut want to transpile the application you can run:

```
npm run tsc
```

6.1 Docker

If you specified the dockerization of the project, you can run the docker container with commands:

```
cd <name>/
docker-compose up
```

7 Modifying environment

Branybuck will generate `.env` and `.env.sample` files for both the app and the potential docker configuration. The configurations will be pre-filled but you can change them at any time. For the application itself the `.env.sample` looks like this:

```
PORT=
JWT_SECTRET=
ADMIN_EMAIL=
HASH_SALT_ROUNDS=
SQLITE_DB=
```

The full docker configuration with the traefik support enabled will have an option to modify the domain of the application shown below.

```
PORT=
JWT_SECTRET=
ADMIN_EMAIL=
HASH_SALT_ROUNDS=
SQLITE_DB=
DOMAIN=
```