pistahx

type safe, design first, haxe web api

github.com/mebyz/pistahx - pistahx.io

by Emmanuel BOTROS YOUSSEF / mebyz
emmanuel.botros@gmail.com

pista*hx* (disclamer !) :

pistahx is an open-source “work in progress” project :)

Background : about me

- I started coding at 15, when I got my hand on this old “TV computer”

play the tape until it loads all the data and you’re ready to play on your TV .... then code using MSX-BASIC and save your game on a new tape !

Background : about me

Then my uncle offered me this “portable computer” I was thrilled. I discovered Windows, BAT scripts, games and game dev, and that was it : *coding became a passion.*

Background : me & Haxe :)

After many years working in the IT field, approaching different kind of businesses, technologies, languages, challenges

... I eventually came across the *HAXE* language & toolkit

(only a few months ago)

HAXE

Background : HAXE @ LeKiosk

Now at LeKiosk (my company) Haxe already helps us solve many challenges. We use it for :

- portable javascript libraries (for mobile apps)
- data pipeline workers (shipped in docker containers and deployed in the cloud)
- production desktop tools, using nodejs target / electron
- admin interfaces
- brand new apis for future projects

=> now, Let's dig into pista*hx* itself !

pista*hx* ? APIs : some (common) problems to solve

- Old & slow legacy code AND development tools.
- Heavy costs, technology stickiness (licenses, infrastructure).
- Inexistant API documentation or specification.
- Performance issues (bad or no caching strategies)
- Predictability / Scalability / Deployability issues
- …… NO FUN

".. maybe we can imagine something better for our new api project ?"

pista*hx*

is an effort to solve these issues

by helping you and your team code/deploy/scale

type safe, design first, haxe web api(s)

pista*hx* , haxe to the rescue ! :

- based on **Haxe** => developers write type-safe Business classes
- **Nodejs** target : pistahx apps can be deployed on almost any stack
- ultra fast **Redis** cache store (multiple layers)
- implements **openAPI**, generates interactive api doc and haxe doc using dox
- **pistahx-spec** generates Haxe typedefs from your API specification
- **pistahx-spec** automatically scaffolds your routes & type-safe mappers
- **pistahx-db** generates Haxe typedefs from your DB schema
- **docker containers** provided to build and deploy / scale your apis

=> let’s see some details !

pista*hx* : partial view of pistahx ecosystem

HAXE
node JS
OPEN {API} INITIATIVE
redis
Sequelize
Logic-less templates.
kibana
elasticsearch
logstash
Passport
docker

pista*hx* : bootstrap a new api using pistahx-app

=> http://www.pistahx.io - http://github.com/mebyz/pistahx

```
git clone git@github.com:mebyz/pistahx-app.git && cd pistahx-app

./prepare.sh

gulp
```

pista*hx* : bootstrap a new api using pistahx-app

=> lets have a look at your project structure after cloning

ALL your app logic (design, Haxe business code) REMAINS in the “app” folder

pista*hx* : bootstrap a new api using pistahx-app

package.json file shows that your app will depend on pistahx and some modules :

```
"pistahx": "https://github.com/mebyz/pistahx",
"pistahx-db": "https://github.com/mebyz/pistahx-db",
"pistahx-spec": "https://github.com/mebyz/pistahx-spec",
...
```

=> pistahx is a dependency of your project

pistahx : development workflow

1 => CONFIGURE your api here : ./app/conf/[env].yaml

2 => DESIGN your api here : ./app/api.yaml

3 => IMPLEMENT your Haxe api here : ./app/Business/**

pista*hx* : implements openAPI / “swagger”

=> openAPI is a strong and reknown specification for REST api standardization

=> many openAPI tools for edition, validation, documentation generation are available

ie : editor.swagger.io (online api design editor + validatior)

ie : swagger-ui (online interactive documentation for your api)

pista*hx* : CONFIGURE in ./app/conf/[env].yaml

an example : ./app/conf/local.yaml

```yaml
APP_NAME: pistahx_app
ENV_NAME: local
CACHE_OUT_TTL_DEFAULT: 60
REDIS_HOST: localhost
REDIS_PORT: 6379
DB_HOST: ...
DB_USER: ...
#ELK_SERVER: to be defined
#JWT_SECRET: local_secret_key
#GOOGLE_CLIENT_ID: to be defined
#GOOGLE_CLIENT_SECRET: to be defined
#GOOGLE_CALLBACK_URL: http://localhost:3000/callback

...

...
```

pista*hx* : DESIGN models in ./app/api.yaml

```
definitions:  …

Employee:
  type: "object"
  properties:
   id:
     type: "integer"
     description: "Unique identifier representing a specific Employee"
   lastName :
     type: "string"
     description: "some description"
   firstName :
     type: "string"
     description: "some description"
    ...
```

pistahx : DESIGN models in ./app/api.yaml

```
definitions: ...

Employees:
  type: "object"
  properties:
   result:
     type: "array"
     items:
      $ref: "#/definitions/Employee"

…

…
```

pistahx : DESIGN routes (./app/api.yaml)

```
paths:
  /employees:
    get:
      operationId: employees
      tags:
      - "Employees"
      summary: {'ttl':3600,'xttl':3600,'cachekey':'','xcachekey':''}
      description: "/employees returns a list of employee"
      responses:
        200:
          description: "An array of employees"
          schema:
            $ref: #/definitions/Employees
```

pistahx : IMPLEMENT Haxe Business.hx

```
function get_employees(db : Sequelize, req : ClientRequest, res : ServerResponse ) : Promise<Array<Employee>> {
      return
      DbRepos.dbEmployees.findAll({
        limit : 5
      }).then(function (dbRes) {
        return dbRes.map(EmployeeMapper.dbEmployeeToEmployee);
      });
}

@:publicFields
class DbRepos {
...
...
  var dbEmployees : DBEmployees;
  dbEmployees = db.import_("models/Employee.js");
}
```

pista*hx* : api.yaml > Routes.hx (pistahx-spec)

pistahx-spec is a haxe>nodejs tool that generates haxe code from your yaml api spec

=> AUTO GENERATE routes boilerplate from your api.yaml file with pistahx-spec:

```
type=routes input=../../app/api.yaml output=../../app/Business/Routes.hx node yaml2hx.js
```

(generates the Routes.hx file : contains Haxe express routes definitions for your api)

pista*hx* : api.yaml > Routes.hx (pistahx-spec)

=> resulting file :

```
// this file is GENERATED by pistahx-spec

...

app.get(
	conf.get('BASE_URL')+'/employees',
	cacheo.route({ expire: 3600 }),
	function(req : PistahxRequest, res : Response){
		Business.get_employees(db, req, res, dbcacher, cacheo, {}).then(function(out) { res.send(out); });
	}
);
```

pista*hx* : api.yaml > Td.hx (pistahx-spec)

=> AUTOGENERATE typedefs & mappers with pistahx-spec:

```
type=typedef input=./app/api.yaml output=./app/Business/TD.hx node ./node_modules/pistahx-spec/yaml2hx.js
```

generates the TD.hx file, containing :

- models typedefs from your spec

- (optionnally) api to db mappers if you use the x-dto-... keys in your spec

pista*hx* : api.yaml > Td.hx (pistahx-spec) API MODELS

=> resulting file :

```
// this file is GENERATED by pistahx-spec
...

typedef Employees = List<Employee>;

typedef Employee = {
                        id : Int,
                        lastName : String,
                        firstName : String,
                        title : String,
                        birthDate : Date

...
```

pista*hx* : api.yaml > Td.hx (pistahx-spec) DB MAPPERS

=> AUTOGENERATE db mappers with pistahx-db:

```
Employee:
  x-dto-model : "Employee"
  type: "object"
  properties:
   id:
    x-dto-field: "EmployeeId"
    type: "integer"
    description: "Unique identifier representing a specific Employee"
   lastName :
    x-dto-field: "LastName"
    type: "string"
    description: "some description"
   ...
```

pista*hx* : api.yaml > Td.hx (pistahx-spec) DB MAPPERS

=> resulting file :

```
// this file is GENERATED by pistahx-spec

...

class EmployeeMapper {

	public static function dbEmployeeToEmployee( i : DB__Employee) : Employee {
			var imap = new thx.AnonymousMap(i);
			return {
			id : i.EmployeeId, // => this mapping is defined in the spec (api.yaml)
			lastName : i.LastName,
			firstName : i.FirstName,

...
```

pista*hx* : DB > DB_[MODEL].hx (pistahx-db)

=> AUTOGENERATE db MODELS with pistahx-db:

```
node_modules/pistahx-db/bin/sequelize-auto -d [DB] -o ./app/Business/models/ -e [DIALECT] -h [HOST]
```

=> generates models (Hx typedefs & sequelize models)

=> resulting files :

```
models/[modelname].hx

// will be refactored to haxe soon, sequelize models
models/[modelname].js
models/[modelname].model.js
models/[modelname].repository.js
```

pistahx : DB > DB_[MODEL].hx (pistahx-db)

=> AUTOGENERATE db MODELS with pistahx-db:

```
// this example assumes a "Employee" table exists

// this file is GENERATED by pistahx-db

...

typedef DB__Employee = {
    EmployeeId: Int,
    LastName: String,
    FirstName: String,
    Title: String,
    ReportsTo: Int,
    BirthDate: Date,
    HireDate: Date,
```

pista*hx* : gulp tasks

=> use gulp to build, run or package your app :

gulp build : installs all dependencies needed to build the project, and builds these dependencies

gulp run : transpiles your spec and business (your app) to distrib/out folder and launch the api

gulp pack : like 'gulp run', but does not run the api

…

…

more with pista*hx* : going wild ! => deploy / scale

"gulp" build results in a runnable distribution package : ./distrib/out

=>this folder can be deployed directly on the cloud.

ie : using AWS elasticbeanstalk, after build:

- cd distrib/out
- eb init
- eb deploy

.. set the ENV variable on the ebs panel so your api knows which env it should run in !

more with pista*hx* : containers (Docker)

To run / deploy your (builded) app in a docker container, you can use the public "mebyz/pistahx-docker-stack" docker image hosted on dockerhub

(based on Alpine linux / node v4.x)

sample Dockerfile :

```
FROM mebyz/pistahx-docker-stack
MAINTAINER emmanuel.botros@gmail.com
WORKDIR /distrib/out
COPY /distrib/out
EXPOSE  3000
CMD ["node", "/app/app.js"]
```

more with pistahx : pro-active cache invalidation !

using pistahx “x-cache-flush” definitions, you can design your own cache invalidation strategies and unleash the true power of your cache cluster (= boost your api performances !)

```
/employee:
  put:
    operationId: employee
    tags:
    - "Employees"
    summary: ""
    description: "put /employee saves an employee to db"
    x-cache-flush:
    - "/employees"      < when modifying an employee (PUT), the /employees (GET) cache will be flushed
    ...
```

pista*hx* : live sample

this sample will show you how to add routes and use models and associatons with pisthax:

0. Lets bootstrap a new pistahx api (and pretend we already configured our developpment env and db) to go faster

(we'll use a local redis cache server and a small demo sqlite DB)

pista*hx* : live sample

your db contains an Artist and an Album tables. you would like to open new api routes for your apps to be able to show albums and artists data.

1. define the album and artist definitions & routes in the yaml file :

pista*hx* : live sample

2. add the album and artist models to our repositories in the Business.hx file

pista*hx* : live sample

3. implement the get_albums and get_album methods in the Business.hx

pista*hx* : live sample

4. fast rebuild using “gulp run” and let’s see the result …

pista*hx* : live sample

TADAA !

the doc now show 2 new routes you can use to get albums

(or one album), along with their respective artist’s name.

pista*hx*, what's next ? => road map to v1.0

- implement (generate ?) an haxe typedef for the complete openAPI spec
- refactor remaining js tasks to haxe (ie : pistahx-db)
- implement other targets than nodejs only ?
- get rid of remaining “untyped” or Dynamic in the code..
- optionnal CRUD generation gulp tasks : get/set/add/... routes
- an api.yaml generator from your db schema
- implement the “legacy api booster” functionnality ...

pistahx

THE END ;)

thanks a lot for your attention & welcome on board to newcomers on the pistahx project !!