

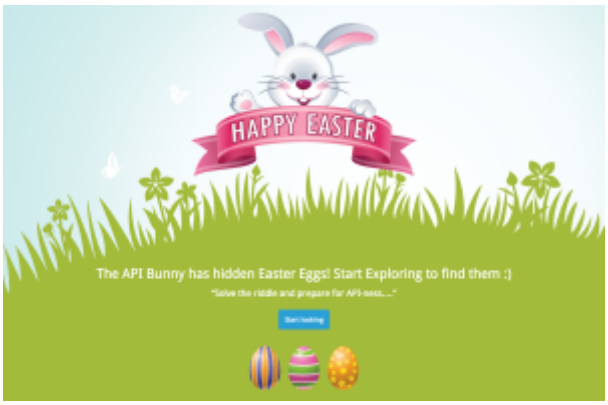
Nicolas
Posted on 04/23/2014

Building APIbunny – using Fortune.js and JSONAPI

If you're part of the API world you may have heard of the APIbunnycontest we launched last Friday. Here's the story behind this 3-day project.

What is people's favorite sport during Easter? Egg hunting of course. At 3scale we came up with the idea of doing something special for Easter, a way to celebrate the holiday pastime of egg hunting in a geeky way. Something where hackers won't actually have to go outside – but can still have fun with a challenge in the spirit of the season!

The idea was to build a maze where hackers need to find the exit, and end on a winning page to claim their prize, and finally, tweet about their quest. APIbunny was born.



We decided to make it hypermedia-based both to try something new, and because it may also be easier for humans to pick up. I then took a day to read as much as possible about HyperMedia APIs. I watched great talks by Steve Klabnik, read most of the Slideshare presentations I could find on the topic, and found a great tutorial from Jason Rhodes that guides you through RPC, REST and Hypermedia.

Technology used

In my early programming days I used PHP and Ruby almost exclusively, I am now in my Javascript phase and I don't see myself using other technology now. I was delighted to find that at the end of his tutorial, Jason gives an overview of Fortune.js, a framework dedicated to build Hypermedia APIs in Javascript. Exactly what I was looking for ! :)

It seems so simple, you define objects and relations between them and the framework builds the links. We used two resources, maze and cell. Maze have multiple cells and a cell belongs to a maze, it also linked to other cells.

How it looks like on Fortune.js

```
var mazeAPI = fortune({
 db: "./db/maze-data"
});

mazeAPI.resource('maze',{
    name: String,
    cells: ['cell'],
    start: {ref: 'cell',inverse:'null'}
});

mazeAPI.resource('cell',{
    name: String,
    readableId: Number,
    north: {ref:'cell', inverse:'south'},
    east: {ref:'cell', inverse:'west'},
    south: {ref:'cell', inverse:'north'},
    west: {ref:'cell', inverse:'east'},
    maze: {ref: 'maze'},
});
```

We just those few lines of code we defined our API and our endpoints. The collections defined like this come with HTTP actions already built-in. So you have access to `GET`, `POST`, `DELETE`, `PUT`, `PATCH` on `/mazes` and `/cells`.

In Fortune.js you can also inverse relationship. If cell A has a *east*relationship with cell B it also means that B has a *west* relationship with B. That could be really useful, but it doesn't really work well in the case of a maze.

```
| 1 | 4 | 7 |
| 2 | 5 | 8 |
| 3 | 6 | 9 |
```

Cell #4 should have

```
{
 west:1
 east:7
 south:5
}
```

But because of inverse #4 is also at *east* of #7 so #4 *west* becomes 7.
So we would have to use inverse `null` instead.

Generating the maze

To describe the maze I used the same input format as Mike Amundsen in his book. A javascript file with an array of cells with their properties.
The script that generates the maze will add cells to the maze by calling the API. When all the cells are created it will go through the file again and build relationships between cells using `PATCH` requests. This is also how we defined the start point of the maze.

Exiting the maze

Because we wanted the API to be machine-readable, we added `type` attributes to cells to recognize *start* and *exit* points from other cells. Reaching the exit cell of the maze was only part of solving the quest. We wanted to send hackers to a winning page where they could share their pride in winning the quest. Ideally this winning page would be unique, requiring hackers solve the maze – not just find the winning page url.



That's where a lot of people got stuck. The *exit_link* attributes was not Hypermedia compliant, and should have been in the links array. To do so, we should have declared it as a reference in Fortune.js to cell object, and then the *exit_link* would have been accessible in each cell. And it would have been easier to find the url to unlock the winning page.

When hackers got this link, most of them got stuck and did not find how to use it. They understood that they should do a `POST` request to it, but the format of the request was not the usual REST POST request, because Fortune.js is generating a JSON-API type of hypermedia API.

In APIbunny the call looks like this

```
{"users":[{"twitter_handle":"mytwitterhandle"}]}
```

We dropped hints about this on the landing page, telling that we were using JSON-API. Some of you found it in the official documentation.

"Protect" the API

When you launch this kind of challenge and ask hackers to hack it, you can expect them to try *everything*. Really. Everything. To prevent the most obvious tries, I "blocked" some of the routes. Fortune.js embed express so it was easy to prevent people from trying calls on */cells,* or */users*. Also just by adding `.readOnly()` to a resource in Fortune.js it enables only `GET` request to it.

Sorry if you tried to modify the maze, but we could not let you do that ;)

What happened on D-day ?

We launched APIbunny.com on Friday. Though our soft launch on Hacker News did not quite work as we expected, we got a lot of interest from the API community – especially from APIscene.com and Api-Craft group.

We received more than 4000 visits on the first day, with more than 1000 visits on the winning pages shared by hackers. Only 30 people have solved it so far. You can see live stats on the public dashboard powered by Keen.io.

Congratulations to Kern Patton for winning the first prize a ticket to the next API Strategy & Practice Conference (APIStrat) in Chicago in September. The others have won 3scale swag and a discounted ticket to APIStrat.

We also want to highlight others hackers for sharing their solution on github.

- Tiborvass in Go – source
- Dyokomizo in Bash – source
- Jschmid in Node – source
- Estoner using Ember.js – source
- Mikz in Javascript – source
- Hannesvdvreken in LaravelPHP – source
- Ianthaler is offering a Hydra-based solution – source

Now APIbunny code is available on Github, feel free to modify it and write clients, we will share it here.

Special mention to Steve Willmott for being open about creating this type of project, Vanessa Ramos for handling on-the-spot social media during launch day and the whole 3scale team for beta testing APIbunny :)