Building JavaScript Microservices with Node.js

When your JavaScript application grows in size you start facing challenges with maintaining the code, fixing bugs, and implementing new features. Also, adding new developers to the project becomes complicated.

Applications are built from pieces, like packages and modules, but at some point those structures aren't enough to reduce the size and complexity of the application. The idea behind distributed systems is to break big, monolithic designs into small, independent programs which communicate with each other to exchange data and perform operations.

One of the many variants of distributed systems is the microservices architecture, which structures an application as a collection of loosely coupled services. Services are fine-grained and the communication protocols are lightweight (like the HTTP protocol).

There are few things worth emphasizing about the superiority of microservices, and distributed systems generally, over monolithic architecture:

- Modularity – responsibility for specific operations is assigned to separate pieces of the application
- Uniformity – microservices interfaces (API endpoints) consists of a base URI identifying a data object and standard HTTP methods (GET, POST, PUT, and DELETE) used to manipultate the object
- Robustness – component failures cause only the absence or reduction of a specific unit of functionality
- Maintainability – system components can be modified and deployed independently
- Scalability – instances of a service can be added or removed to respond to changes in demand.
- Availability – new features can be added to the system while maintaining 100% availability.
- Testability – new solutions can be tested directly in the ''battlefield of production'' by implementing them for restricted segments of users to see how they behave in real life.

In addition, every microservice can be written using the language, technique, or framework that's most appropriate to the tasks it will perform. The only feature that is necessary is the ability to publish RESTful APIs for communication with other services.

To accomplish the tasks you will need the following:

- Node.js and npm (The Node.js installation will also install npm.)

Create the heroes service

Go to the directory under which you'd like to create the project and create following directory and file structure:

```
./heroes/heroes.js
```

If you'd like to use source code control, this would be a good time to initialize a repository. Don't forget to add a *.gitignore* file if you're using Git.

Initialize the npm project inside *./heroes* directory and install necessary dependencies by executing the following command instructions:

- **npm init –y**
- **npm install express body-parser**

It's time to implement the service. Add this JavaScript code to the *./heroes/heroes.js* file:

heroes.js

heroes > heroes.js > heroes

```
const express = require('express');
const path = require('path');
const bodyParser = require('body-parser');

const port = process.env.PORT || '3000';
const app = express();
app.use(bodyParser.json());

const powers = [
  { id: 1, name: 'flying' },
  { id: 2, name: 'teleporting' },
  { id: 3, name: 'super strength' },
  { id: 4, name: 'clairvoyance'},
  { id: 5, name: 'mind reading' }
];

const heroes = [
  {
      id: 1,
      type: 'spider-dog',
      displayName: 'Cooper',
      powers: [1, 4],
      busy: false
  },
```

```
  {
    id: 2,
    type: 'flying-dogs',
    displayName: 'Jack & Buddy',
    powers: [2, 5],
    busy: false
  },
  {
    id: 3,
    type: 'dark-light-side',
    displayName: 'Max & Charlie',
    powers: [3, 2],
    busy: false
  },
  {
    id: 4,
    type: 'captain-dog',
    displayName: 'Rocky',
    powers: [1, 5],
    busy: false
  }
];

app.get('/heroes', (req, res) => {
  console.log('Returning heroes list');
  res.send(heroes);
});

app.get('/powers', (req, res) => {
  console.log('Returning powers list');
  res.send(powers);
});

app.put('/hero/:heroId', (req, res) => {
  const heroId = parseInt(req.params.heroId);
  const foundHero = heroes.find(subject => subject.id === heroId);

  if (foundHero) {
    for (let attribute in foundHero) {
      if (req.body[attribute]) {
        foundHero[attribute] = req.body[attribute];
        console.log(`Set ${attribute} to ${req.body[attribute]} in hero: ${heroId}`);
      }
    }
    res.status(202).header({Location: `http://localhost:${port}/hero/${foundHero.id}`}).send(foundHero);
  } else {
    console.log(`Hero not found.`);
    res.status(404).send();
  }
});

app.post('/hero', (req, res) => {
  console.log('Adding hero');
  heroes.push(req.body);
  res.send(heroes);
});

console.log(`Heroes service listening on port ${port}`);
app.listen(port);
```

The code for the heroes service provides:

- a list of heroes super powers
- a list of heroes
- API endpoints to get the list of heroes, to add a new hero and update a hero's profile

Test the heroes.js service

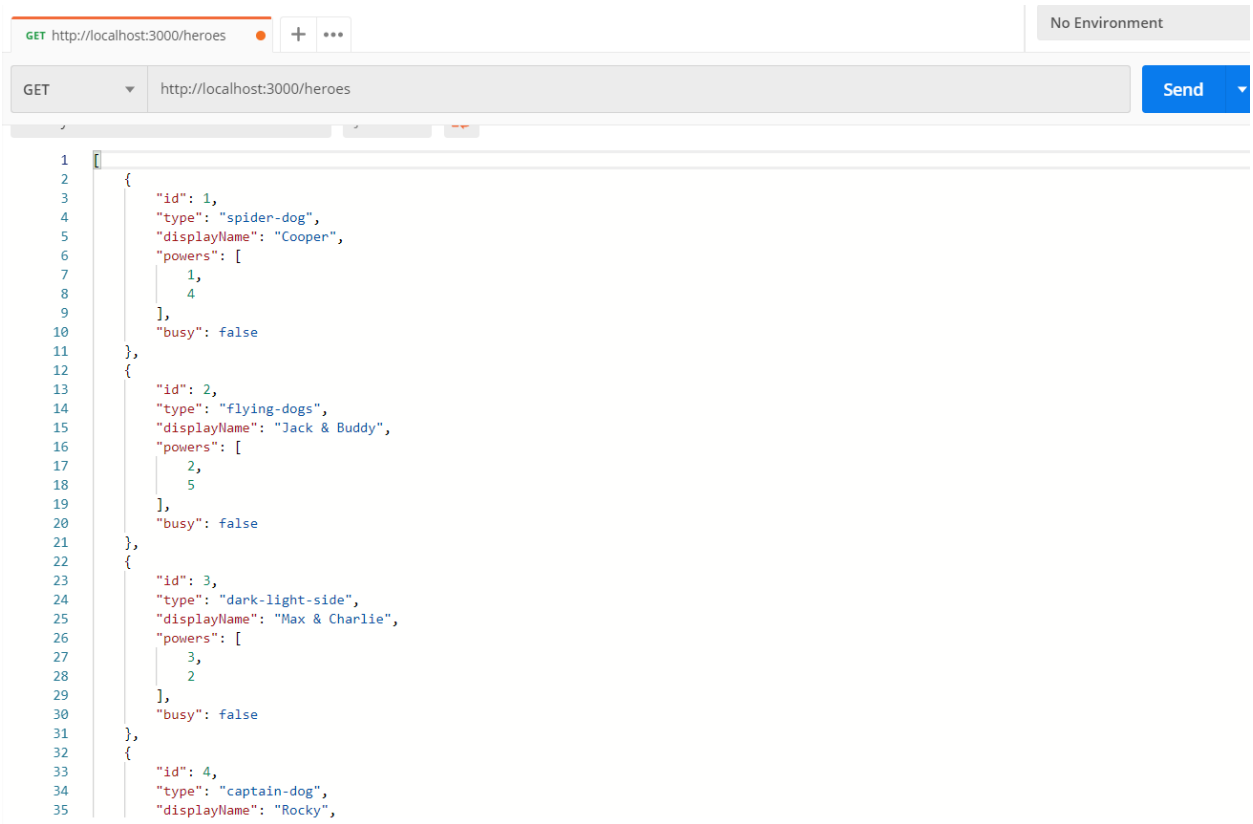
Run the service by executing the following command line instruction:

node ./heroes/heroes.js

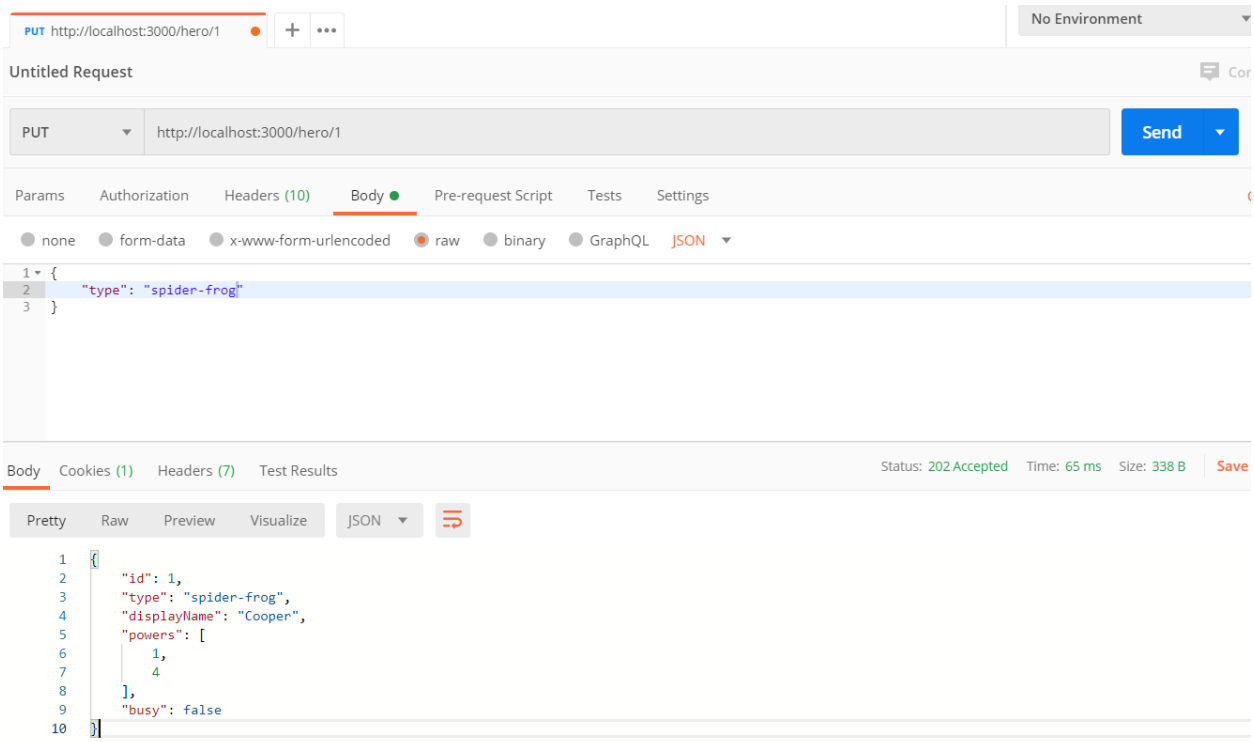
You can check to see if the service works as expected by using **Postman** or your browser.

In Postman the body of the response should look like this:

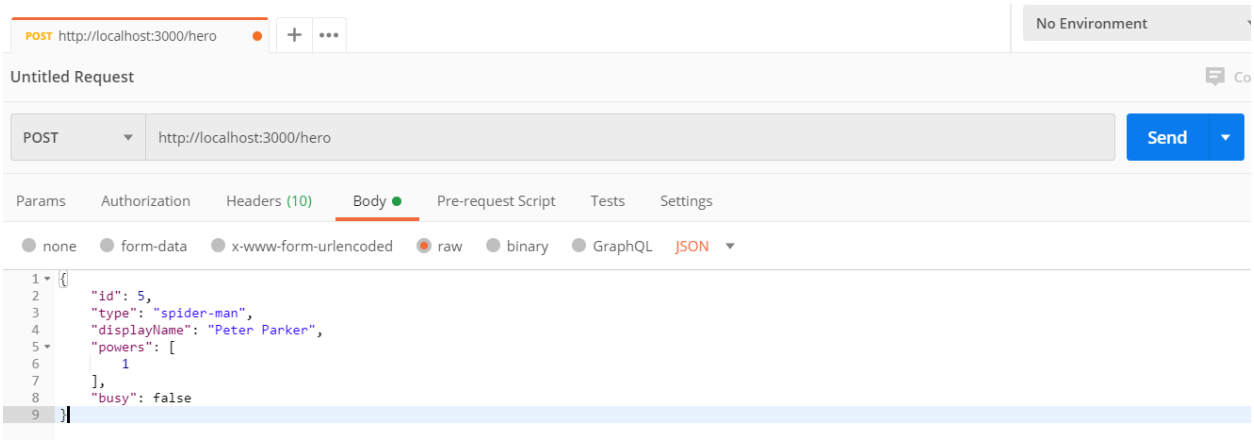
GET request - /heroes



PUT request - /hero/:herold



POST request - /hero



Response (list of heroes with a new hero added)

Create the threats service

The microservices architecture of our application uses a separate service to represent the challenges that only a superhero can overcome. It also provides an API endpoint for matching superheros to threats.

The procedure for creating the threats service is similar to heroes service. In the main directory of your project, create following directory and file structure:

```
./threats/threats.js
```

In the *./threats* directory, initialize the project and install its dependencies with the following npm command line instructions:

- **npm init –y**
- **npm install express body-parser**

Don't forget to include this project in source code control, if you're using it.

Place this JavaScript code in the *./threats/threats.js* file:

threats.js

threats > threats.js > app.get('/threats') callback

```
const express = require('express');
const bodyParser = require('body-parser');
const request = require('request');

const port = process.env.PORT || '3001';
const app = express();

app.use(bodyParser.json());

const heroesService = 'http://localhost:3000';

const threats = [
  {
      id: 1,
      displayName: 'Pisa tower is about to collapse.',
      necessaryPowers: ['flying'],
      assignedHero: 0
  },
  {
      id: 2,
      displayName: 'Engineer is going to clean up server-room.',
      necessaryPowers: ['teleporting'],
      assignedHero: 0
  },
  {
      id: 3,
      displayName: 'John will not understand the joke',
      necessaryPowers: ['clairvoyance'],
      assignedHero: 0
  }
];
```

threats.js

threats > threats.js > threats > assignedHero

```
33  app.get('/threats', (req, res) => {
34    console.log('Returning threats list');
35    res.send(threats);
36  });
37
38  app.post('/assignment', (req, res) => {
39    request.put({
40        headers: {'content-type': 'application/json'},
41        url: `${heroesService}/hero/${req.body.heroId}`,
42        body: `{
43            "busy": true
44        }`
45    }, (err, heroResponse, body) => {
46        if (!err) {
47            const threatId = parseInt(req.body.threatId);
48            const threat = threats.find(subject => subject.id === threatId);
49            threat.assignedHero = req.body.heroId;
50            res.status(202).send(threat);
51        } else {
52            res.status(400).send({problem: `Hero Service responded with issue ${err}`});
53        }
54    });
55  });
56
57
58  console.log(`Threats service listening on port ${port}`);
59  app.listen(port);
```

Apart from the threats list, and basic methods like listing them, this service also has a POST method, *`/assignment`*, which attaches a hero to the given threat:

```
38  app.post('/assignment', (req, res) => {
39    request.post({
40        headers: {'content-type': 'application/json'},
41        url: `${heroesService}/hero/${req.body.heroId}`,
42        body: `{
43            "busy": true
44        }`
45    }, (err, heroResponse, body) => {
46        if (!err) {
47            const threatId = parseInt(req.body.threatId);
48            const threat = threats.find(subject => subject.id === threatId);
49            threat.assignedHero = req.body.heroId;
50            res.status(202).send(threat);
51        } else {
52            res.status(400).send({problem: `Hero Service responded with issue ${err}`});
53        }
54    });
55  });
```

Because the code implements inter-services communication, it needs to know the address of the heroes service, as shown below. If you changed the port on which the heroes service runs you'll need to edit this line:

```
10   const heroesService = 'http://localhost:3000';
```

Test the threats service

If you've stopped the heroes service, or closed its terminal window, restart it.

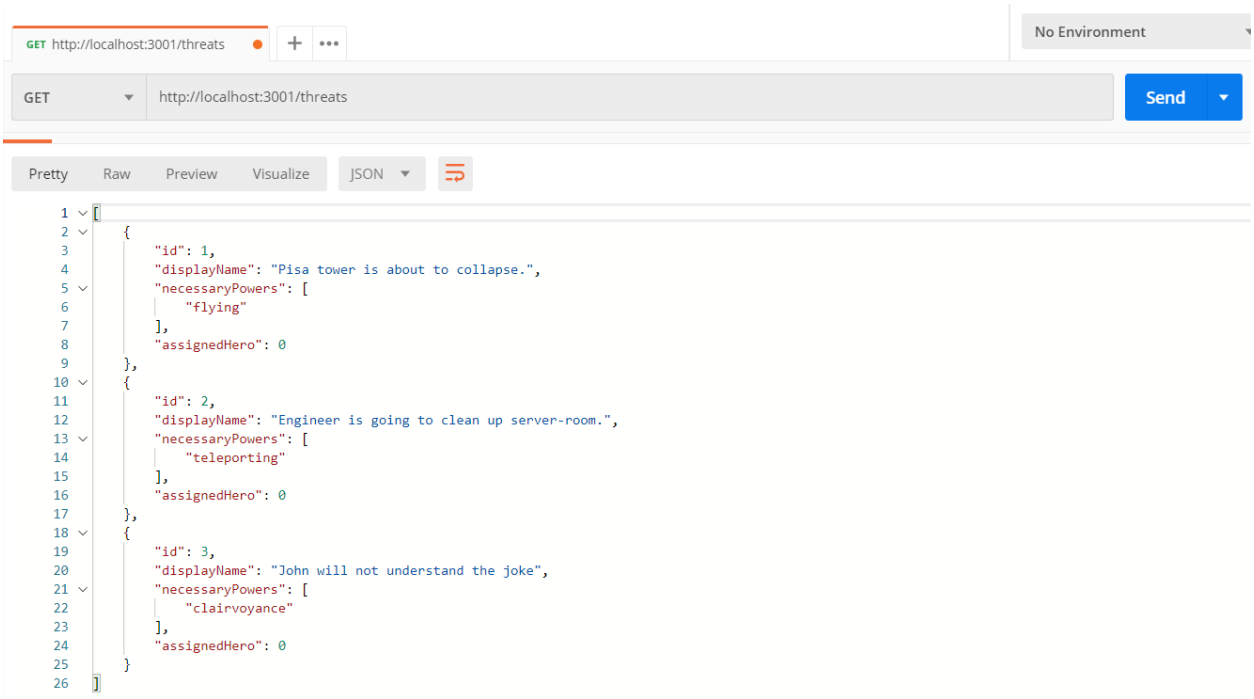
Open another terminal window and start the threats service by executing the following command line instruction:

```
D:\node-soa>node ./threats/threats.js
Threats service listening on port 3001
```

In the same way you tested the heroes service, test the threats service by executing a web request using Postman or your browser.

In Postman the body of the response should look like this:

GET request - /threats



POST request - /assignment