Comparison of Node Backend Framework

Xintong Ma
Rice University

ABSTRACT

Express.js is the most popular Node.js web application framework used today. It seems to be the base dependency in most Node.js web applications, even some popular frameworks like Sails.js are built off of Express. However Express is not perfect, There is no built in error handling in express, it is easy to get lost in all of the middleware that could be added to solve a solution, and there are many ways to do one thing. In this paper, I will compare Express with other popular backend frameworks like Koa and Hapi. Every framework has advantages over other frameworks, express is mature, Koa makes writing middleware much easier, Hapi is good for large teams. Developers should choose framework based on their situation.

INTRODUCTION

Express was first introduced in 2009 by TJ Holowaychuk, it has the biggest community not only out of the three frameworks compared here but out of all the web application frameworks for Node.js. It is the most matured framework. It offers a simple way to get a server up and running and promotes code reuse. And it is also the base for a lot of other frameworks. Following are several popular Node.js frameworks built on Express:

- Feathers: Build prototypes in minutes and production ready real-time apps in days.
- ItemsAPI: Search backend for web and mobile applications built on Express and Elastic search.
- KeystoneJS: Website and API Application Framework / CMS with an auto-generated React.js Admin UI.
- Kraken: Secure and scalable layer that extends Express by providing structure and convention.
- LEAN-STACK: The Pure JavaScript Stack.
- LoopBack: Highly-extensible, open-source Node.js framework for quickly creating dynamic end-to-end REST APIs.
- Sails: MVC framework for Node.js for building practical, production-ready apps.
- Blueprint: Highly-configurable MVC framework for composing production-ready services from reusable components.
- MEAN: Opinionated full stack JavaScript framework that simplifies and accelerates web application development.

Hapi was introduced in 2011, it was originally built on top of Express. Later it was developed into its own framework. The framework has a robust plugin system and numerous key features, including input validation, configuration-based functionality, implement caching, error handling, logging and more.

Hapi is based on configuration over code, this is very useful for large teams to add consistency and reusability. Also this framework is being backed by WalmartLabs as well as many other big name companies like Disney, Concrete and PayPal, this situation make sure Hapi continuing to mature into a great framework.

Koa is also introduced by TJ Holowaychuk, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs. Through leveraging generators Koa allows you to ditch callbacks and greatly increase error-handling. This also improves readability of the application. Koa does not bundle any middleware within core and it is basically a bare bone framework where the developer can pick the middle ware they want to use rather than compromising to the middleware that comes with Express or Hapi.

COMPARISON

1. Creating a server

The first step for any developers when working on a Node.js web application is to create a basic server, like we did in class. So let us create a server using each framework to see their similarities and differences.

Express

```
1  var express = require('express');
2  var app = express();
3
4  var server = app.listen(3000, function() {
5      console.log('Express is listening to http://localhost:3000');
6  });
```

Fig. 1. Use Express to create a server

Code to create a server using Express is shown in figure 1. We require express and then instantiate it by assigning it to the variable app. Then instantiate a server to listen to a port, port 3000.

Koa

```
1  var koa = require('koa');
2  var app = koa();
3
4  var server = app.listen(3000, function() {
5      console.log('Koa is listening to http://localhost:3000');
6  });
```

Fig. 2. Use Koa to create a server

Code to create a server using Koa is shown in figure 2. You can see that Koa is similar to Express. You just require Koa instead of express.

Hapi

```
var Hapi = require('hapi');
var server = new Hapi.Server(3000);

server.start(function() {
    console.log('Hapi is listening to http://localhost:3000');
});
```

Fig. 3. Use Hapi to create a server

Code to create a server using Koa is shown in figure 3. Hapi is the unique one of the group. First like always, Hapi is required but instead of instantiating a Hapi app, you create a new Server and specify the port. In Express and Koa we get a callback function but in Hapi we get a new server object. Then once we call server.start() we start the server on port 3000 which then returns a callback.

2. Routes

Now I will compare one of the most important features of a server, routing. First create the "Hello world" application for each framework and then move on to something a little more useful, REST API.

1. Hello world

Express

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
    res.send('Hello world');
});

var server = app.listen(3000, function() {
    console.log('Express is listening to http://localhost:3000');
});
```

Fig. 4. Use Express to create an endpoint

Code to create an endpoint using Express is shown in figure 4. We are using the get method to catch the incoming request of "GET /" and then invoke a callback function that handles two parameters req and res. For this example we are only utilizing res to return back to the page a string using send. Express has a variety of built in methods that are used to handle the routing functionality. The following are some of the more commonly used methods that are supported by Express: get, post, put, head, delete.

Koa

```
var koa = require('koa');
var app = koa();

app.use(function *() {
    this.body = 'Hello world';
});

var server = app.listen(3000, function() {
    console.log('Koa is listening to http://localhost:3000');
});
```

Fig. 5. Use Koa to create an endpoint

Code to create an endpoint using Koa is shown in figure 5. Here any function preceded by a star means the function will return a generator object. Within the app.use() the generator function sets the response body. In Koa the context which is equivalent to the identifier is an encapsulation of node's request and response objects. “this.body” is a method in the Koa Response. This example we used one of the few middleware provided in the Koa core. This middleware we used catches all routes and responds with the string provided.

Hapi

```
var Hapi = require('hapi');
var server = new Hapi.Server(3000);

server.route({
    method: 'GET',
    path: '/',
    handler: function(request, reply) {
        reply('Hello world');
    }
});

server.start(function() {
    console.log('Hapi is listening to http://localhost:3000');
});
```

Fig. 6. Use Hapi to create an endpoint

Code to create an endpoint using Hapi is shown in figure 6. Here we are using the built in method that the server object provides us, server.route() which has the following options: path, method and handler. The HTTP method can handle the typical requests get, put, post, delete.

2. REST API

The Hello world never really does much other than shows the most basic way to get an application up and running. REST APIs are almost a must in all data heavy applications and will help better understand how these frameworks can be used.

Express

```
1   var express = require('express');
2   var app = express();
3   var router = express.Router();
4
5   // REST API
6   router.route('/items')
7   .get(function(req, res, next) {
8     res.send('Get');
9   })
10  .post(function(req, res, next) {
11    res.send('Post');
12  });
13
14  router.route('/items/:id')
15  .get(function(req, res, next) {
16    res.send('Get id: ' + req.params.id);
17  })
18  .put(function(req, res, next) {
19    res.send('Put id: ' + req.params.id);
20  })
21  .delete(function(req, res, next) {
22    res.send('Delete id: ' + req.params.id);
23  });
24
25  app.use('/api', router);
26
27  // index
28  app.get('/', function(req, res) {
29    res.send('Hello world');
30  });
31
32  var server = app.listen(3000, function() {
33    console.log('Express is listening to http://localhost:3000');
34  });
```

Fig. 7. Use Express to create an API

Figure 7 is API written in Express. We added our REST API to our existing Hello World application. Express offers a little shorthand for handling routes. First we create a Router() object then with the help of router.route() we can define different method under the same path without writing the path multiple times. This is a nice approach because it reduces the chance of developer errors and minimizes the places to change the HTTP method verbs. Finally, use() function help us to add the middleware we just wrote to the specific path.

Koa

```
var koa = require('koa');
var route = require('koa-route');
var app = koa();

// REST API
app.use(route.get('/api/items', function*() {
    this.body = 'Get';
}));
app.use(route.get('/api/items/:id', function*(id) {
    this.body = 'Get id: ' + id;
}));
app.use(route.post('/api/items', function*() {
    this.body = 'Post';
}));
app.use(route.put('/api/items/:id', function*(id) {
    this.body = 'Put id: ' + id;
}));
app.use(route.delete('/api/items/:id', function*(id) {
    this.body = 'Delete id: ' + id;
}));

// all other routes
app.use(function *() {
  this.body = 'Hello world';
});

var server = app.listen(3000, function() {
  console.log('Koa is listening to http://localhost:3000');
});
```

Fig. 8. Use Koa to create an API

Figure 8 is API written in Koa. It's pretty obvious that Koa doesn't have the ability to reduce the repetitive route verbs like Express. It also requires a separate middleware to handle routes. Koa-route is a good choice because it is maintained by the Koa team but there are a lot of routes available to use by other maintainers. The routes are very similar to Express with using the same keywords for their method calls like get, put. The most exciting feature of Koa is that it has the best speed of all three frameworks

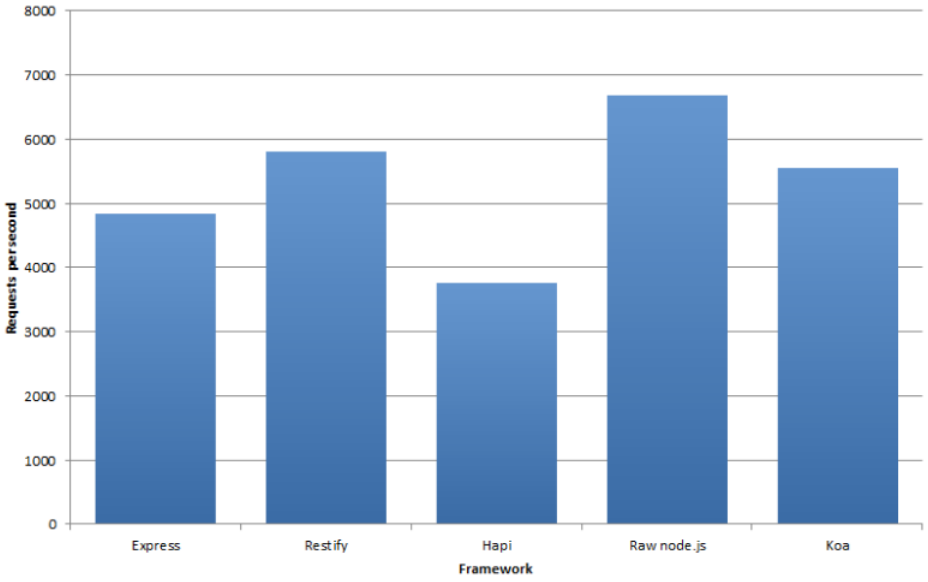

Fig. 9. Compare framework speed

Hapi

```
var Hapi = require('hapi');
var server = new Hapi.Server(3000);
server.route([
  { method: 'GET',
    path: '/api/items',
    handler: function(request, reply) {
      reply('Get item id');
    }
  },
  { method: 'POST',
    path: '/api/items',
    handler: function(request, reply) {
      reply('Post item');
    }
  },
  { method: 'PUT',
    path: '/api/items/{id}',
    handler: function(request, reply) {
      reply('Put item id: ' + request.params.id);
    }
  },
  { method: 'DELETE',
    path: '/api/items/{id}',
    handler: function(request, reply) {
      reply('Delete item id: ' + request.params.id);
    }
  },
  { method: 'GET',
    path: '/',
    handler: function(request, reply) {
      reply('Hello world');
    }
  }
]);
server.start(function() {
  console.log('Hapi is listening to http://localhost:3000');
});
```

Fig. 10. Use Hapi to create an API

Figure 10 is API written in Hapi. First impressions of the routes in Hapi are how clean and readable they are compared to the other frameworks. Even the required options method, path, handler, and replay for the routes are easy to the eye. Like Koa, there is a lot of reuse of code making the room for error larger. However this is intention, Hapi is more concerned about configuration and wants the code to be cleaner for easier use in a team. Hapi also wanted to improve error handling which it does without any code being written on the developers end.

CONCLUSION

We have seen some good and practical examples of all three frameworks. As we previously discussed, Express is definitely the most popular and most recognized framework of the three. It is almost a reaction to first create a server using Express when starting new development on an application but hopefully now there might be some thought involved whether to use Koa or Hapi as an alternative. Koa shows real

promise for the future and it is the fastest framework. Hapi has a lot of nice plugin features. Today, these Node.js frameworks are shaping the future of web and application development technology. I am sure that with the on-going development in the field there will be a lot more nice frameworks in the near future.

REFERENCE

1. https://expressjs.com
2. http://hapijs.com