Restify vs. Sails vs. Hapi: Node.js Framework Comparison

CHRIS TOBOLSKI | APRIL 26, 2019 | DEVELOPER TIPS, TRICKS & RESOURCES (HTTPS://STACKIFY.COM/DEVELOPERS/)

In the last decade, Node.js (https://stackify.com/node-js-blogs-youtube-channels/) has rapidly risen in popularity as a server-side programming language. With that popularity, we've seen no shortage of frameworks aiming to make developers' lives easier. Although they

are lesser-known than other Node.js frameworks (https://stackify.com/node-js-framework/), Hapi (https://hapijs.com/), Sails (https://sailsjs.com/), and Restify (http://restify.com/) are a few options that have a lot to offer.

It can be difficult to know what framework is the best fit for your needs. As software developers, we're often asked to make technology choices with limited information. In this post, I'm going to pretend I have to choose one of these three frameworks for a new app. By doing a little homework as if I had to choose, I hope to make your life a bit easier. I'll compare these frameworks in a few ways that I think are important when choosing a technology. Throughout the process, I'll assign points to each framework that seems to best meet the criteria being discussed.

What's a framework anyway?

A framework is a collection of tools that provide common functionality for a certain type of application. Frameworks allow developers to quickly start working on business outcomes. We can start solving *our* problems instead of those that have been solved already. In our case, Sails, Hapi, and Restify are server-side application frameworks. The most obvious functionality a server-side framework should provide is the ability to handle HTTP requests.

What about Sails, Hapi, and Restify?

Sails, Hapi, and Restify are all frameworks, but they weren't all built with the same intent. Sails is marketed as a Rails-style MVC framework. That means it's going to offer a lot of features, but it will probably be complex to take full advantage of. Hapi is a web and services application framework. It can still do MVC, but it's positioned as a more balanced framework. A Hapi application could expose services or a web app. Restify explicitly states in its documentation (http://restify.com/) that it is a web service framework focused on building "semantically correct RESTful web services."

So what does that mean for us?

It means that while we're going to evaluate these frameworks relative to each other, our choice will ultimately be contingent on the goals of our application. For example, if we know we need an MVC framework, we might eliminate Restify altogether. However, we can still compare these frameworks in some common areas.

Scoring strategy

I'm going to score these frameworks in relation to each other on a number of different factors. I'll give points to one or more based on their merits for each point of comparison. In the end, I'll pick a winner! If I were choosing a framework that I'm going to have to live with for a long time, I think there are several factors I would consider:

- Community—the framework is well liked and adopted
- Documentation—it's easy to support
- Learning curve—we can train new people
- Performance—the framework can meet my business needs

Community

A healthy community around a technology makes working with that technology so much easier. Given that, here are a few ways we might interpret community health:

- Age—we aren't worried about it going away
- Issue management—there are active maintainers
- Downloads—people are using it
- GitHub stars—people like it

Sails, Hapi, and Restify are all about the same age, coming to GitHub within a year of each other (between 2011 and 2012). Each framework pushes updates frequently, and, at the time of writing, all three of the frameworks have pushed updates in the last month.

Popularity shouldn't be the main driver of this sort of decision, but it's often a good indicator that a product is doing something right. This is where the frameworks currently stand in relation to each other in the popularity department:

FRAMEWORK	GITHUB STARS	WEEKLY DOWNLOADS
Hapi	10,948	242,723
Restify	9,150	81,227
Sails	20,378	29,195

At the time of this writing, Sails has 328 issues. Restify has 43. However, Hapi has an astoundingly low nine issues. The Hapi documentation website even has a page devoted to issue tracking! That's some amazing maintainer support for such a frequently downloaded framework. Hapi also has the most downloads. That's why I'm giving the overall win for community to Hapi.

Hapi: 1
Sails: 0
Restify: 0

Documentation

When looking at the documentation, I'm mostly concerned that there's a quick-start guide and some references that allow me to see example code. This isn't so much about seeing which is better, but rather about seeing if one is significantly worse.

Hapi (https://hapijs.com/tutorials) and Restify (http://restify.com/docs/home/) tick the boxes for a quick-start guide and examples. Hapi has tutorials directly on the site. Restify has some good examples too, but you have to dig for them a bit. They're in an examples folder in the Restify GitHub repository (https://github.com/restify/node-restify/tree/master/examples).

Sails seems to have a great documentation website. However, the quick-start guide isn't as simple because it involves bootstrapping an entire app with a CLI tool. To counter some of the added complexity of the quick-start, the Sails team provides a repository with a non-

trivial sample application written with Sails and Vue.js. Having a sample app for reference is a big plus, which helps make up for the relatively complicated quick-start.

Without using the documentation on a daily basis, it's difficult to tell how good it is in reality. However, the good news is that there are no red flags for any of the frameworks, and thus no clear winners in this department.

Hapi: 2
Sails: 1
Restify: 1

Learning curve

In an effort to get a rough idea about these frameworks' learning curves, I created a project with each one. I wanted to see how quickly I could get going with a simple use case, such as creating an endpoint that returns some data as JSON. This was extremely easy to do both with Restify and Hapi. Sails, however, was another story.

Hapi

First, I implemented a simple GET call in Hapi to return a list of awesome Blizzard games.

```
"use strict";

const Hapi = require("hapi");

const server = Hapi.server({
    host: "localhost",
    port: 8000
});

server.route({
    method: "GET",
    path: "/games",
    handler: (request, h) => {
        const games = [
            "World of Warcraft",
            "Diablo",
            "Starcraft",
            "Heroes of the Storm"
        ];
        return { games };
    }
});

const start = async () => {
    try {
        await server.start();
    } catch (error) {
        console.error(error);
        process.exit(1);
    }

    console.log("Server running at: ", server.info.uri);
};

start();
```

Restify

I did the same thing for Restify. Like Hapi, it was pretty straightforward.

```
const restify = require("restify");

function respond(request, response, next) {
    const games = [
        "World of Warcraft",
        "Diablo",
        "Starcraft",
        "Heroes of the Storm"
    ];

    response.send({ games });
    next();
}

const server = restify.createServer();
server.get("/games", respond);
server.listen(8000, () => {
    console.log("%s listening at %s", server.name, server.url);
});
```

Sails

Unfortunately, when I tried to return my precious games in Sails, the implementation was not very straightforward.

Sails has an opinionated view on how the app should be structured. This isn't necessarily good or bad, but it does add increased overhead when learning it for the first time. With these sorts of frameworks, other people have put in some effort toward creating a structure that allows developers to quickly add new functionality. Hopefully, that upfront learning cost translates into saving time later on, but that

doesn't change the fact that it's harder to learn initially. For Sails, I had to add an entry to the routes file that took the string value of an action in a controllers directory.

```
module.exports.routes = {
    "/games": "games"
};

module.exports = async function(req, res) {
    games = ["World of Warcraft", "Diablo", "Starcraft", "Heroes of
 the Storm"];
    return res.json({ games });
};
```

While this isn't a super complex example, it gives me an idea about what it would be like to implement something from scratch. Sails has already introduced an abstraction and corresponding implementation for me. This is confusing to me as a newcomer, and especially on simple projects, separating the actions from the routes in this way can be verbose and confusing. I worry that attempting to do other sorts of things in Sails, such as handling a database connection, will require knowledge of how the creators think apps should be set up. If their abstractions and mental model line up with yours, maybe that's a good thing. I thought this was confusing, but your mileage may vary.

We found out earlier that Restify is positioned as a web service framework focused on semantically correct RESTful services. It exposes optional middleware for modifying or taking actions on requests and responses. Hapi has a lot of plugins that can accomplish various things, but those features aren't included in the base package. Both Restify and Hapi have an edge over Sails in the learning department because the features are opt-in, and you can learn them one at a time.

Hapi: 3
Sails: 1
Restify: 2

Performance

I was able to find some benchmarks run in environments resembling what I might use for a production server. From late 2018, these tests (https://www.techempower.com/benchmarks/#section=data-r17&hw=cl&test=json) clearly indicate that Restify has a performance edge over Hapi and Sails for JSON serialization.

I also ran a simple performance test on my machine using Apache Bench and the same server endpoints I created during the learning curve section.

The results on my machine were somewhat similar to those from the benchmarking site. Sails seriously lags behind Hapi and Restify. Hapi performed better on my machine, though still not as well as Restify. Admittedly, the tests I ran on my machine are certainly not ideal, given that my environment isn't similar to a hosted environment. However, it's still a good indicator, especially considering how much the results varied. Restify wins here, though Hapi gets some points too.

It's great that we can run a simple performance test (https://stackify.com/fundamentals-web-application-performance-testing/) before we pick a framework. Although when we have customers and a business relying on it, using local benchmarking tools isn't a great substitute for application monitoring. Tools like Retrace (https://stackify.com/retrace/) can help you understand and improve your Node.js application's performance characteristics.

Hapi: 4
Sails: 1
Restify: 3

Restify vs Sails vs Hapi: Final scores and thoughts

To end my initial thought experiment of what I would do if tasked with picking a framework from these three, I think I would choose Hapi. It seems to stand out from the other two frameworks, specifically in regards to its community support.

Restify follows close behind because of how well it performs and how easy it is to start using it. Plus, if I didn't get too invested in Hapi's framework-specific plugins, it wouldn't be difficult to migrate my logic over to Restify. But when compared to Hapi, its community support doesn't match up.

Sails can bootstrap some impressive features right out of the gate, but it's less interesting to me because of the rigidity and the commitment to learning framework-specific nuances.

Assuming the community is flourishing and the performance meets my needs, I believe the ability to start quickly with a small piece of well-understood functionality is of utmost importance. The learning curve for Sails seems like a difficult pill to swallow in this regard. You could make the argument that it would scale better once you understand the paradigm, and you might be right. You have to make the best choice for you, your team, and your business. So the biggest takeaway is probably my initial statement: context matters!

(/retrace-error-monitoring/)

Error Tracking (https://stackify.com/retrace-error-monitoring/)

(/retrace-log-management/)

Centralized Logging (https://stackify.com/retrace-log-management/)

(/retrace-app-metrics/)

App & Server Metrics (https://stackify.com/retrace-app-metrics/)

Learn More
(/retrace/)

About the Author
()

About Chris Tobolski

This post was written by Chris Tobolski. Chris is a full stack software engineer at FordLabs who enthusiastically practices lean validation while building human-centered products. He loves motivating teams and individuals to strive to reach their potential through continuous improvement. And while he's a full stack engineer, he currently enjoys working with Typescript and React.

Sign Up Today

Search Blog

Search

Topics/Keywords

ASP.NET (https://stackify.com/?tag=asp.net,net-core)

.NET Core (https://stackify.com/content/net-core/)

Java (https://stackify.com/content/java/)

Azure (https://stackify.com/content/azure/)

AWS (https://stackify.com/content/AWS/)

Cloud (https://stackify.com/?tag=cloud,azure,aws)

Product Updates (https://stackify.com/stackify/)

App Monitoring (https://stackify.com/?tag=monitoring,apm)

App Performance Tips (https://stackify.com/?tag=performance,profiler,apm)

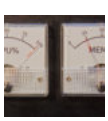
Error Handling (https://stackify.com/?tag=exception,exceptions,error,errors)

Logging Tips (https://stackify.com/?tag=logs,logging)

DevOps (https://stackify.com/content/DevOps/)

Popular Posts

ASP.NET Performance: 9 Types of Tools You Need to Know!



How to Troubleshoot IIS Worker Process (w3wp) High CPU Usage

How to Monitor IIS Performance: From the Basics to Advanced IIS Performance Monitoring

SQL Performance Tuning: 7 Practical Tips for Developers

Looking for New Relic Alternatives & Competitors? Learn Why Developers Pick Retrace

Recent Posts

OOP Concept for Beginners: What is...
(https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/)

Hiring a DevOps Engineer: Useful Ti...
(https://stackify.com/hiring-a-devops-engineer-useful-tips-and-best-practices/)

How To Ensure You Are Followi...
(https://stackify.com/how_to_ensure_you_are_following_best_coding_practices/)

OOP Concepts for Beginners: What is Polymorphism (https://stackify.com/oop-conce...
(https://stackify.com/oop-concept-polymorphism/)

A Full Guide on Using...
(https://stackify.com/a-full-guide-on-using-application-monitoring-for-your-business/)

(/stackify-developer-ebooks/)

(https://stackify.com/guest-blogging-guidelines/)

Get In Touch

Contact Us

Request a Demo

Start Free Trial

Products

Retrace

Prefix

.NET Monitoring

Java Monitoring

PHP Monitoring

Node.js Monitoring

Ruby Monitoring

Python Monitoring

Retrace vs New Relic

Retrace vs Application Insights

Solutions

Application Performance Management

Centralized Logging

Code Profiling

Error Tracking

Application & Server Monitoring

Real User Monitoring

For Developers

For DevOps

Resources

What is APM?

Pricing

Case Studies

Blog

Documentation

Free eBooks

Free Webinars

Videos

Ideas Portal

ROI Calculator

Support

News

Company

About Us

News

Careers

GDPR

Security Information

Terms & Conditions

Privacy Policy

PO Box 2159
Mission, KS 66201
816-888-5055 (tel:18168885055)