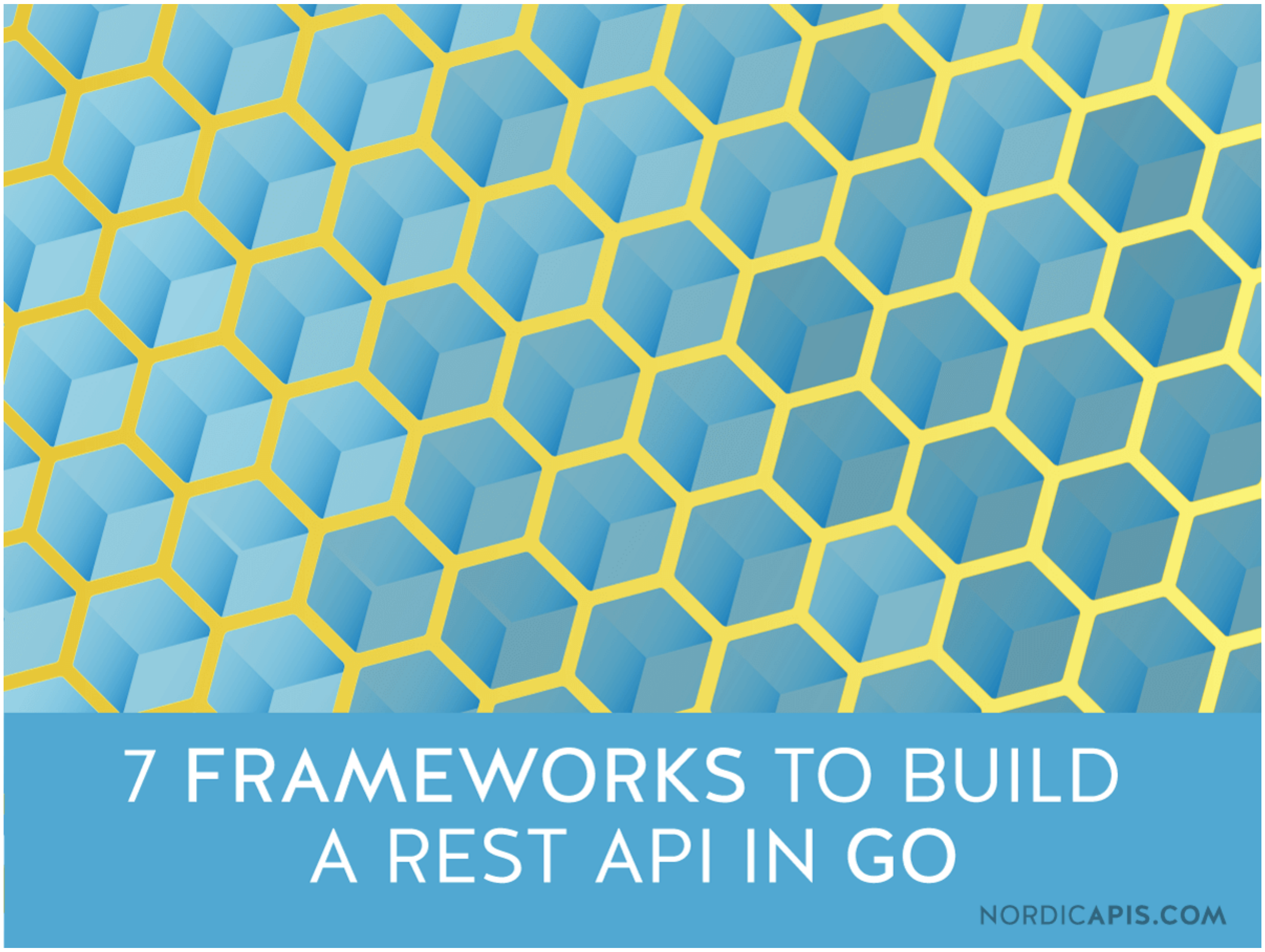
Blog

7 Frameworks To Build A REST API In Go (https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/)

POSTED BY KRISTOPHER SANDOVAL (HTTPS://NORDICAPIS.COM/AUTHOR/SANDOVALEFFECT/) | JULY 4, 2017

Updated on March 29th, 2018



We've previously talked about **Go** – it's a very powerful, efficient, and lean **language** that powers both enterprise and small group applications. While Go itself is very powerful, as with any language, additional functionality is often desired, or outright required.

Go Golang!

When those situations arise, having a solid **framework** with which to depend on is very important. Thankfully, Go has matured since its release, and boasts a wide range of amazing frameworks.

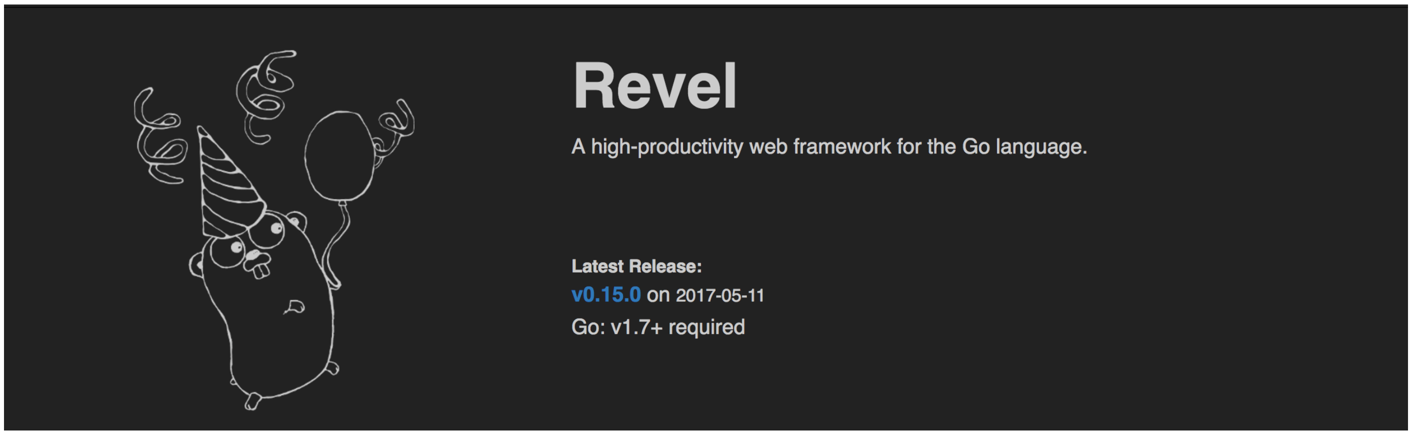
While "framework" is often confused for dependencies or extensions, in reality, a good framework is simply a **library** that provides support for a range of activities and services. Frameworks are often framed within the strictures of *lightweight* and *fully featured*, and when choosing your specific framework, you must consider not only what your API currently requires, but what you might want to implement in the future.

Today, we compare seven popular Go frameworks that can be used to build a REST API – we'll identify what differentiates them, and hopefully discover a framework that's right for your given application.

Also Read: Writing Microservices in Go (http://nordicapis.com/writing-microservices-in-go/)

1 – Revel (https://github.com/revel/revel)

A high-productivity web framework for the Go language.



(https://github.com/revel/revel)

Pros

Revel is first and foremost **fully featured**. Unlike some of the other frameworks on this list, Revel is designed to come out of the box with much of its feature-set pre-configured and installed for optimal functionality. This means that you can get going pretty much with no setup, which is very appealing for many startups and small groups.

Likewise, Revel doesn't require very many third party libraries or middleware implementations in order to do complex tasks, which, unlike some entries on this list, means that it's relatively **self contained**. By packaging everything together out of the box and ensuring even complex tasks can be done with the default installation, Revel seems to be positioning itself as a "one stop shop" solution.

This also comes with the noted quality of having a relatively reduced complexity for high functionality. While other frameworks can tie into third party distributions to enable the same levels of functionality, this means added complexity and requirements for third party dependencies.

Takeaway

- **Full Feature set**: Revel is fully featured out of the box, with packaged libraries and feature sets for everything from small to complex tasks.
- **Self-Contained**: By eschewing dependence on third party libraries, Revel works "out of the box", reducing complexity between interacting libraries and extensions.

Cons

The fact that Revel is fully featured could, for some, be a negative. While it's great that Revel is fully featured out of the box, this also means that the code base is much **larger** than other solutions, and thereby not very lean. This is an interesting quality when one looks at Go, which is intended to be lean for solutions from tiny to huge. While having everything preconfigured and prepackaged is great in terms of quick setup, it does mean that you lose some agility.

While this isn't a big issue for many, the lack of native support for MongoDB is. While this support can be implemented using third party solutions, it's something that one would expect with a "kitchen sink" implementation where everything else is packaged in – missing a key feature like this makes it a hard sell to some.

Takeaway

- **Larger codebase**: Revel is a "kitchen sink" implementation, including everything you need to get going. This means added framework weight and size, making it not lean.
- **No MongoDB**: The lack of MongoDB is an issue for many people utilizing the framework, and while this support can be added using third party implementations, it adds unnecessary work you would not expect from such an otherwise complete framework.

2 – Gin (https://gin-gonic.github.io/gin/)

The fastest full-featured web framework for Golang. Crystal clear.

Gin Gonic
The fastest full-featured web framework for Golang. Crystal clear.
GITHUB PAGE
API REFERENCE
$ go get github.com/gin-gonic/gin

Pros

Whereas solutions like Revel promise an all in one experience, Gin delivers a very **minimalistic**, trimmed down framework that carries with it only the most essential features, libraries, and functionalities. This makes Gin extremely **lean** – and this is really the huge selling point for frameworks like Gin.

This simple, succinct design ethos is mirrored in the documentation (https://gin-gonic.github.io/gin/), which is direct and effective. This makes Gin a great framework to start with, and can make for easier debugging and issue tracking.

Of note is that Gin was designed with **Martini**, another framework, in mind. By using httprouter (https://github.com/julienschmidt/httprouter) for its traffic handling, Gin managed to increase speed by over 40 times from Martini. This makes it a good choice between spartan code and speed.

Takeaway

- **Minimalist**: Gin is very minimalistic, including only essential features, libraries, and functionalities. This makes it extremely lean and fit for systems with low power.
- **Usable**: Very simple framework with decent documentation, so it's very easy to learn and debug.
- **Agile**: Extremely fast, especially compared to the genesis framework, Martini.

Cons

Gin is very spartan, and for many, that's a good thing. For enterprise solutions, however, Gin simply does not cut it. While you can in theory tie in third party implementations and other extensions, support for these solutions is not as robust as in other frameworks, and serves to negate much of what makes Gin a positive thing in the first place.

Likewise, Gin's limitations means that, even if you do extend the framework, you're moving a lot of the processing to the client. Thus, any limitations on the client side are going to be reflected on the server implementation and thus the actual functionality of the API.

The server can only do so much in Gin – and with that, you get limited functionality.

Takeaway

- **Not for Enterprises**: Gin is a very spartan framework, and thus is not suitable for applications that require a large backend or several complex server functions.
- **Big Client**: The framework tends of offload a lot of work to the client due to server limitations, limiting potential implementations to what you have.

We also reviewed 5 Lightweight *PHP* Frameworks to Build REST APIs (http://nordicapis.com/5-lightweight-php-frameworks-build-rest-apis/)

3 – Martini (https://github.com/olebedev/martini)

Classy web framework for Go

Pros

Martini is extremely **lean**, but unlike Gin, boasts impressive and easy to integrate third party support. These additional libraries take what is the otherwise incredibly lean Martini code base and allow it to magnify and leverage its functionality to be something more than just its components. This additional third party support makes Martini less of a ***framework*** like Gin, and more like an ***ecosystem***, like Revel.

That's the last of the similarities with Revel, however. Martini is very lean by design, and is meant to do a lot of work with a minimal amount of overhead. This means that it occupies a space somewhere between enterprise and small team programming, offering good **scalability**.

Martini also offers a wide range of support for routing methodologies and formats, and offers support for wildcards, variable parameters, regex stricture, and more. This makes it arguably more **powerful** than Gin for not much more overhead.

Finally, Martini has been in the game for a while – accordingly, there is ample documentation (https://github.com/go-martini/martini) and a large install base. If you have a Martini question, chances are it's been answered.

Takeaway

- **Lean**: Martini is lean, and has a great third party support base, making it modular and scalable.
- **Good docs**: Documentation for Martini is great, and the experience and age of the platform means most questions have documented answers.
- **Nice routing**: Routing under Martini is a joy – complex parameters and various data formats are supported.

Cons

Martini isn't all fun and games, though – there's a reason Gin was developed as a quasi-replacement. Martini doesn't handle traffic routing through httprouter like Gin does, meaning that it's **40x slower**, with an arguably more complex and heavy implementation of basically the same libraries and classes.

While the age of the platform is in one case a benefit for knowledge base and documentation, it's also a death knell for **versioning** – Martini has not (in its core implementation) been maintained since 2014. While additional permutations and forks are still maintained, these each have negatives and positives that represent their Martini core.

Finally, Martini does something that turns a lot of developers off of the framework – **dependency injection** as a methodology for handler discovery. Martini discovers handlers and passing methodology using a dependency injection scheme, which is fine for the intended purpose, but has the side effect of circumventing Go's typing system. Considering that the type system is a big reason for the adoption of Go, this is a definite negative.

Takeaway

- **Slower**: 40x slower than Gin.
- **Not actively maintained**: Standard Martini (not counting forks and mutations) hasn't been maintained since 2014. This lack of evolution could be a risk.
- **Dependency injection**: Martini finds out what your handlers are and the passing request methodology using dependency injection. This means that for many cases, it circumvents the typing system inherent in Go.

4 – Web.go (https://github.com/hoisie/web)

The easiest way to create web applications with Go

Pros

Web.go is unique in that it's a very **lightweight** framework that offers additional functionality over Go due to a **tree routing system**. This is a much more efficient system than simple list routing, as it allows for routing via relationships rather than purpose or usage. What this ultimately means is a much more lightweight, efficient, and easy to use framework that also boasts impressive gains in routing efficiencies.

It should be noted that Web.go is designed from the ground up to be extremely basic. Web.go is spartan by choice, and any additional features are likely to not change Web.go in any significant way in terms of the underlying codebase size or requirements.

Takeaway

- **Minimalistic**: Extremely minimalistic both in form and function. Like other minimalist implementations, Web.go is meant to be basic without much added to the framework.
- **Tree Routing**: Designed to route using trees rather than lists, delivering impressive efficiency gains in traffic routing. This makes the framework extremely low resource-cost.

Cons

Web.go is minimalist by design – but, just like other frameworks who attempt to make a lightweight implementation, this actually works against it in many ways. This is even more pronounced considering that Web.go is meant to be minimalistic by its very nature – there's precious little that Web.go does that Go cannot on its own.

Notably, what it does do differently – in this case, routing is the best example – can be done in Go with additional third party solutions that, while increasing complexity and codebase, offer more than a simple feature. The question as to whether or not path routing is worth the added complexity, however small, is an underlying current when considering Web.go.

Takeaway

- **Doesn't extend Go *that much*:** There's extremely little what Web.go does that the standard Go framework cannot do on its own. While the path tree routing system definitely is powerful, whether or not that's worth adding complexity to what is otherwise a basic Go implementation is questionable.

5 – Gorilla (http://www.gorillatoolkit.org/)

A web toolkit for the Go programming language | Repo (https://github.com/gorilla/)

Pros

Gorilla is a great example of scalability through **modularity**. Gorilla is designed to be able to drop packages (http://www.gorillatoolkit.org/pkg/), tie in new extensions, enable modules, and more, all without sacrificing the core functionality of the framework itself. For this reason, Gorilla is beloved by both enterprise and small scale groups, as it's the perfect example of **modular system scaling**, representing both ease and efficacy of such a situation.

Additionally, its native support of websockets (http://nordicapis.com/5-protocols-for-event-driven-api-architectures/) means that Gorilla is ready to go out of the box, with additional methodologies and approaches like HTTP routing, URL schemas, and arbitrary functions serving as additional routing mechanisms for both new and established web applications.

Takeaway

- **Websockets**: Supports websockets out of the box, which is hugely valuable.
- **Routing**: HTTP routing, URL schemes, and even arbitrary functions are likewise supported as a routing mechanism.
- **Scalable**: Gorilla can be scaled in either direction. You can drop packages, add packages, tie into new extensions, etc., without sacrificing the core Gorilla functionality.

Cons

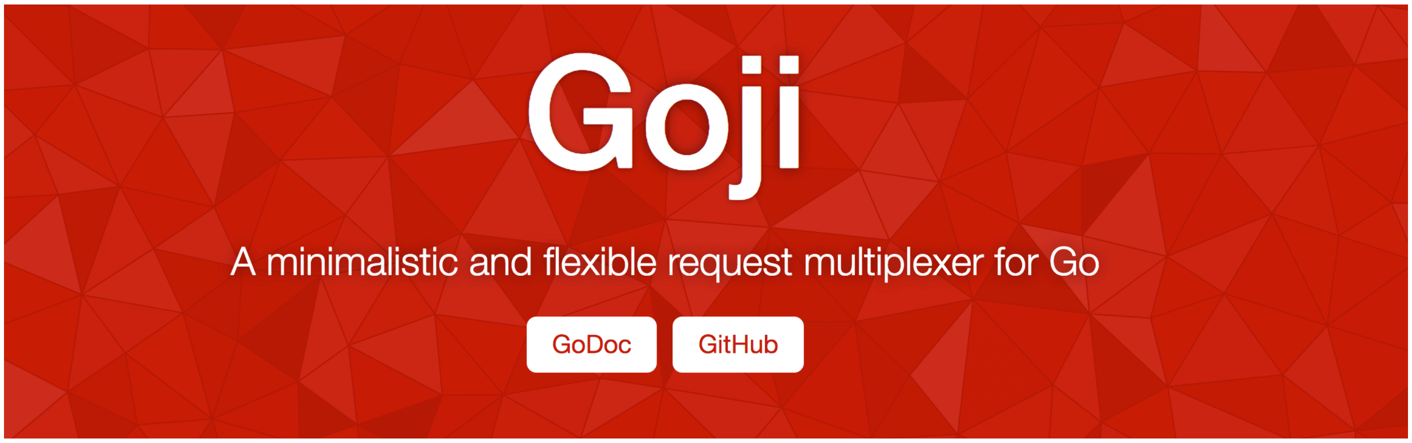
While the ability to be modular means you can get whatever you want, the effort put into **configuration** could mean a lot of wasted time to get functionality to where it needs to be. **Efficiency** also takes a hit in terms of out of the box configuration versus other frameworks. While this certainly can be fixed by dropping packages and adopting more efficient routing methods, other frameworks are much more efficient out of the box.

Takeaway

- **Configuration effort**: Gorilla requires more time and effort to configure modular components.
- **Performance**: There's a marked decrease in performance between out of the box Gorilla and other out of the box more "lean" solutions. This can of course be changed, but many developers creating a new API will first use Gorilla out of the box as a base, making it a serious consideration.

6 – Goji (https://github.com/zenazn/goji)

A minimalistic and flexible request multiplexer for Go | Repo (https://github.com/goji/goji)



Pros

Like Web.go, Goji is essentially a **layer** over the Go language. This means that it's a *framework through abstraction*, and is perhaps the smallest implementation possible. This lightweight and efficient processing makes it a good starting point for **lean** APIs.

Of note is that Goji comes integrated with **Einhorn**, a language agnostic socket manager that provides automatic, out of the box websocket support. As with Gorilla, this is hugely beneficial, and not a feature that would typically be assumed of such a small, lightweight framework.

These two elements make Goji a great choice for lightweight, fast, efficient, **low-resource** APIs that need to get a lot done with relatively low resources available.

Takeaway

- **Very thin**: Essentially an abstraction layer – very minimal in size, and so extremely lightweight and efficient.
- **Einhorn**: Integrates with Einhorn, a language agnostic socket manager that provides out of the box websocket support.

Cons

As with other lean implementations, unfortunately, there's not much that Goji does that Go does not. Because of this, it's very much a consideration between effort put forth to implement and the reward in terms of feature set. You can get much more for less effort with other implementations out of the box. That being said, the main draw here is **websockets** – and that might be enough to make this con a non-issue given certain implementation requirements.

Takeaway

- **Limited**: Essentially an abstraction layer – it doesn't do anything that Go doesn't already do.

7 – Beego (https://beego.me/)

Pros

Rounding out our selections for this piece, Beego is a fully featured framework that owes much of its development conceptualization and approach to Revel. While Beego is relatively lean for what it does, it is a "batteries included" framework, offering quite a **wide feature set** for what ultimately is not a big code base.

Part of this functionality is the awesome "**Bee tool**", a tool that checks for automated changes to the codebase and performs automated functions dependent on those changes. Bee tool can be configured to automatically build new revisions, archive function changes, etc.

Great functionality doesn't stop there, either. Beego has a great **ORM system** in place that allows for modeling relations and resources in a highly effective manner. This ultimately means **faster processing** and **better traffic routing**, which is always a welcome feature.

Takeaway

- **Full featured**: Just like Revel, Beego is more fully featured, and packs a lot into a relatively lean (for what it does) framework.
- **Bee tool**: "Bee tool" checks for automatic changes to the codebase, and is configurable to what it does. You can push a build with every change, have certain build archiving functions, etc.
  * Solid built-in ORM for application database organization.

Cons

Beego is relatively lean, which is good – from the ground up, the framework was designed to have a great amount of functionality without much overhead. Unfortunately, this design concept has some unintended consequences when it comes to **page caching**.

Beego caches content and pages in an effort to make for better user experience and decreasing the overhead that would otherwise be generated with constantly creating new representations and pages. While this is great for user experience, it does lead to **silent build failure**. Bee tool is great for versioning, but if a failed build is pushed, this is often transparent to both the user and developer by designing, meaning that, until a cache expires, your API may be rejecting an incredible amount of useful and valid traffic but presenting out of date data.

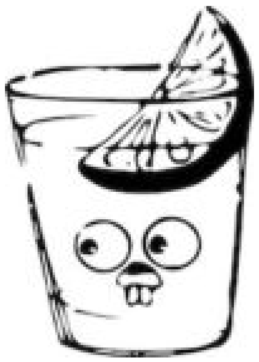
Beego is also rather immature. Because of this, its community is very small, meaning both third party plugins and extensions as well as documentation and **knowledge base** articles are sparse.

Finally, Beego, especially compared to some more lean implementations, is incredibly **verbose**, and promotes a sort of "chattiness" at the API level. This can be harmful to the health of related systems and in the efficacy of data delivery – while it can be negated using muting and batch processing, it's still something that requires yet more overhead to solve.

Takeaway

- **Caching**: Beego caches previous versions of pages, which is fine for user experience, but it can lead to silent build failure.
- **Chatty as a busy bee**: Extremely verbose, especially compared to more lean frameworks.
- **No community**: Very immature – lacks the kind of community Revel *revels* in.

In Summary…



As with anything, this is very much a consideration of what is best for your stated needs. Each of these frameworks bost specific pros and cons, and each of those pros and cons are very much opinion driven. You will find many who disregard the complexity topic in favor of abstraction, and still more who ignore verbosity as an issue and tout it as a feature – but regardless of where you stand on a given topic, you are likely to find some feature within that is beneficial to the point of adoption.

In summary, for enterprise solutions, **Revel** is still likely the strongest suggestion here – its strong community base and well-proven third party implementations make its shortcomings easier to deal with, especially when dealing with hardware that negates many of the processing overhead shortcomings.

Go treat yourself to a Gin *Gonic*.

For smaller solutions, something like **Beego** might be better, if chattiness is somehow reduced within your internal system. If you want a smaller but more mature system, either **Gin** or **Martini** are good choices. **Web.go** and **Gorilla** are good frameworks for those wanting a little more than Go, but not a fully featured out of the box framework. Finally, **Goji** is great for implementing websockets via Einhorn, especially in such a lean way – though, obviously, if you want more than just websockets, there are better choices that require slightly more overhead and effort.

Other Frameworks

We try to be comprehensive in our tech lists, but it looks like we may have forgotten some:

- Negroni (https://github.com/urfave/negroni): Golang framework, alternative to Martini
- Echo (https://echo.labstack.com/): High performance, extensible, minimalist Go web framework

Resources

- Go Framework Comparison (https://github.com/diyan/go-web-framework-comparsion)

60

Facebook (https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/&t=7+Frameworks+To+Build+A+REST+API+In+Go)

19 Twitter 7 Google+ (https://plus.google.com/share?url=https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/)

4

LinkedIn (https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=7+Frameworks+To+Build+A+REST+API+In+Go&url=https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/)

Reddit (http://reddit.com/submit?url=https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/&title=7+Frameworks+To+Build+A+REST+API+In+Go)

HackerNews (https://news.ycombinator.com/submitlink?u=https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/&t=7+Frameworks+To+Build+A+REST+API+In+Go)

Total: 90

7 (https://nordicapis.com/tag/7/), api (https://nordicapis.com/tag/api/), APIs (https://nordicapis.com/tag/apis/), Beego (https://nordicapis.com/tag/beego/), composability (https://nordicapis.com/tag/composability/), echo (https://nordicapis.com/tag/echo/), Einhorn (https://nordicapis.com/tag/einhorn/), framework (https://nordicapis.com/tag/framework/), frameworks (https://nordicapis.com/tag/frameworks/), Gin (https://nordicapis.com/tag/gin/), Go (https://nordicapis.com/tag/go/), Go language (https://nordicapis.com/tag/go-language/), Goji (https://nordicapis.com/tag/goji/), golang (https://nordicapis.com/tag/golang/), Gorilla (https://nordicapis.com/tag/gorilla/), Gorilla Toolkit (https://nordicapis.com/tag/gorilla-toolkit/), httprouter (https://nordicapis.com/tag/httprouter/), lean (https://nordicapis.com/tag/lean/), martini (https://nordicapis.com/tag/martini/), MongoDB (https://nordicapis.com/tag/mongodb/), negroni (https://nordicapis.com/tag/negroni/), open source (https://nordicapis.com/tag/open-source/), packages (https://nordicapis.com/tag/packages/), rest (https://nordicapis.com/tag/rest/), REST API (https://nordicapis.com/tag/rest-api/), Revel (https://nordicapis.com/tag/revel/), scalability (https://nordicapis.com/tag/scalability/), seven (https://nordicapis.com/tag/seven/), simplicity (https://nordicapis.com/tag/simplicity/), web (https://nordicapis.com/tag/web/), web API (https://nordicapis.com/tag/web-api/), web application (https://nordicapis.com/tag/web-application/), web applications (https://nordicapis.com/tag/web-applications/), web framework (https://nordicapis.com/tag/web-framework/), Web.go (https://nordicapis.com/tag/web-go/), websocket (https://nordicapis.com/tag/websocket/)

7 Comments (https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/#disqus_thread)

About Kristopher Sandoval

Kristopher is a web developer and author who writes on security and business. He has been writing articles for Nordic APIs since 2015.

(https://nordicapis.com/author/sandovaleffect/)
(https://www.linkedin.com/in/kristophersandoval/)

What Qualities Make a Great... (https://nordicapis.com/qualities-make-great-api-product-owner/)

Insights From the Stack... (https://nordicapis.com/insights-from-the-stack-overflow-2017-developer-survey/)

7 Comments **Nordic APIs** 1 Login

Recommend 3 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Diógenes A. Fernandes Hermínio • a year ago

Great post!

3 ^ | v • Reply • Share ›

Reza Rahnama-ye- Moqaddam • a year ago

You've forgotten about echo!
https://echo.labstack.com/
This one is great and much better than Beego or Gin in my opinion

1 ^ | v • Reply • Share ›

Jehan Musa • 3 months ago

but which one is actively being maintained?

^ | v • Reply • Share ›

Petrus Prinsloo • 3 months ago

This post needs Echo in it and some of the older, unmaintained frameworks removed. Echo is a great framework that works well. I tried it with a project and was quick to get going in and didn't give me any issues.

^ | v • Reply • Share ›

emrah • 8 months ago

This is really old information. Martini is not maintained because author said it has bad ideas in it. And He created new one called negroni. And echo framework should be on the list.

^ | v • Reply • Share ›

NordicAPIs Mod → emrah • 8 months ago

To be fair, the author did acknowledge Martini's lack of maintenance as a con. Thank you for suggesting Negroni & Echo, we've added them to the bottom of the article for now.

^ | v • Reply • Share ›

Trần Mỹ • 8 months ago

Thanks for the post

^ | v • Reply • Share ›

ALSO ON **NORDIC APIS**

Act Now Before The GDPR Deadline

2 comments • 9 months ago

Ben Virdee-Chapman — Great rubric here, Kristopher -- thanks for compiling for the API community.(FYI The source link under the Controllers and Processors quote …

What is the Richardson Maturity Model?

2 comments • 7 months ago

Alex — Just for a bit of fun, here's the Richardson Maturity Model represented in terms of Pizzas. https://dzone.com/articles/...

Using JSON-LD To Establish Semantic Linked Data

3 comments • a year ago

Bill C. Doerrfeld — Thanks for sharing Scott! I'm inserting the link to Hydra once more because the one above isn't going through - http://www.hydra-cg.com/

Tools to Make HATEOAS Compliance Easier

1 comment • 6 months ago

Erik Wilde — are you sure about ION being new? the github repo looks rather old, and the domain name ionwg.org linked from the repo seems to have become …

Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy

(https://nordicapis.com/?post_type=events&p=9354)

(https://www.youtube.com/playlist?list=PLd2MPdIXKO13xNW_IMASNsH6HgbdEAqpI)

(https://nordicapis.com/?post_type=events&p=9568)

SMARTER TECH DECISIONS USING APIS

Subscribe to our mailing list

Subscribe

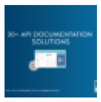
POPULAR POSTS

(https://nordicapis.com/3-common-methods-api-authentication-explained/) 3 Common Methods of API Authentication Explained (https://nordicapis.com/3-common-methods-api-authentication-explained/)

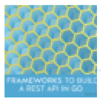
by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on February 6, 2018

(https://nordicapis.com/best-practices-api-error-handling/) Best Practices for API Error Handling (https://nordicapis.com/best-practices-api-error-handling/)

by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on June 15, 2017


(https://nordicapis.com/ultimate-guide-to-30-api-documentation-solutions/) Ultimate Guide to 30+ API Documentation Solutions (https://nordicapis.com/ultimate-guide-to-30-api-documentation-solutions/)

by Bill Doerrfeld (https://nordicapis.com/author/billdoerrfeld/) | posted on November 30, 2016


7 Frameworks To Build A REST API In Go

by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on July 4, 2017

(https://nordicapis.com/13-node-js-frameworks-to-build-web-apis/) 13 Node.js Frameworks to Build Web APIs (https://nordicapis.com/13-node-js-frameworks-to-build-web-apis/)

by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on October 26, 2017

Search

Search

RECENT POSTS

Simple Rules to Make Support Engineering Painless (https://nordicapis.com/simple-rules-to-make-support-engineering-painless/)

7 Types Of API Business Models (https://nordicapis.com/7-types-of-api-business-models/)

Why APIs Are Vital to Advancing the Connected Car (https://nordicapis.com/why-apis-are-vital-to-advancing-the-connected-car/)

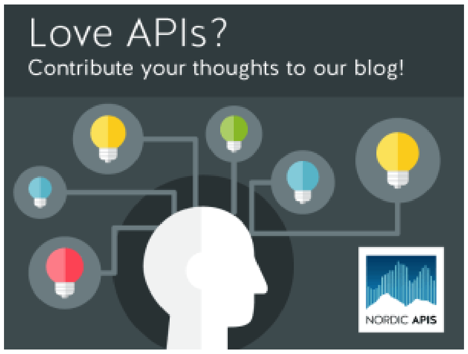
Embracing Diversity In The API Space (https://nordicapis.com/embracing-diversity-in-the-api-space/)

Bring on the Players: Who Wins in Open Banking? (https://nordicapis.com/bring-on-the-players-who-wins-in-open-banking/)

SUBSCRIBE TO OUR FEED

Nordic APIs RSS (http://nordicapis.com/feed/)

CREATE WITH US



(https://docs.google.com/a/twobotechnologies.com/forms/d/12Ng9A_QKUjmAHDgv8Pxb4uLIkECGJawV3vwAWJ4WxTs/viewform)

TWITTER (HTTPS://TWITTER.COM/NORDICAPIS) FACEBOOK (HTTPS://WWW.FACEBOOK.COM/NORDICAPIS)

YOUTUBE (HTTPS://WWW.YOUTUBE.COM/USER/NORDICAPIS)

SLIDESHARE (HTTP://WWW.SLIDESHARE.NET/NORDICAPIS)

INSTAGRAM (HTTPS://WWW.INSTAGRAM.COM/NORDICAPIS/) RSS (HTTP://NORDICAPIS.COM/FEED/)