Security and Real World HTTP Servers

LIGHTHOUSE LABS

AGENDA

- Encrypted Cookies
- Password Storage
- HTTPS
- REST

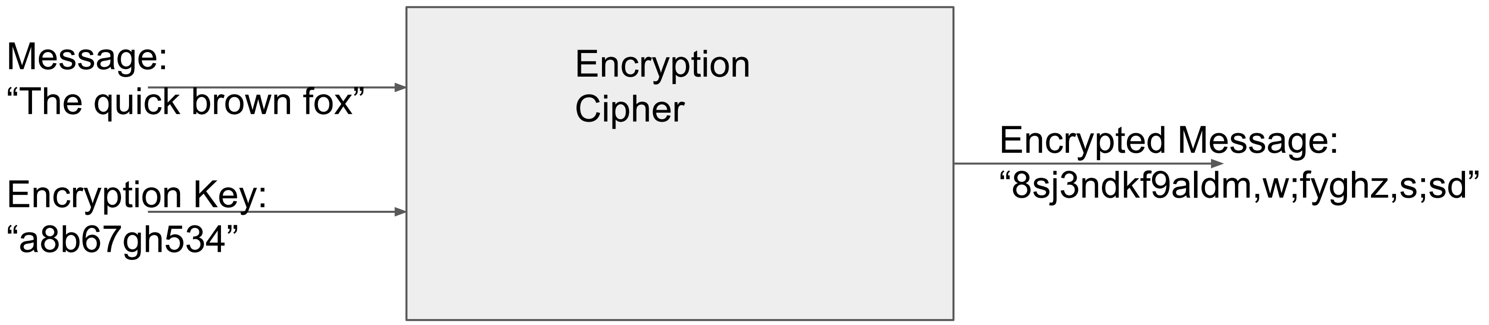
LIGHTHOUSE LABS

Encrypted & Signed Cookies

LIGHTHOUSE LABS

Encryption

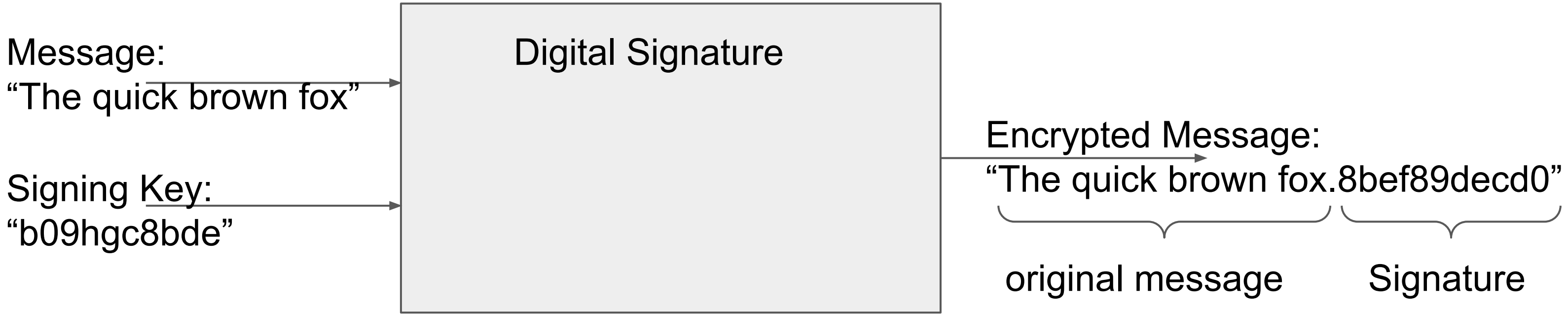
- Encryption is the act of Digitally Concealing the contents of a message
- Only the parties with the original **encryption key** can decrypt the message
- An encrypted cookie can only be read and modified by the server that created the cookie



LIGHTHOUSE LABS

Signing

- Signing is the act of Digitally Signing the contents of a message
- Anyone can read the message
- Anyone can validate that the message has not been changed
- Only the signing party can change the message



Encryption and Signing

	Encryption	Signing
Data Unchanged	✓	✓
Data Readable	✓	

Cookies are insecure

- By default, cookies are neither signed or encrypted
- Cookies are stored in the user’s browser - and are user modifiable
- By default **cookies are insecure**
- It is our responsibility to protect our Users by securing our cookies
    - Use plain cookies for insecure data you want to share with Browser
    - Use encrypted or signed cookies by default
    - Store as little user data in the cookie as possible

Best Practices

LIGHTHOUSE LABS

Session Spoofing

Demo

Storing Passwords

LIGHTHOUSE LABS

Secure Password Storage Is Hard

- Security is complex
- Security is only as good as the weakest link in the chain
- The goals of password security are unintuitive

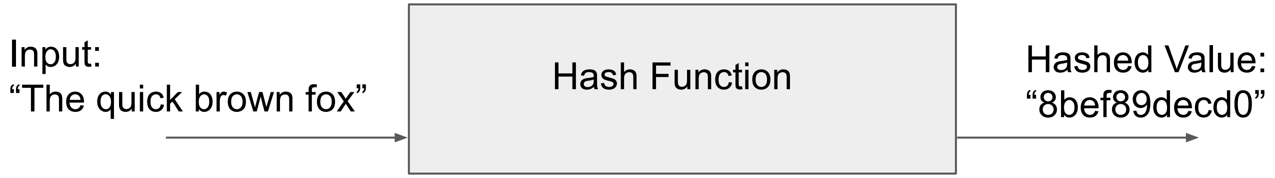
Goals of Password Storage

- Even if our entire database of user information is leaked
  - Hackers should not be able to determine our User’s Passwords
  - Hackers should not be able to determine if two Users have the same password

Aside: Hash Functions

Hash Functions

- Most important primitive security operation
- Deceptively simple
- 1-way functions where the input is unguessable based on the output
- The hashed value of an input is a fingerprint of the input value
    - A given value **always** hashes to the same hashed value
    - No two inputs hash to the same hashed value (mostly)



LIGHTHOUSE LABS

Store Password Hashes

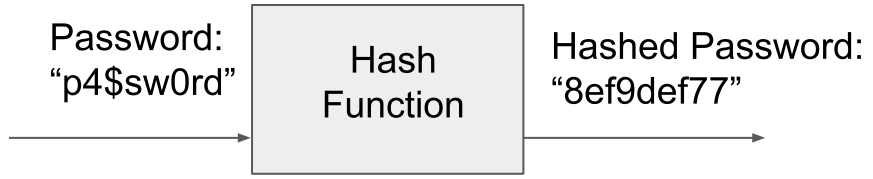
User Id	~~Plain Text Password~~	Hashed Password
1	~~lawrence~~	a89fgh5
2	~~password~~	b73he5
3	~~str4wb3rri3s~~	g87ha8
4	~~password~~	b73he5
5	~~passw0rd~~	a598ef

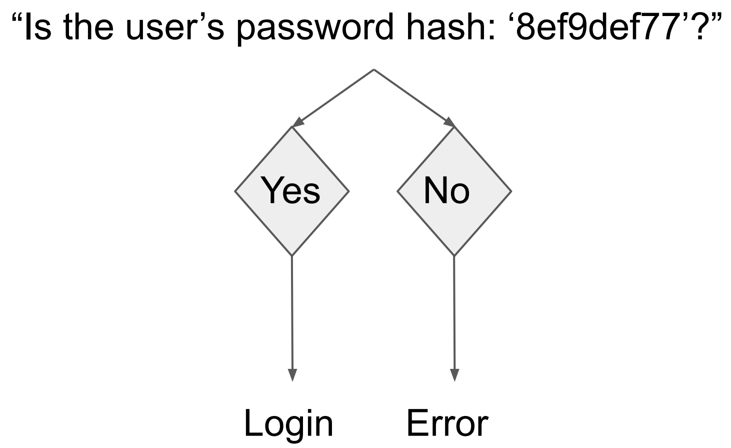
If we don’t know the user’s password, we can’t leak it.

LIGHTHOUSE LABS

Store Password Hashes

- Never store a users password directly
- Instead store the password hash
- On login, hash the entered password to compare against the stored value





LIGHTHOUSE LABS

User’s Love Bad Passwords

User Id	Plain Text Password	Hashed Password
1	lawrence	a89fgh5
2	password	b73he5
3	str4wb3rri3s	g87ha8
4	password	b73he5
5	passw0rd	a598ef

User’s with the same password have the same password hash

LIGHTHOUSE LABS

Password Salting

User Id	Plain Text Password	Salt	Password + Salt	Hashed Pass+Salt
1	lawrence	76as	lawrence:76as	a65gh
2	password	12bg	password:12bg	h153f
3	str4wb3rri3s	76ec	str4wb3rri3s:76ec	3gha5
4	password	ce88	password:ce88	a152f
5	passw0rd	g00g	passw0rd:g00g	gh596a

LIGHTHOUSE LABS

HTTPS

HTTPS

- HTTPS is a security layer that wraps the HTTProtocol with encryption
- TLS (Transport Layer Security) is the tech used to do encryption
- Encryption is asymmetric using public/private key exchange
  - Public Key is freely shared
  - Private Key is closely guarded secret
  - Messages are Encrypted with the Public Key and decrypted with the Private Key
  - Only the Receiver can decrypt messages, because the decryption key is private
- All requests, metadata, and data are encrypted under HTTPS

REST

LIGHTHOUSE LABS

REST

- REST is **RE**presentational **S**tate **T**ransfer
- REST is a way of structuring CRUD
- REST allows users of your API/website to be able to predict the API’s structure
- REST allows you to avoid having to think about what your paths should be

REST

REST allows us to map the CRUD operations to a specific method and path

Operation	**Path**	**Method**
Create	/<resource>	POST
Read	/<resource>/:id	GET
Update	/<resource>/:id	PUT/PATCH
Delete	/<resource>/:id	DELETE

LIGHTHOUSE LABS

Library Example

Our Library can do all CRUD operations on Books

Operation	**Path**	**Method**
Create	/books	POST
Read	/books/:id	GET
Update	/books/:id	PUT/PATCH
Delete	/books/:id	DELETE

Implicit Paths (For non API)

Our Library can do all CRUD operations on Books

Operation	**Path**	**Method**
Index	/books	GET
New (Form)	/books/new	GET
Create	/books	POST
Read	/books/:id	GET
Edit (Form)	/books/:id/edit	GET
Update	/books/:id	PUT/PATCH
Delete	/books/:id	DELETE

LIGHTHOUSE LABS

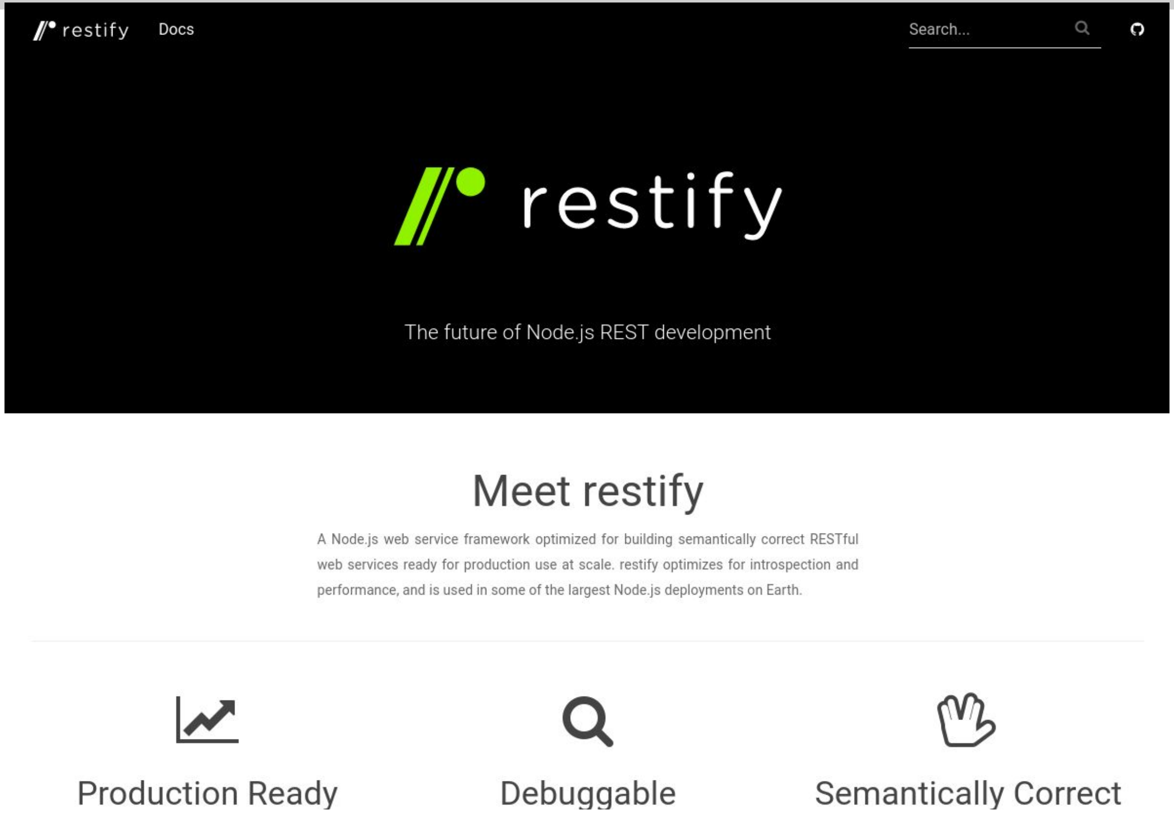
POST, PUT or PATCH

Under REST:

- POST is the general purpose write method - used for Creating new resources
- PUT is the write methods when you are updating all aspects of a resource
- PATCH is the write method used when you are only updating certain properties of a resource

Express.js Alternatives

Restify JS



The future of Node.js REST development

Meet restify

A Node.js web service framework optimized for building semantically correct RESTful web services ready for production use at scale. restify optimizes for introspection and performance, and is used in some of the largest Node.js deployments on Earth.

Production Ready

Debuggable

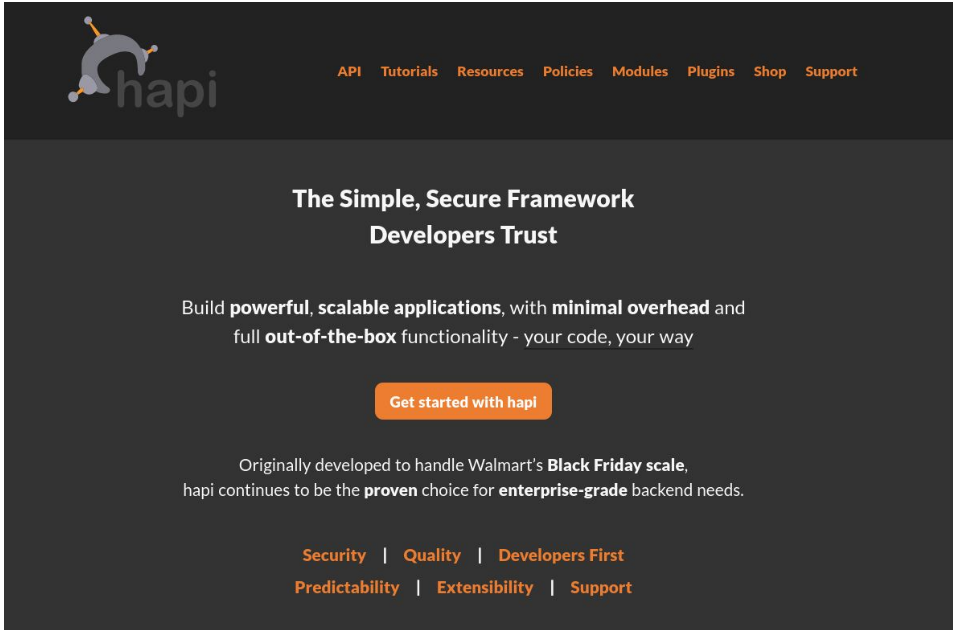
Semantically Correct

LIGHTHOUSE LABS

Koa JS

next generation web framework for node.js

Hapi JS


hapi
API Tutorials Resources Policies Modules Plugins Shop Support

The Simple, Secure Framework Developers Trust

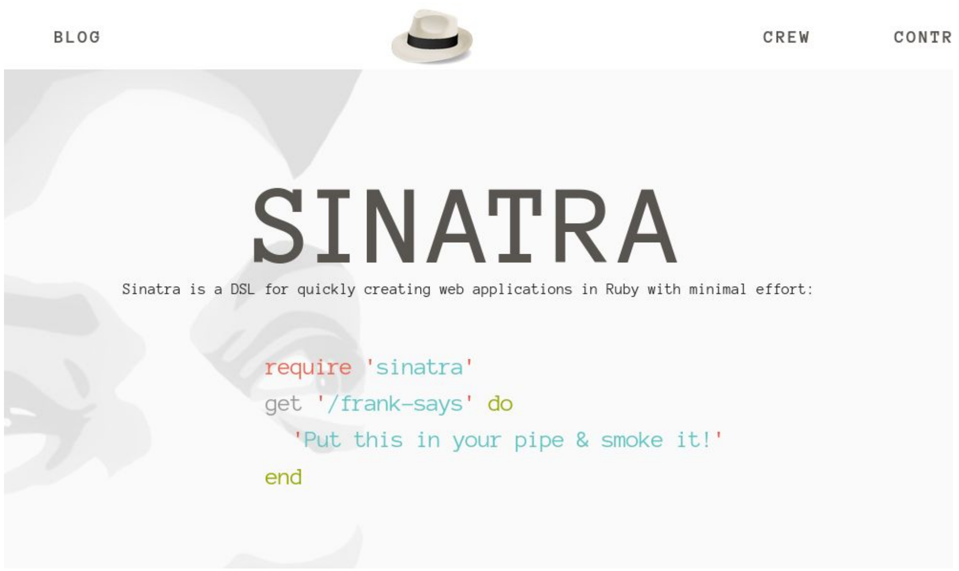
Build **powerful**, **scalable applications**, with **minimal overhead** and full **out-of-the-box** functionality - your code, your way

Originally developed to handle Walmart's **Black Friday scale**,
hapi continues to be the **proven** choice for **enterprise-grade** backend needs.

Security | Quality | Developers First
Predictability | Extensibility | Support

LIGHTHOUSE LABS

Sinatra Ruby

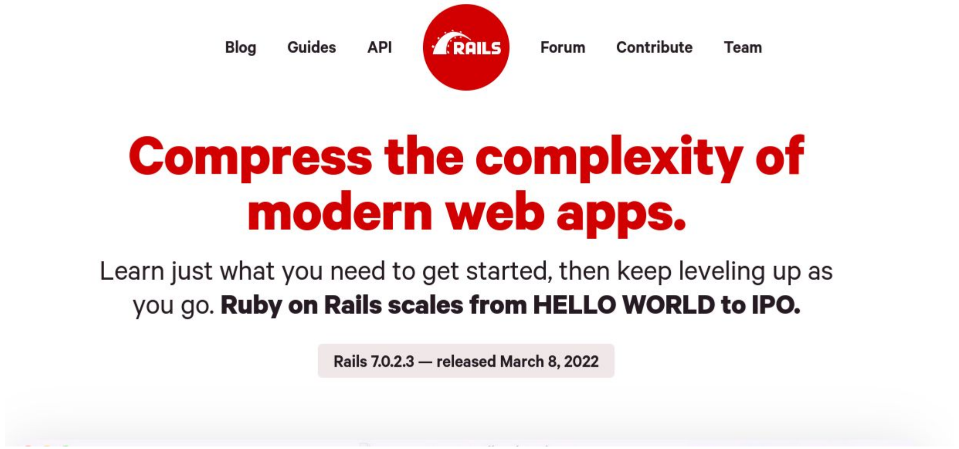


SINATRA

Sinatra is a DSL for quickly creating web applications in Ruby with minimal effort:

```
require 'sinatra'
get '/frank-says' do
  'Put this in your pipe & smoke it!'
end
```

RoR Ruby

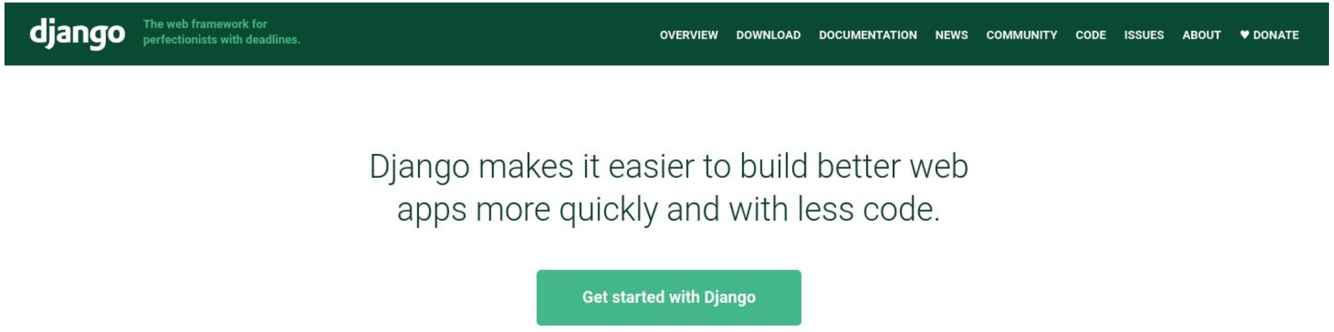


Compress the complexity of modern web apps.

Learn just what you need to get started, then keep leveling up as you go. **Ruby on Rails scales from HELLO WORLD to IPO.**

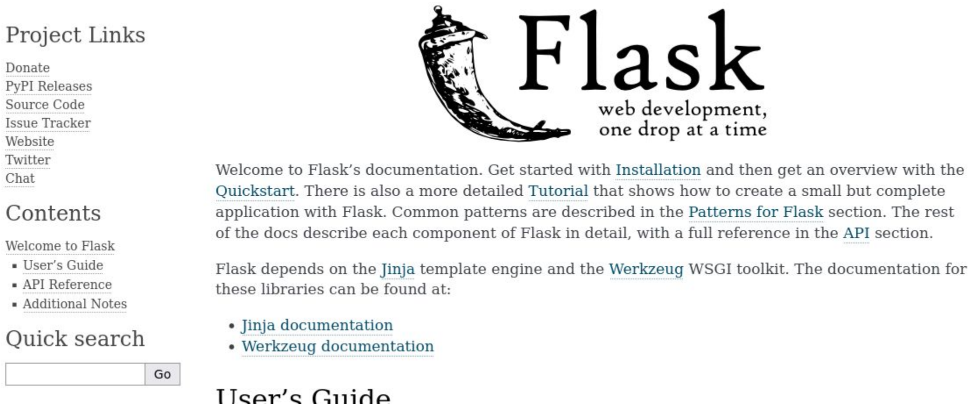
Rails 7.0.2.3 — released March 8, 2022

Django Python



Django makes it easier to build better web apps more quickly and with less code.

Flask Python



Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Website
Twitter
Chat

Contents

Welcome to Flask
- User's Guide
- API Reference
- Additional Notes

Quick search

Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs describe each component of Flask in detail, with a full reference in the API section.

Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. The documentation for these libraries can be found at:

- Jinja documentation
- Werkzeug documentation

User's Guide

RESTful Routing in express.js

Demo

Questions?

LIGHTHOUSE LABS