Takagi - Framework for CoAP

Dominik Matoulek

Who I am?

- Ruby developer
- Edge computing enthusiast
- Builds sensor boxes

Everyone knows HTTP

- What about IoT?
- Small devices does not have enough RAM.
- Headers, parsing

- Solution? CoAP! (RFC 7252)

CoAP is totally different, right?

Method	CoAP	HTTP
GET	✅	✅
POST	✅	✅
PUT	✅	✅
DELETE	✅	✅
PATCH	Nope	✅
HEAD	Nope	✅

So I can make a REST?

CoAP	HTTP
2.01	201
4.00	400
4.04	404
5.00	500

How hard it can be?

```
CoapServer server = new CoapServer();
server.add(new CoapResource("hello") {
  public void handleGET(CoapExchange exchange) {
    exchange.respond("Hello world!");
  }
});
server.start();
```

It looks brilliant, but I like this

```
@RestController

public class RestController {

    @GetMapping(value = "/hello")

    public Student getTestData() {

        return “hello”;

    }

}
```

What else?

Feature	CoAP libraries	“Web” Frameworks
Dynamic routing	DIY	✅
Modular	Just library, other DYI	✅ (usually)
Parameters	Somewhere	✅
Approach	Low-level	High-level

The result is Takagi

- Inspired by Sinatra (That’s why Takagi)
- Written in Ruby
- What if CoAP is just HTTP?

- https://github.com/domitea/takagi

Little detour: Controllers in Sinatra

```
require 'sinatra'


class FrankApp < Sinatra:Base

  get '/frank-says' do

    'Put this in your pipe & smoke it!'
  end

end



FrankApp.run!
```

In Takagi

```
require 'takagi'

class SensorAPI < Takagi::Base
  get "/sensor/:id" do |params|
    puts params[:id].to_i
  end
end

SensorAPI.run!
```

So we have REST, what next?

- CoAP is not just HTTP
- CoAP is also MQTT

- Meet RFC 7641 - Observing Resources in the Constrained Application Protocol (CoAP)

Wait a minute...

	MQTT	CoAP (RFC 7641)
Subscribe	SUBSCRIBE topic	GET + Observe
Publish	PUBLISH topic	NOTIFY
QoS	defined	CON/NON

And Takagi?

```
 reactor do
       observable "/sensors/temp" do # this endpoint can be observerved in CoAP way
         { temp: 42.0 }
       end
       observe "coap://temp_server/temp" do |params| # you can also observe another CoAP endpoints
         puts params.inspect
       end
  end
end
```

And how I can define...?

- Models? You can use Sequel
- Views? Well, it’s CoAP, not HTTP!
- HTTP stuff? You can use… Sinatra!

Why Takagi?

- Human-friendly CoAP
- Sinatra-like routing
- MVC like structure (if you add models)
- Runs where Ruby runs
- REST and MQTT combo
- Focus on sensors, not boilerplate

- Every Backend developer can do CoAP with Takagi

Questions?

Thank you for your attention