

Platformatic

View online

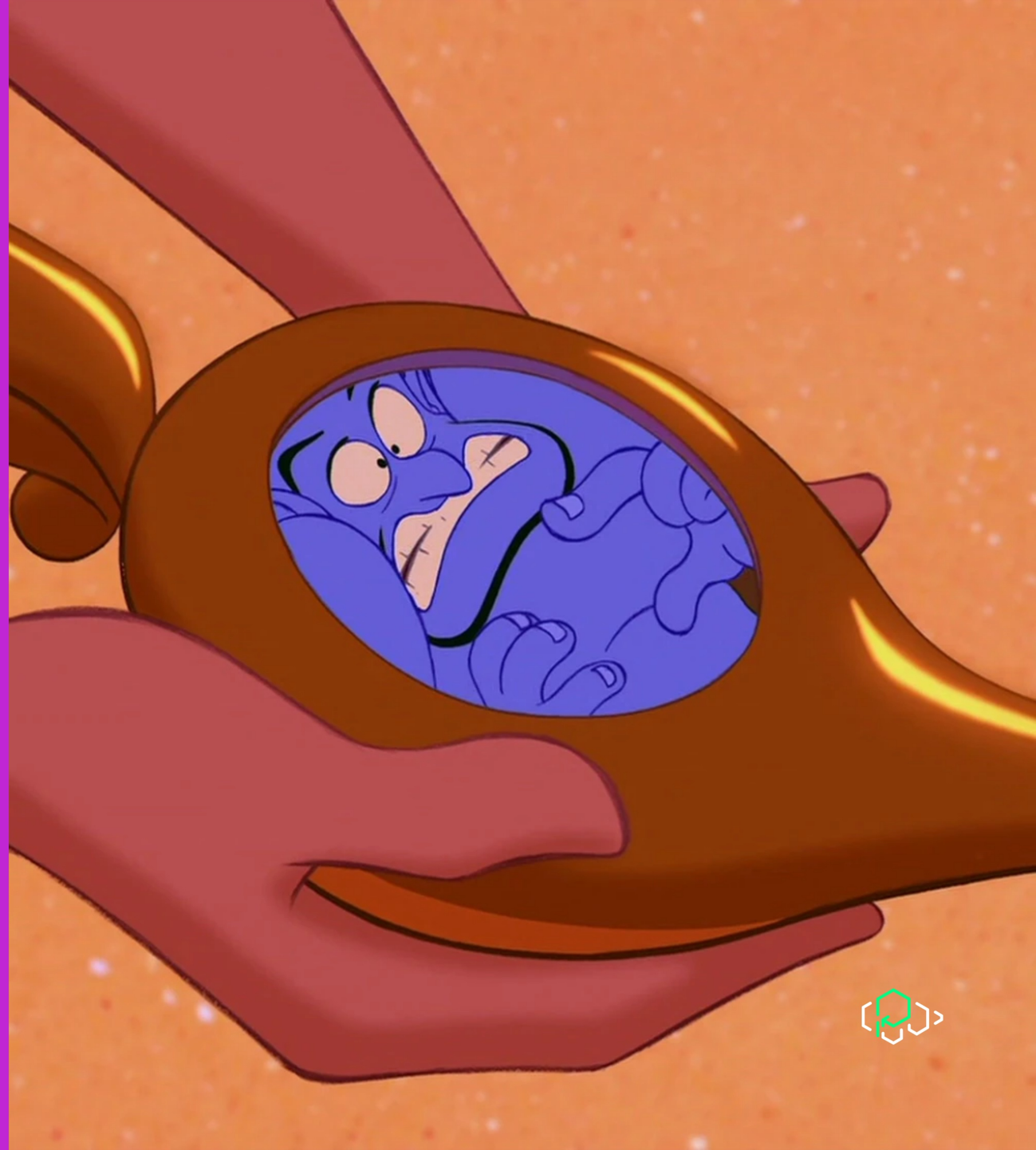
Download PDF

Welcome to the QRverse: let's build a rendering service

Paolo Insogna

Node.js TSC, Principal Engineer

Phenomenal cosmic powers... Itty bitty living space.



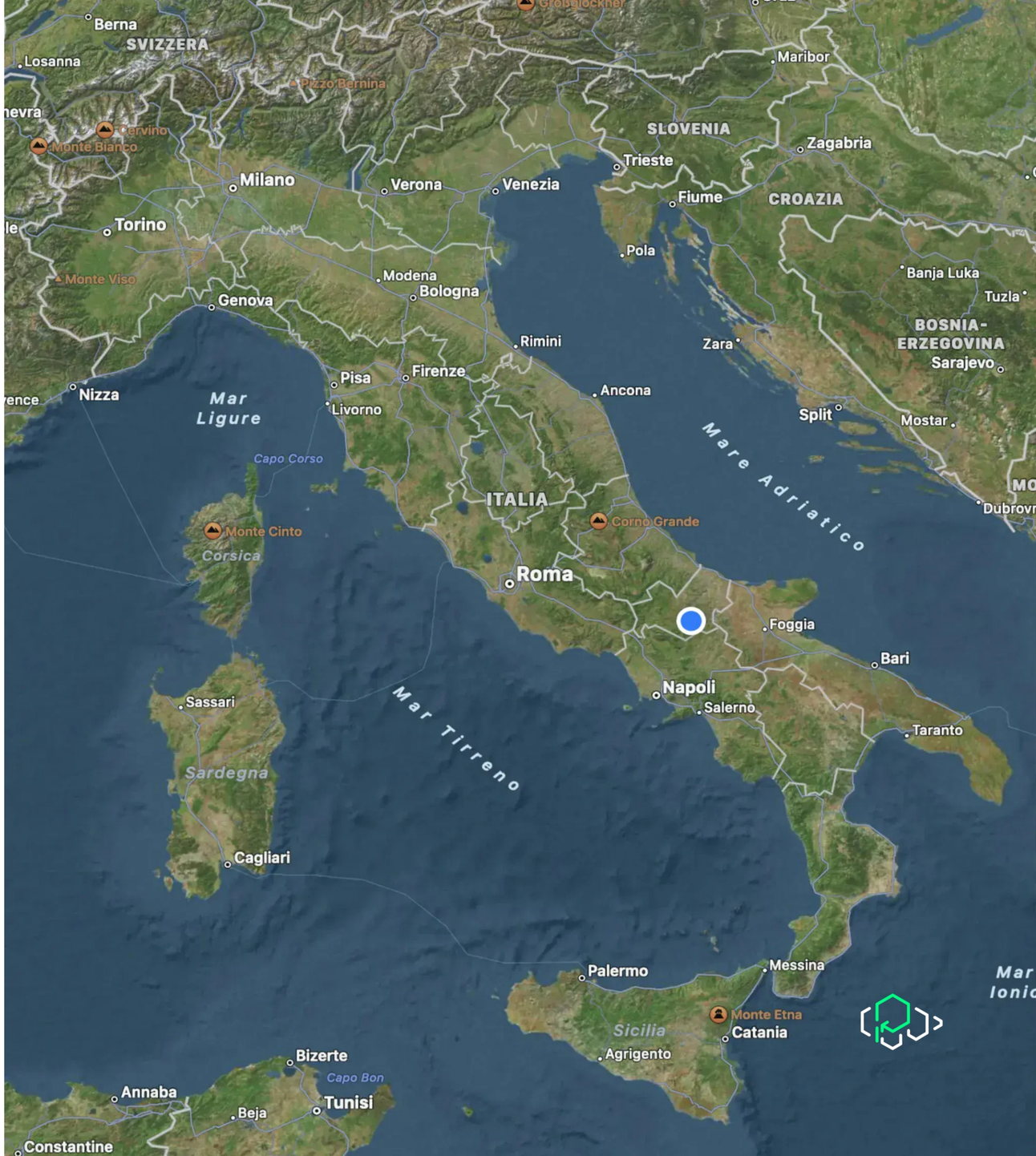
Hello, I'm **Paolo**!



Node.js Technical Steering Committee Member

Platformatic Principal Engineer

`paoloinsogna.dev` `ShogunPanda` `p_insogna` `pinsogna`



Some companies trusting us

Booking.com

Article

supabase

Case Study

SPENDESK

Case Study

Have you ever heard of QR codes?



Yes, of course!



They are everywhere



Easy to use

Static codes can be printed, while dynamic codes can be generated with low effort.



Lot of space

Each code can contain up to **7089** numbers, **4296** characters, **2953** bytes or **1817** kanjis.

History of QR Codes



Created in Japan in 1994

Denso Wave invented it to carry more data in a single code for its automotive parts.



Over 30 years of standards

Several revision of the **ISO 18004** standard were created to ensure compliancy.



Far beyond the original use case

QR are currently used in many aspect of our life, from official uses to games.

Main characteristics



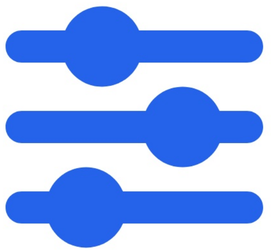
2-dimension code

The size and orientation are detected at scan time.



Multiple modes

The data is encoded by choosing between 4 different modes.



Configurable resiliency

Four error thresholds that can recover up to **30%** damaged block.

QR Codes make our life way easier ...



... sometimes in a weird way!



If we look closely...



Position markers (finders)

They are positioned in three corners to help the scanner determine the boundaries.



Alignment markers

They are complementary to the position markers to determine the orientation.



Timing markers

They are used from the scanner to understand the size.



Version information

They specify which of the many QR version has been chosen.



Format information

Similarly, they specify the encoding chosen.



Data and Error correction

The rest of the code delivers the data and the error correction (ECC), one next to each other.



Quiet zone

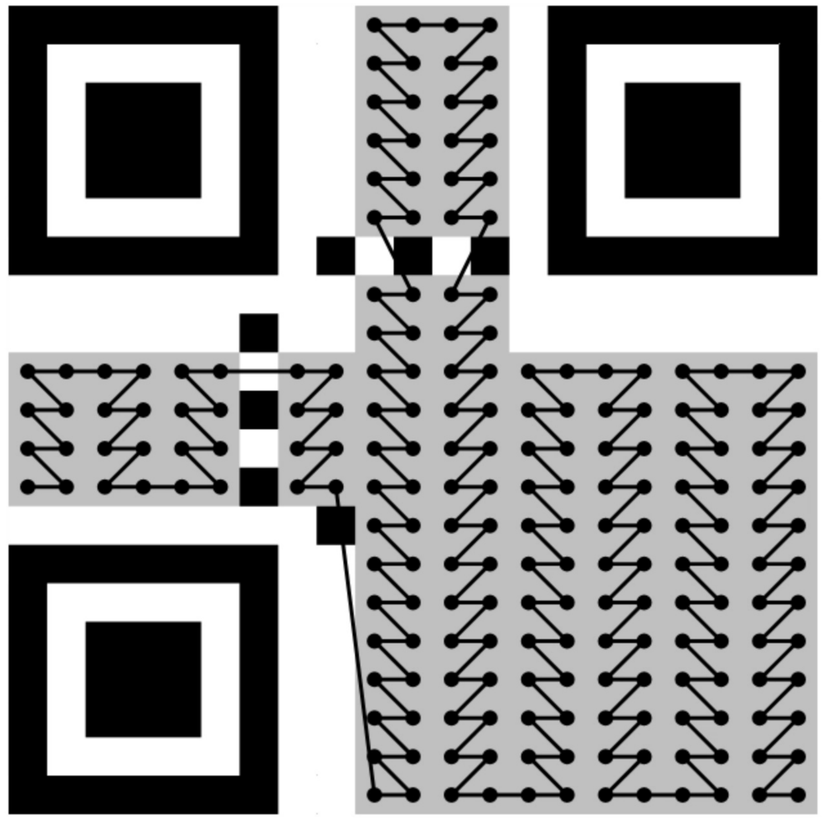
To help the scanner identify a QR code, it is required to pad the code in all directions.



The best path might not be the straight one

The data and ECC sections in a code are filled from the bottom right to the top left.

Each sequence of 8 bits is written in block divided in two columns. When encountering alignments, rows are skipped.



Can we get to the practice please?

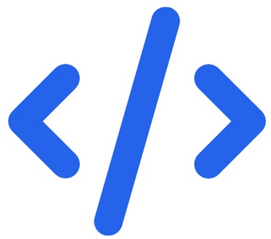


Let's build a QR rendering service



QR codes are easy to draw

Everything is well documented in ISO 18004 standard.



(Almost) no libraries needed

Once you know which color each cell is, you're good to go.



Suited for backends

Let's prove it with the best one out there, **Watt**.



Let's go!



Create the skeleton

We can use **create-platformatic**, it is that simple!

```
shogun@panda:~/qr$ pnpx wattpm init
[12:28:13.861] DONE (63654): Created a watt application in /Volumes/DATI/Users/Shogun/qr.
```



Database service - Create a migration

All the data schema is managed via migrations.

```
CREATE TABLE IF NOT EXISTS urls (
  id VARCHAR(36) PRIMARY KEY,
  url TEXT NOT NULL,
  count INTEGER DEFAULT 0
);
```



Database service - Track our clicks

```
1   import sql from '@databases/sql'
2
3   export default async function (app) {
4     app.put(
5       '/urls/:id/track',
6       { // ... schema definition ... },
7       async function (request) {
8         const id = request.params.id
9
10        await app.platformatic.db.query(
11          sql`UPDATE urls SET count = count + 1 WHERE id = ${id}`
12        )
13
14        const matching = await app.platformatic.entities.url.find({
15          where: { id: { eq: id } }
16        })
17
18        return matching[0]
19      }
20    )
21  }
```



URL service - Shorten a URL

```
server.post(
  "/",
  { // ... schema definition ... },
  async function (request, reply) {
    const rootUrl = `http://${request.headers['x-forwarded-host']}`
    const response = await fetch('http://database.plt.local/urls', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ id: randomUUID(), url: request.body.url })
    })

    // ... Response error handling ...

    const { id } = await response.json()
    const url = new URL(`/urls/${id}`, rootUrl)
    const url64 = Buffer.from(url.toString()).toString('base64url')
    const qr = new URL(`/qr/${url64}`, rootUrl)

    return { url: qr.toString() }
  }
)
```



URL service - Track clicks

```
server.get(
  "/:id",
  { // ... schema definition ... },
  async function (request, reply) {
    const response = await fetch(
      `http://database.plt.local/urls/${request.params.id}/track`,
      { method: 'PUT' }
    )

    if (response.status !== 200) {
      const error = await response.text()
      reply.status(response.status)
      return error
    }

    const url = await response.json()
    reply.header('x-clicks', url.count)
    reply.redirect(url.url, 301)
  }
)
```



URL service - Get all stats

```
server.get('/stats', async function (_, reply) {
  const response = await fetch('http://database.plt.local/urls')

  if (response.status !== 200) {
    const error = await response.text()
    reply.status(response.status)
    return error
  }

  const urls = await response.json()
  return urls
})
```



Frontend - Add a rendering UI

The frontend file uses **Preact without JSX**. No transpilation needed.

```
import fastifyStatic from '@fastify/static'
import fastify from 'fastify'
import { fileURLToPath } from 'node:url'

export async function create() {
  const server = fastify({ logger: true })
  await server.register(
    fastifyStatic,
    { root: fileURLToPath(new URL('./public', import.meta.url)) }
  )

  return server
}
```



Where are the QR codes?



Let's dive in!



QR service - Some helpers (1/4)

```
1  export function isFinder(modules, y, x) {
2    return (
3      (y < 7 && (x < 7 || x > modules - 8)) || // Top corners
4      (y > modules - 8 && x < 7) // Bottom left corner
5    )
6  }
```

1

QR service - Some helpers (2/4)

```
export function getAlignments(modules) {
  const version = (modules - 17) / 4
  const aligments = []

  const currentAlignments = alignmentsByVersion[version - 1]
  for (const i of currentAlignments) {
    for (const j of currentAlignments) {
      if (isFinder(modules, i, j)) {
        // Skip the one overlapping with finders
        continue
      }

      aligments.push([i - 2, j - 2])
    }
  }

  return aligments
}
```



QR service - Some helpers (3/4)

```
export function isAlignment(aligments, y, x) {
  return aligments.some(([topLevelY, topLevelX]) => {
    return (
      y >= topLevelY &&
      y <= topLevelY + 4 &&
      x >= topLevelX &&
      x <= topLevelX + 4
    )
  })
}
```



QR service - Some helpers (4/4)

Similarly, **drawFinder** and **drawAligment** behave the same.
They are way more complex and I omitted them here to preserve your mental health.

```
export function drawCircle(x, y) {
  const posX = padding + x * unit
  const posY = padding + y * unit + radius

  return `M${posX},${posY} a${radius},${radius},0,1,0,${diameter},0 a${radius},${radius},0,1,0,${-diameter},0 `
}
```

QR service - Service skeleton

```
import { createServer } from 'node:http'
import QRCodeGenerator from 'qrcode-generator'
import {
  drawAlignment,
  drawCircle,
  drawFinder,
  getAlignments,
  isAlignment,
  isFinder,
  padding,
  unit
} from './rendering.js'

export function create() {
  return createServer((req, res) => {
    const data = req.url.split('/').at(-1)
    const url = Buffer.from(data, 'base64url').toString('utf-8')

    // Generate and return the QR code ...
  }
})
```



QR service - Compute modules

The service uses **qrcode-generator** from **Kazuhiko Arase** (which has no other dependencies).

```
const qr = QRCodeGenerator(0, "H")
qr.addData(url)
qr.make()

const modules = qr.getModuleCount()
const aligments = getAlignments(modules)
const dimension = modules * unit + padding
let svgPath = ""
let svgPoints = ""
```

QR service - Draw regular modules

```
// Draw regular modules
for (let x = 0; x < modules; x++) {
  for (let y = 0; y < modules; y++) {
    if (
      !qr.isDark(y, x) ||
      isFinder(modules, y, x) ||
      isAlignment(aligments, y, x)
    ) {
      continue
    }

    svgPoints += drawCircle(x, y)
  }
}

svgPath += `<path d="${svgPoints}" fill="currentColor" stroke="none" />`
```



QR service - Draw other elements

```
// Draw the finders
svgPath += drawFinder(0, 0)
svgPath += drawFinder(modules - 7, 0)
svgPath += drawFinder(0, modules - 7)

// Draw the aligments
for (const aligment of aligments) {
  svgPath += drawAlignment(aligment[0], aligment[1])
}
```



QR service - Return the SVG

```
res.writeHead(200, { 'x-plt-qr': url, 'content-type': 'image/svg+xml' })
res.end(`
  <svg
    xmlns="http://www.w3.org/2000/svg" version="1.1"
    width="${dimension}" height="${dimension}" viewBox="0 0 ${dimension} ${dimension}"
  >
    ${svgPath}
  </svg>
`)
```



Check it out!

All the code shown today is online on GitHub.

https://github.com/ShogunPanda/plt-qr-rendering

One last thing™

“Sharing is good, and with digital technology, sharing is easy.”

Richard Stallman





Thank you!

Paolo Insogna

Node.js TSC, Principal Engineer

@p_insogna

paolo.insogna@platformatic.dev

Platformatic