

Benchmark API Response YAZA dan Bahasa Lain

Pengukuran loopback HTTP/1.1 dan JSON | 2026-10-28 13:56:24 WIB | macOS-26.5-arm64-arm-64bit

10 implementasi	510 sampel	0 request gagal	R tidak tersedia
YAZA, Go, Nim, Python, Node.js, Ruby, PHP, Java, Kotlin, C#.NET	3 pengulangan; 1.000 request per sampel utama	Kontrak JSON divalidasi sebelum timing	Rscript tidak terpasang di mesin

Temuan utama

Pada GET JSON kecil, p50 YAZA adalah 140 us pada concurrency 1, 710 us pada concurrency 8, lalu 105.9 ms pada concurrency 32. Pada concurrency 32, Go mencatat 270 us dan Node.js 284 us. Hasil yang sama-sama lambat pada health route menunjukkan masalah high-concurrency yang tidak terbatas pada serialisasi JSON.

YAZA dapat menyajikan kontrak API dan lolos seluruh pemeriksaan fungsional. Benchmark ini menunjukkan bahwa implementasi server YAZA yang diuji perlu profiling konkurensi sebelum dapat dinilai kompetitif untuk layanan API dengan banyak koneksi bersamaan.

Latency GET JSON kecil

Nilai latensi adalah p50 / p95 / p99 dalam us; throughput adalah request per detik. Median dari tiga pengulangan.

Bahasa	c=1 p50/p95/p99	c=1 req/s	c=8 p50/p95/p99	c=8 req/s	c=32 p50/p95/p99	c=32 req/s
YAZA	140 / 298 / 393	5,839	710 / 1,434 / 1,499	9,794	105,934 / 112,393 / 113,081	401
Go	39 / 60 / 86	23,758	104 / 191 / 380	69,385	270 / 822 / 1,127	93,179
Nim	37 / 54 / 73	25,202	99 / 185 / 287	71,866	370 / 634 / 911	78,966
Python	78 / 96 / 125	12,353	402 / 800 / 1,039	18,378	1,757 / 2,329 / 3,102	17,637
Node.js	34 / 45 / 56	28,057	83 / 137 / 248	93,227	284 / 583 / 699	99,228
Ruby	136 / 238 / 269	6,882	1,059 / 2,113 / 4,828	6,768	4,322 / 8,180 / 9,748	7,058
PHP	109 / 144 / 181	8,834	317 / 686 / 855	21,124	1,878 / 2,952 / 6,740	13,485
Java	76 / 137 / 184	12,127	157 / 245 / 409	48,664	374 / 675 / 900	79,621
Kotlin	71 / 128 / 187	12,683	169 / 274 / 438	43,061	364 / 676 / 822	80,884
.NET / C#	70 / 96 / 169	12,809	271 / 521 / 630	26,402	1,218 / 1,669 / 2,844	24,421

P95 / p99 dan throughput pada concurrency 32

Bahasa	GET /healthz	GET JSON, values=1	GET JSON, values=16	GET JSON, values=64	POST JSON echo
YAZA	109.0 / 111.2 ms 496 req/s	112.4 / 113.1 ms 401 req/s	114.6 / 115.3 ms 396 req/s	112.8 / 115.2 ms 424 req/s	112.8 / 113.2 ms 549 req/s
Go	0.6 / 0.8 ms 120,647 req/s	0.8 / 1.1 ms 93,179 req/s	0.7 / 1.0 ms 110,861 req/s	0.7 / 0.9 ms 101,610 req/s	0.9 / 1.2 ms 89,365 req/s
Nim	0.7 / 1.0 ms 77,399 req/s	0.6 / 0.9 ms 78,966 req/s	0.6 / 0.8 ms 82,128 req/s	0.7 / 0.8 ms 68,096 req/s	0.6 / 0.8 ms 84,539 req/s
Python	2.1 / 2.5 ms 21,502 req/s	2.3 / 3.1 ms 17,637 req/s	2.4 / 2.9 ms 16,592 req/s	2.3 / 2.6 ms 17,247 req/s	2.9 / 3.9 ms 16,276 req/s
Node.js	0.6 / 0.7 ms 103,308 req/s	0.6 / 0.7 ms 99,228 req/s	0.6 / 0.7 ms 89,950 req/s	0.7 / 0.8 ms 86,136 req/s	0.6 / 0.8 ms 88,216 req/s
Ruby	5.6 / 7.1 ms 10,219 req/s	8.2 / 9.7 ms 7,058 req/s	8.8 / 10.2 ms 6,655 req/s	8.2 / 9.3 ms 6,897 req/s	6.3 / 7.8 ms 8,711 req/s
PHP	1.3 / 1.6 ms 29,824 req/s	3.0 / 6.7 ms 13,485 req/s	1.9 / 2.6 ms 21,930 req/s	1.3 / 1.4 ms 27,809 req/s	1.7 / 1.8 ms 23,985 req/s
Java	1.0 / 1.4 ms 54,020 req/s	0.7 / 0.9 ms 79,621 req/s	0.8 / 1.0 ms 71,065 req/s	0.8 / 1.0 ms 73,054 req/s	0.8 / 1.4 ms 74,182 req/s
Kotlin	0.8 / 1.1 ms 64,533 req/s	0.7 / 0.8 ms 80,884 req/s	0.7 / 0.9 ms 77,267 req/s	0.8 / 1.1 ms 79,911 req/s	0.8 / 1.0 ms 70,796 req/s
.NET / C#	0.7 / 1.0 ms 63,475 req/s	1.7 / 2.8 ms 24,421 req/s	1.5 / 2.0 ms 29,683 req/s	1.3 / 1.5 ms 33,979 req/s	0.8 / 0.9 ms 91,339 req/s

Cold start sampai ready

Satu peluncuran per runtime; compile tidak dihitung.

Bahasa	ms
YAZA	7.6
Go	32.6
Nim	31.2
Python	89.3
Node.js	161.6
Ruby	92.9
PHP	120.5
Java	206.6
Kotlin	114.3
.NET / C#	135.1

Metode dan batas hasil

Lima workload utama: health, GET JSON dengan array 1/16/64 elemen, dan POST JSON echo. Tiap workload berjalan pada concurrency 1, 8, dan 32, tiga kali, dengan 1.000 request per sampel. Kontrol koneksi baru memakai GET JSON kecil dengan 100 request per sampel pada concurrency 1 dan 8.

Totalnya 456.000 request terukur di sepuluh bahasa, ditambah warmup. Setiap server berjalan sendiri di 127.0.0.1:32100. Load generator Go net/http yang sama memeriksa status, membaca seluruh response body, dan mengukur waktu round-trip. Respons diuji secara semantik sebelum timing.

Tidak ada TLS, database, autentikasi, atau jaringan internet. YAZA memakai API native dan satu event loop; runtimes lain memakai server standar masing-masing. Ini perbandingan stack runtime/server pada satu mesin, bukan pengukuran bahasa murni.

Mesin: macOS-26.5-arm64-arm-64bit; 10 logical CPU; YAZA YAZA native compiler 0.1.0 (self-hosted YAZA source); Go go version go1.27.3 darwin/arm64; Nim Nim Compiler Version 2.2.12 [MacOSX: arm64]; Python 3.9.6; Node v26.0.0; Ruby ruby 2.6.10p210 (2022-04-12 revision 67958) [universal.arm64e-darwin25]; PHP PHP 8.5.6 (cli) (built: May 5 2026 21:19:36) (NTS); Java openjdk version "17.0.14" 2025-01-21; Kotlin info: kotlinc-jvm 2.4.21 (JRE 27.0.1.0.3); .NET 8.0.127.

Batas implementasi yang ditemukan: code generator YAZA gagal compile response shape yang memiliki dua nested shape berbeda. Untuk menyelesaikan pengujian tanpa mengubah compiler, profile menjadi satu nested object dan pagination diratakan menjadi field page, limit, total.

Kesimpulan: API YAZA lokal berfungsi dan prosesnya cepat mulai, namun hasil p50/throughput merosot tajam pada concurrency 32, termasuk endpoint health tanpa JSON kompleks. Profiling event loop, koneksi, dan backpressure adalah langkah lanjutan; angka ini belum membuktikan penyebab tunggal.

Keep-alive vs koneksi baru

GET response kecil; median p50 tiga pengulangan. Sampel koneksi baru berisi 100 request, bukan 1.000.

Bahasa	c1 keep-alive us	c1 baru us	c8 keep-alive us	c8 baru us
YAZA	140	264	710	837
Go	39	92	104	244
Nim	37	87	99	214
Python	78	143	402	612
Node.js	34	85	83	214
Ruby	136	196	1,059	1,288
PHP	109	112	317	275
Java	76	102	157	298
Kotlin	71	99	169	272
.NET / C#	70	116	271	323

Lampiran kode API

Kode lengkap berikut adalah kode server yang dijalankan. File load generator bersama ada di sources/loadgen/main.go.

YAZA

API native YAZA; JSON serializier bawaan

sources/yaza/src/main.yz

```
api:
  listen "127.0.0.1:3000"
  routes "routes"
```

sources/yaza/src/application.yz

```
use "controllers/ApiResponseImpl"
```

sources/yaza/src/routes/routes.yz

```
group "/api" auth public:
  route GET "/response" => controller ApiResponseRequest.api_response
  route POST "/echo" status 200 => controller ResponseEcho.api_echo
```

sources/yaza/src/controllers/ApiResponseRequest.yz

```
controller ApiResponseRequest:
  query requestId: Number = 42
  query name: Text = "Ada"
  query active: Boolean = true
  query score: Float64 = 98.75
  query items: Number = 1
  query page: Number = 2
  query limit: Number = 20
```

sources/yaza/src/controllers/ResponseEcho.yz

```
use "../contracts/ApiResponse"

controller ResponseEcho:
  body payload: ApiResponse
  example json {"requestId":42,"name":"Ada","active":true,"score":98.75,"note":null,"tags":["api","benchmark","yaza"],"profile":{"locale":"id-ID","rank":7,"verified":true},"values":[1,2,3],"page":2,"limit":20,"total":137}
```

sources/yaza/src/contracts/ApiResponse.yz

```
shape ResponseProfile:
  locale: Text
  rank: Number
  verified: Boolean

shape ApiResponse:
  requestId: Number
  name: Text
  active: Boolean
  score: Float64
  note: Option[Text]
  tags: TextList
  profile: ResponseProfile
  values: List[Number]
```

```
page: Number
limit: Number
total: Number
```

sources/yaza/src/controllers/ApiResponseImpl.yz

```
use "../contracts/ApiResponse"

fn api_response() -> ApiResponse:
  var request_id: Number = 42
  var name: Text = "Ada"
  var active: Boolean = true
  var score: Float64 = 98.75
  var item_count: Number = 1
  var page: Number = 2
  var limit: Number = 20

  let requested_id: Option[Number] = http_query_number("requestId")
  let requested_name: Option[Text] = http_query_parameter("name")
  let requested_active: Option[Boolean] = http_query_boolean("active")
  let requested_score: Option[Float64] = http_query_float64("score")
  let requested_count: Option[Number] = http_query_number("items")
  let requested_page: Option[Number] = http_query_number("page")
  let requested_limit: Option[Number] = http_query_number("limit")

  if option_is_some(requested_id):
    set request_id = option_unwrap(requested_id)
  if option_is_some(requested_name):
    set name = option_unwrap(requested_name)
  if option_is_some(requested_active):
    set active = option_unwrap(requested_active)
  if option_is_some(requested_score):
    set score = option_unwrap(requested_score)
  if option_is_some(requested_count):
    set item_count = option_unwrap(requested_count)
  if option_is_some(requested_page):
    set page = option_unwrap(requested_page)
  if option_is_some(requested_limit):
    set limit = option_unwrap(requested_limit)

  if item_count < 1:
    set item_count = 1
  if item_count > 64:
    set item_count = 64

  var tags: TextList = text_list()
  list_push(tags, "api")
  list_push(tags, "benchmark")
  list_push(tags, "yaza")

  var values: List[Number] = List[Number]()
  for i in 0..item_count:
    list_push(values, i + 1)

  return ApiResponse(request_id, name, active, score, none, tags, ResponseProfile("id-ID", 7, true), values, page, limit, 137)

fn api_echo() -> ApiResponse:
  let payload: Option[ApiResponse] = http_body_shape("payload")
  return option_unwrap(payload)
```

Go

net/http + encoding/json

sources/go/main.go

```
package main

import (
    "encoding/json"
    "net/http"
    "strconv"
)

type Profile struct {
    Locale   string `json:"locale"`
    Rank     int    `json:"rank"`
    Verified bool    `json:"verified"`
}

type Response struct {
    RequestID int64  `json:"requestId"`
    Name      string `json:"name"`
    Active    bool   `json:"active"`
    Score     float64 `json:"score"`
    Note      any     `json:"note"`
    Tags      []string `json:"tags"`
    Profile   Profile `json:"profile"`
    Values    []int   `json:"values"`
    Page      int64   `json:"page"`
    Limit     int64   `json:"limit"`
    Total     int     `json:"total"`
}

func intQuery(r *http.Request, key string, fallback int64) int64 {
    v, err := strconv.ParseInt(r.URL.Query().Get(key), 10, 64)
    if err != nil { return fallback }
    return v
}

func response(r *http.Request) Response {
    count := int(intQuery(r, "items", 1)); if count < 1 { count = 1 }; if count > 64 { count = 64 }
    score, err := strconv.ParseFloat(r.URL.Query().Get("score"), 64); if err != nil { score = 98.75 }
    active := true
    if v := r.URL.Query().Get("active"); v != "" { active = v == "true" }
    name := r.URL.Query().Get("name"); if name == "" { name = "Ada" }
    values := make([]int, count); for i := range values { values[i] = i+1 }
    return Response{intQuery(r, "requestId", 42), name, active, score, nil, []string{"api", "benchmark", "yaza"}, Profile{"id-ID", 7, true}, values, intQuery(r, "page", 2), intQuery(r, "limit", 20), 137}
}

func send(w http.ResponseWriter, status int, value any) {
    w.Header().Set("Content-Type", "application/json; charset=utf-8")
    body, err := json.Marshal(value)
    if err != nil { w.WriteHeader(500); return }
    w.WriteHeader(status); _, _ = w.Write(body)
}

func handler(w http.ResponseWriter, r *http.Request) {
    switch {
    case r.Method == http.MethodGet && r.URL.Path == "/healthz": send(w, 200, map[string]string{"status": "ok"})
    case r.Method == http.MethodGet && r.URL.Path == "/api/response": send(w, 200, response(r))
    case r.Method == http.MethodPost && r.URL.Path == "/api/echo":
        var payload any; decoder := json.NewDecoder(r.Body); decoder.UseNumber()
        if err := decoder.Decode(&payload); err != nil { send(w, 400, map[string]string{"error": "invalid_json"}); return }; send(w, 200, payload)
    default: send(w, 404, map[string]string{"error": "not_found"})
    }
}

func main() { http.HandleFunc("/", handler); _ = http.ListenAndServe("127.0.0.1:32100", nil) }
```

Nim

std/asynhttpserver + std/json

sources/nim/server.nim

```
import std/[asynhttpserver, asyncdispatch, json, strutils, uri]

proc queryValue(query, key, fallback: string): string =
  for pair in decodeQuery(query):
    if pair[0] == key: return pair[1]
  fallback

proc intValue(query, key: string, fallback: int): int =
  try: parseInt(queryValue(query, key, $fallback))
  except ValueError: fallback

proc response(query: string): JsonNode =
  let count = max(1, min(64, intValue(query, "items", 1)))
  var values = newJSONArray()
  for i in 1..count: values.add(%i)
  result = %* {
    "requestId": intValue(query, "requestId", 42),
    "name": queryValue(query, "name", "Ada"),
    "active": queryValue(query, "active", "true").toLowerAscii == "true",
    "score": parseFloat(queryValue(query, "score", "98.75")),
    "note": nil,
    "tags": ["api", "benchmark", "yaza"],
    "profile": {"locale": "id-ID", "rank": 7, "verified": true},
    "page": intValue(query, "page", 2),
    "limit": intValue(query, "limit", 20),
    "total": 137
  }
  result["values"] = values

proc callback(req: Request) {.async, gcsafe.} =
  var status = Http200
  var body = "{}"
  if req.reqMethod == HttpMethod and req.url.path == "/healthz":
    body = "{\"status\": \"ok\"}"
  elif req.reqMethod == HttpMethod and req.url.path == "/api/response":
    body = ($response(req.url.query))
  elif req.reqMethod == HttpMethod and req.url.path == "/api/echo":
    try: body = $(parseJson(req.body))
    except JsonParsingError:
      status = Http400
      body = "{\"error\": \"invalid_json\"}"
  else:
    status = Http404
    body = "{\"error\": \"not_found\"}"
  var headers = newHttpHeaders([{"Content-Type", "application/json; charset=utf-8"}])
  await req.respond(status, body, headers)

when isMainModule:
  var server = newAsyncHttpServer()
  echo "READY"
  waitFor server.serve(Port(32100), callback, address = "127.0.0.1")
```

Python

ThreadingHTTPServer + json

sources/python/server.py

```
#!/usr/bin/env python3
import json
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer
from urllib.parse import urlparse, parse_qs

def make_response(query):
    def value(name, default, convert):
        try:
            return convert(query.get(name, [str(default)])[0])
        except (TypeError, ValueError):
            return default

    count = max(1, min(64, value("items", 1, int)))
    return {
        "requestId": value("requestId", 42, int),
        "name": value("name", "Ada", str),
        "active": value("active", True, lambda x: x.lower() == "true"),
        "score": value("score", 98.75, float),
        "note": None,
        "tags": ["api", "benchmark", "yaza"],
        "profile": {"locale": "id-ID", "rank": 7, "verified": True},
        "values": list(range(1, count + 1)),
        "page": value("page", 2, int),
        "limit": value("limit", 20, int),
        "total": 137,
    }

class Handler(BaseHTTPRequestHandler):
    protocol_version = "HTTP/1.1"

    def log_message(self, *_args):
        pass

    def send_json(self, status, value):
        body = json.dumps(value, separators=(",", ":"), ensure_ascii=False).encode()
        self.send_response(status)
        self.send_header("Content-Type", "application/json; charset=utf-8")
        self.send_header("Content-Length", str(len(body)))
        self.end_headers()
        self.wfile.write(body)

    def do_GET(self):
        path = urlparse(self.path)
        if path.path == "/healthz":
            return self.send_json(200, {"status": "ok"})
        if path.path == "/api/response":
            return self.send_json(200, make_response(parse_qs(path.query, keep_blank_values=True)))
        self.send_json(404, {"error": "not_found"})

    def do_POST(self):
        if urlparse(self.path).path != "/api/echo":
            return self.send_json(404, {"error": "not_found"})
        try:
            length = int(self.headers.get("Content-Length", "0"))
            payload = json.loads(self.rfile.read(length))
            self.send_json(200, payload)
        except (ValueError, json.JSONDecodeError):
            self.send_json(400, {"error": "invalid_json"})

if __name__ == "__main__":
    server = ThreadingHTTPServer(("127.0.0.1", 32100), Handler)
```

```
server.daemon_threads = True  
print("READY", flush=True)  
server.serve_forever()
```

Node.js

core http + JSON

sources/node/server.js

```
"use strict";
const http = require("node:http");
const { URL } = require("node:url");

function makeResponse(query) {
  const number = (key, fallback) => {
    const n = Number(query.searchParams.get(key));
    return Number.isFinite(n) ? Math.trunc(n) : fallback;
  };
  const scoreValue = Number(query.searchParams.get("score"));
  const count = Math.max(1, Math.min(64, number("items", 1)));
  const activeValue = query.searchParams.get("active");
  return {
    requestId: number("requestId", 42),
    name: query.searchParams.get("name") ?? "Ada",
    active: activeValue === null ? true : activeValue.toLowerCase() === "true",
    score: Number.isFinite(scoreValue) ? scoreValue : 98.75,
    note: null,
    tags: ["api", "benchmark", "yaza"],
    profile: { locale: "id-ID", rank: 7, verified: true },
    values: Array.from({ length: count }, (_, i) => i + 1),
    page: number("page", 2),
    limit: number("limit", 20),
    total: 137,
  };
}

function send(res, status, value) {
  const body = Buffer.from(JSON.stringify(value));
  res.writeHead(status, { "Content-Type": "application/json; charset=utf-8", "Content-Length": body.length });
  res.end(body);
}

const server = http.createServer((req, res) => {
  const url = new URL(req.url, "http://127.0.0.1:32100");
  if (req.method === "GET" && url.pathname === "/healthz") return send(res, 200, { status: "ok" });
  if (req.method === "GET" && url.pathname === "/api/response") return send(res, 200, makeResponse(url));
  if (req.method === "POST" && url.pathname === "/api/echo") {
    let raw = "";
    req.setEncoding("utf8");
    req.on("data", chunk => { raw += chunk; });
    req.on("end", () => {
      try { send(res, 200, JSON.parse(raw)); }
      catch { send(res, 400, { error: "invalid_json" }); }
    });
    return;
  }
  send(res, 404, { error: "not_found" });
});
server.listen(32100, "127.0.0.1", () => console.log("READY"));
```

Ruby

WEBrick + JSON

sources/ruby/server.rb

```
require "webrick"
require "json"
require "uri"

def response(query)
  get = ->(key, fallback) { query.fetch(key, fallback) }
  begin
    count = Integer(get.call("items", "1"))
  rescue ArgumentError, TypeError
    count = 1
  end
  count = [[count, 1].max, 64].min
  active_value = get.call("active", "true").downcase == "true"
  score = Float(get.call("score", "98.75")) rescue 98.75
  {
    requestId: (Integer(get.call("requestId", "42")) rescue 42),
    name: get.call("name", "Ada"), active: active_value, score: score, note: nil,
    tags: ["api", "benchmark", "yaza"],
    profile: { locale: "id-ID", rank: 7, verified: true },
    values: (1..count).to_a,
    page: (Integer(get.call("page", "2")) rescue 2),
    limit: (Integer(get.call("limit", "20")) rescue 20), total: 137
  }
end

server = WEBrick::HTTPServer.new(Port: 32100, BindAddress: "127.0.0.1", AccessLog: [], Logger: WEBrick::Log.new(File::NULL))
server.mount_proc "/" do |req, res|
  res["Content-Type"] = "application/json; charset=utf-8"
  path = URI(req.request_uri.to_s).path
  if req.request_method == "GET" && path == "/healthz"
    res.status = 200; res.body = JSON.generate({ status: "ok" })
  elsif req.request_method == "GET" && path == "/api/response"
    res.status = 200; res.body = JSON.generate(response(req.query))
  elsif req.request_method == "POST" && path == "/api/echo"
    begin
      res.status = 200; res.body = JSON.generate(JSON.parse(req.body))
    rescue JSON::ParserError
      res.status = 400; res.body = '{"error":"invalid_json"}'
    end
  else
    res.status = 404; res.body = '{"error":"not_found"}'
  end
end
trap("INT") { server.shutdown }
puts "READY"
server.start
```

PHP

PHP CLI server + json

sources/php/router.php

```
<?php
declare(strict_types=1);

function int_param(string $key, int $fallback): int {
    $value = $_GET[$key] ?? null;
    if ($value === null || filter_var($value, FILTER_VALIDATE_INT) === false) return $fallback;
    return (int)$value;
}

function response_data(): array {
    $count = max(1, min(64, int_param('items', 1)));
    $score = isset($_GET['score']) && is_numeric($_GET['score']) ? (float)$_GET['score'] : 98.75;
    return [
        'requestId' => int_param('requestId', 42),
        'name' => $_GET['name'] ?? 'Ada',
        'active' => !isset($_GET['active']) || strtolower((string)$_GET['active']) === 'true',
        'score' => $score, 'note' => null,
        'tags' => ['api', 'benchmark', 'yaza'],
        'profile' => ['locale' => 'id-ID', 'rank' => 7, 'verified' => true],
        'values' => range(1, $count), 'page' => int_param('page', 2),
        'limit' => int_param('limit', 20), 'total' => 137,
    ];
}

function send_json(int $status, $value): void {
    http_response_code($status);
    header('Content-Type: application/json; charset=utf-8');
    echo json_encode($value, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES | JSON_THROW_ON_ERROR);
}

$path = parse_url($_SERVER['REQUEST_URI'] ?? '/', PHP_URL_PATH);
$method = $_SERVER['REQUEST_METHOD'] ?? 'GET';
if ($method === 'GET' && $path === '/healthz') send_json(200, ['status' => 'ok']);
elseif ($method === 'GET' && $path === '/api/response') send_json(200, response_data());
elseif ($method === 'POST' && $path === '/api/echo') {
    try { send_json(200, json_decode(file_get_contents('php://input'), true, 512, JSON_THROW_ON_ERROR)); }
    catch (JsonException $e) { send_json(400, ['error' => 'invalid_json']); }
} else send_json(404, ['error' => 'not_found']);
```

Java

JDK HttpServer + Gson 2.10.1

sources/java/ApiServer.java

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.concurrent.Executors;

public final class ApiServer {
    private static final Gson GSON = new GsonBuilder().disableHtmlEscaping().serializeNulls().create();

    private static Map<String, String> query(String raw) {
        Map<String, String> result = new LinkedHashMap<>();
        if (raw == null || raw.isEmpty()) return result;
        for (String part : raw.split("&")) {
            String[] pair = part.split("=", 2);
            result.put(URLDecoder.decode(pair[0], StandardCharsets.UTF_8),
                pair.length > 1 ? URLDecoder.decode(pair[1], StandardCharsets.UTF_8) : "");
        }
        return result;
    }

    private static long number(Map<String, String> q, String key, long fallback) {
        try { return Long.parseLong(q.getDefault(key, Long.toString(fallback))); }
        catch (NumberFormatException e) { return fallback; }
    }

    private static double decimal(Map<String, String> q, String key, double fallback) {
        try { return Double.parseDouble(q.getDefault(key, Double.toString(fallback))); }
        catch (NumberFormatException e) { return fallback; }
    }

    private static Map<String, Object> response(Map<String, String> q) {
        int count = (int) Math.max(1, Math.min(64, number(q, "items", 1)));
        ArrayList<Integer> values = new ArrayList<>(count);
        for (int i = 1; i <= count; i++) values.add(i);
        Map<String, Object> profile = new LinkedHashMap<>();
        profile.put("locale", "id-ID"); profile.put("rank", 7); profile.put("verified", true);
        Map<String, Object> result = new LinkedHashMap<>();
        result.put("requestId", number(q, "requestId", 42));
        result.put("name", q.getDefault("name", "Ada"));
        result.put("active", Boolean.parseBoolean(q.getDefault("active", "true")));
        result.put("score", decimal(q, "score", 98.75)); result.put("note", null);
        result.put("tags", new String[]{"api", "benchmark", "yaza"}); result.put("profile", profile);
        result.put("values", values); result.put("page", number(q, "page", 2));
        result.put("limit", number(q, "limit", 20)); result.put("total", 137);
        return result;
    }

    private static void send(HttpExchange exchange, int status, String body) throws IOException {
        byte[] bytes = body.getBytes(StandardCharsets.UTF_8);
        exchange.getResponseHeaders().set("Content-Type", "application/json; charset=utf-8");
        exchange.sendResponseHeaders(status, bytes.length);
        try (var out = exchange.getResponseBody()) { out.write(bytes); }
    }

    private static void handle(HttpExchange exchange) throws IOException {
        String method = exchange.getRequestMethod();
        String path = exchange.getRequestURI().getPath();
        if (method.equals("GET") && path.equals("/healthz")) send(exchange, 200, "{\"status\":\"ok\"}");
        else if (method.equals("GET") && path.equals("/api/response"))
            send(exchange, 200, GSON.toJson(response(query(exchange.getRequestURI().getRawQuery()))));
        else if (method.equals("POST") && path.equals("/api/echo")) {
            try {

```

```

        String body = new String(exchange.getRequestBody().readAllBytes(), StandardCharsets.UTF_8);
        send(exchange, 200, GSON.toJson(com.google.gson.JsonParser.parseString(body)));
    } catch (RuntimeException e) { send(exchange, 400, "{\"error\": \"invalid_json\"}"); }
    } else send(exchange, 404, "{\"error\": \"not_found\"}");
}
public static void main(String[] args) throws Exception {
    HttpServer server = HttpServer.create(new InetSocketAddress("127.0.0.1", 32100), 0);
    server.createContext("/", ApiServer::handle);
    server.setExecutor(Executors.newFixedThreadPool(Math.max(8, Runtime.getRuntime().availableProcessors())));
    server.start();
    System.out.println("READY");
}
}

```

Kotlin

JDK HttpServer + Gson 2.10.1

sources/kotlin/ApiServer.kt

```
import com.google.gson.GsonBuilder
import com.google.gson.JsonParser
import com.sun.net.httpserver.HttpExchange
import com.sun.net.httpserver.HttpServer
import java.net.InetSocketAddress
import java.net.URLDecoder
import java.nio.charset.StandardCharsets
import java.util.concurrent.Executors

object ApiServer {
    private val gson = GsonBuilder().disableHtmlEscaping().serializeNulls().create()

    private fun query(raw: String?): Map<String, String> = (raw?.split('&')?.filter { it.isNotEmpty() }?.associate { part -> val pieces = part.split('=', limit = 2) URLDecoder.decode(pieces[0], "UTF-8") to URLDecoder.decode(pieces.getOrElse(1) { "" }, "UTF-8") })

    private fun response(q: Map<String, String>): Map<String, Any?> {
        fun long(key: String, fallback: Long) = q[key]?.toLongOrNull() ?: fallback
        val count = long("items", 1).coerceIn(1, 64).toInt()
        return linkedMapOf(
            "requestId" to long("requestId", 42),
            "name" to (q["name"]?.let { "Ada" }),
            "active" to (q["active"]?.toBooleanStrictOrNull() ?: true),
            "score" to (q["score"]?.toDoubleOrNull() ?: 98.75),
            "note" to null,
            "tags" to listOf("api", "benchmark", "yaza"),
            "profile" to linkedMapOf("locale" to "id-ID", "rank" to 7, "verified" to true),
            "values" to (1..count).toList(),
            "page" to long("page", 2), "limit" to long("limit", 20), "total" to 137
        )
    }

    private fun send(exchange: HttpExchange, status: Int, body: String) {
        val bytes = body.toByteArray(StandardCharsets.UTF_8)
        exchange.responseHeaders.set("Content-Type", "application/json; charset=utf-8")
        exchange.sendResponseHeaders(status, bytes.size.toLong())
        exchange.responseBody.use { it.write(bytes) }
    }

    private fun handle(exchange: HttpExchange) {
        val path = exchange.requestURI.path
        when {
            exchange.requestMethod == "GET" && path == "/healthz" -> send(exchange, 200, "{\"status\":\"ok\"}")
            exchange.requestMethod == "GET" && path == "/api/response" -> send(exchange, 200, gson.toJson(response(query(exchange.requestURI.rawQuery))))
            exchange.requestMethod == "POST" && path == "/api/echo" -> try {
                val body = exchange.requestBody.bufferedReader(StandardCharsets.UTF_8).readText()
                send(exchange, 200, gson.toJson(JsonParser.parseString(body)))
            } catch (_: RuntimeException) { send(exchange, 400, "{\"error\":\"invalid_json\""} )
            else -> send(exchange, 404, "{\"error\":\"not_found\""} )
        }
    }

    @JvmStatic fun main(args: Array<String>) {
        val server = HttpServer.create(InetSocketAddress("127.0.0.1", 32100), 0)
        server.createContext("/") { exchange -> handle(exchange) }
        server.executor = Executors.newFixedThreadPool(maxOf(8, Runtime.getRuntime().availableProcessors()))
        server.start()
        println("READY")
    }
}
```

.NET / C#

.NET 8 HttpListener + System.Text.Json

sources/csharp/Program.cs

```
using System.Globalization;
using System.Net;
using System.Text;
using System.Text.Json;

static class ApiServer
{
    static long Long(HttpListenerRequest request, string key, long fallback) =>
        long.TryParse(request.QueryString[key], NumberStyles.Integer, CultureInfo.InvariantCulture, out var value) ? value : fallback;

    static object Response(HttpListenerRequest request)
    {
        var count = (int)Math.Clamp(Long(request, "items", 1), 1, 64);
        var score = double.TryParse(request.QueryString["score"], NumberStyles.Float, CultureInfo.InvariantCulture, out var s) ? s : 98.75;
        var active = request.QueryString["active"] is not string a || a.Equals("true", StringComparison.OrdinalIgnoreCase);
        return new Dictionary<string, object?> {
            ["requestId"] = Long(request, "requestId", 42), ["name"] = request.QueryString["name"] ?? "Ada",
            ["active"] = active, ["score"] = score, ["note"] = null,
            ["tags"] = new[] { "api", "benchmark", "yaza" },
            ["profile"] = new { locale = "id-ID", rank = 7, verified = true },
            ["values"] = Enumerable.Range(1, count).ToArray(), ["page"] = Long(request, "page", 2),
            ["limit"] = Long(request, "limit", 20), ["total"] = 137
        };
    }

    static async Task Send(HttpListenerResponse response, int status, object value)
    {
        var bytes = JsonSerializer.SerializeToUtf8Bytes(value);
        response.StatusCode = status;
        response.ContentType = "application/json; charset=utf-8";
        response.ContentLength64 = bytes.Length;
        await response.OutputStream.WriteAsync(bytes);
        response.Close();
    }

    static async Task Handle(HttpListenerContext context)
    {
        try {
            var req = context.Request;
            if (req.HttpMethod == "GET" && req.Url!.AbsolutePath == "/healthz") await Send(context.Response, 200, new { status = "ok" });
            else if (req.HttpMethod == "GET" && req.Url!.AbsolutePath == "/api/response") await Send(context.Response, 200, Response(req));
            else if (req.HttpMethod == "POST" && req.Url!.AbsolutePath == "/api/echo") {
                using var doc = await JsonDocument.ParseAsync(req.InputStream);
                await Send(context.Response, 200, doc.RootElement);
            } else await Send(context.Response, 404, new { error = "not_found" });
        } catch (JsonException) { await Send(context.Response, 400, new { error = "invalid_json" }); }
        catch { context.Response.Abort(); }
    }

    public static async Task Main()
    {
        var listener = new HttpListener();
        listener.Prefixes.Add("http://127.0.0.1:32100/");
        listener.Start();
        Console.WriteLine("READY");
        while (true) {
            var context = await listener.GetContextAsync();
            _ = Handle(context);
        }
    }
}
```

sources/csharp/ApiServer.csproj

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <Optimize>true</Optimize>
  </PropertyGroup>
</Project>
```