DevSheets

Hono

Small, fast web apps on Web Standards, in Hono 4.

01 Create & run

```
npm create hono@latest my-app
```

```
import { Hono } from 'hono'
const app = new Hono()
app.get('/', (c) => c.text('Hello Hono!'))
export default app
```

Pick a template: bun, nodejs, deno, cloudflare-workers, vercel, aws-lambda. Or add it with npm install hono.

02 Routes & HTTP methods

```
app.get('/users', (c) => c.json([]))
app.post('/users', (c) => c.json({}, 201))
app.put('/users/:id', handler)
app.delete('/users/:id', handler)
app.all('/hello', handler)   // any method
app.on('PURGE', '/cache', handler)
app.on(['PUT', 'DELETE'], '/post', handler)
```

Also patch, options and head. A handler gets the context c and returns a Response.

03 Path parameters

```
app.get('/posts/:id/tags/:tag', (c) => {
  const { id, tag } = c.req.param()
  return c.json({ id, tag })
})
app.get('/users/:id{[0-9]+}', handler)
app.get('/files/:path{.+}', handler)
```

c.req.param('id') reads one value. {regex} limits a segment; /animal/:type? makes the last one optional. A bare * captures nothing, so use :path{.+} for the rest of a path.

04 Route groups & basePath

```
const users = new Hono()
users.get('/', (c) => c.json([]))
users.get('/:id', (c) => c.json({}))
const app = new Hono().basePath('/api')
app.route('/users', users)
// GET /api/users, GET /api/users/:id
```

app.route(path, subApp) mounts a sub-app; basePath() prefixes every route in the app.

05 Request data

```
app.post('/posts', async (c) => {
  const q = c.req.query('q')         // ?q=hi
  const tags = c.req.queries('tag') // array
  const auth = c.req.header('Authorization')
  const body = await c.req.json()
  return c.json({ q, tags, auth, body })
})
```

Body readers are async: json(), text(), formData() and parseBody(). Header names are case-insensitive.

06 Responses

c.text(str)	Plain text
c.json(data, 201)	JSON, optional status
c.html(markup)	HTML
c.body(null, 204)	Raw body or no content
c.redirect(url)	302; pass 301 as 2nd argument
c.header(k, v)	Set a response header

setCookie(c, name, value, options), getCookie and deleteCookie come from hono/cookie.

07 Middleware

```
app.use('*', async (c, next) => {
  const start = Date.now()
  await next()             // handler runs
  const ms = Date.now() - start
  c.header('X-Response-Time', `${ms}ms`)
})
```

Return a Response instead of calling next() to stop the chain. createMiddleware() from hono/factory adds types.

08 Context variables

```
type Env = { Variables: { userId: string } }
const app = new Hono<Env>()
app.use('*', async (c, next) => {
  c.set('userId', '12345')
  await next()
})
app.get('/me', (c) =>
  c.json({ id: c.get('userId') }))
```

c.set() and c.get() carry values through one request. Strict TypeScript rejects keys not in Variables.

09 Errors & 404

```
if (!user) throw new HTTPException(404)
app.onError((err, c) =>
  err instanceof HTTPException
    ? err.getResponse()
    : c.text('Server error', 500))
app.notFound((c) => c.text('No route', 404))
```

HTTPException(status, { message }) is in hono/http-exception. A sub-app's own onError() wins for its routes.

Hono

Validation, typed clients, runtimes and tests.

10 Built-in middleware

`logger()`	Log each request
`cors(options)`	CORS and preflight
`secureHeaders()`	Security headers
`compress()`	Compress responses
`etag()`	ETag headers
`prettyJSON()`	Indent JSON on ?pretty

Each one has its own import path, such as hono/cors.

11 Auth middleware

```
app.use('/admin/*', basicAuth({
  username: 'admin', password: 'secret',
}))
app.use('/api/*', bearerAuth({ token }))
app.use('/auth/*', jwt({
  secret: 'my-jwt-secret', alg: 'HS256',
}))
```

jwt() requires alg; read the payload with c.get('jwtPayload'). sign() from hono/jwt takes exp in seconds.

12 Validation with Zod

```
const schema = z.object({
  name: z.string().min(1),
  email: z.email(),
})
app.post('/users',
  zValidator('json', schema),
  (c) => c.json(c.req.valid('json'), 201))
```

npm i @hono/zod-validator zod. Targets: json, query, param, header, form, cookie. Bad input gets a 400.

13 RPC: export the app type

```
const app = new Hono()
  .get('/users/:id', (c) =>
    c.json({ id: c.req.param('id') }))
  .post('/users',
    zValidator('json', schema),
    (c) => c.json(c.req.valid('json'), 201))
export type AppType = typeof app
```

Chain the routes: separate app.get() calls lose the route types. Groups chain too with .route().

14 RPC: typed client

```
const client = hc<AppType>(
  'http://localhost:8787')
const res = await client.users[':id'].$get({
  param: { id: '123' },
})
const user = await res.json() // typed
```

hc is in hono/client. $post() and the other methods take json, query, form and param values the same way.

15 Streaming & SSE

```
app.get('/sse', (c) =>
  streamSSE(c, async (stream) => {
    while (true) {
      const data = new Date().toISOString()
      await stream.writeSSE({ data })
      await stream.sleep(1000)
    }
  }))
```

From hono/streaming. writeSSE() also takes event, id and retry; stream.onAbort() runs when the client leaves.

16 Cloudflare Workers

```
const app = new Hono<{
  Bindings: { MY_KV: KVNamespace }
}>()
app.get('/kv/:key', async (c) => {
  const key = c.req.param('key')
  return c.json(await c.env.MY_KV.get(key))
})
```

Declare vars, KV, D1 and R2 in wrangler.jsonc or .toml and read them from c.env. Local secrets go in .dev.vars. Ship with npx wrangler deploy.

17 Bun, Deno & Node.js

```
// Bun: bun run --hot src/index.ts
export default app      // port 3000
// Deno: deno run --allow-net main.ts
Deno.serve(app.fetch)
// Node.js: npm i @hono/node-server
serve({ fetch: app.fetch, port: 3000 })
```

serve is in @hono/node-server. AWS Lambda: export const handler = handle(app) from hono/aws-lambda.

18 Testing with app.request()

```
it('GET / says hello', async () => {
  const res = await app.request('/')
  expect(res.status).toBe(200)
  expect(await res.text()).toContain('Hono')
})
// 2nd arg: fetch options; 3rd: mock c.env
await app.request('/kv/a', {}, { MY_KV })
```

No server runs: app.request() returns a Response. Works with Vitest, Jest or any test runner.