HTP – Harbour Thin Proxy Model

Reviving a 30-Year-Old Clipper DOS Accounting System – Now with a Modern Web UI

The modernization was extremely simple because the entire system follows the **HTP – Harbour Thin Proxy Model**:

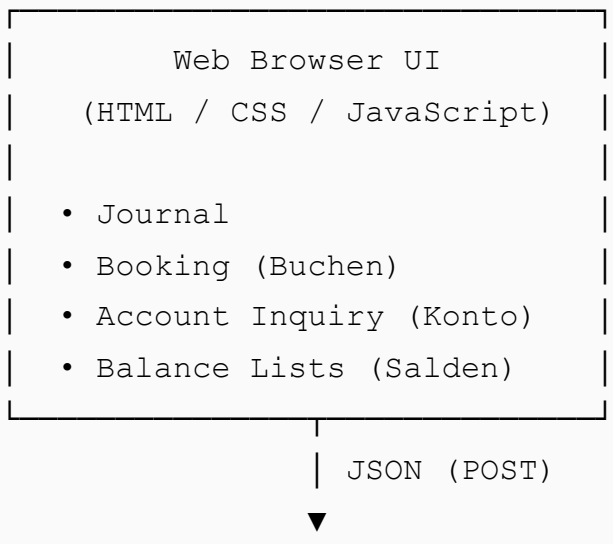
```
UI  →  Thin JSON Proxy  →  Microservice
```

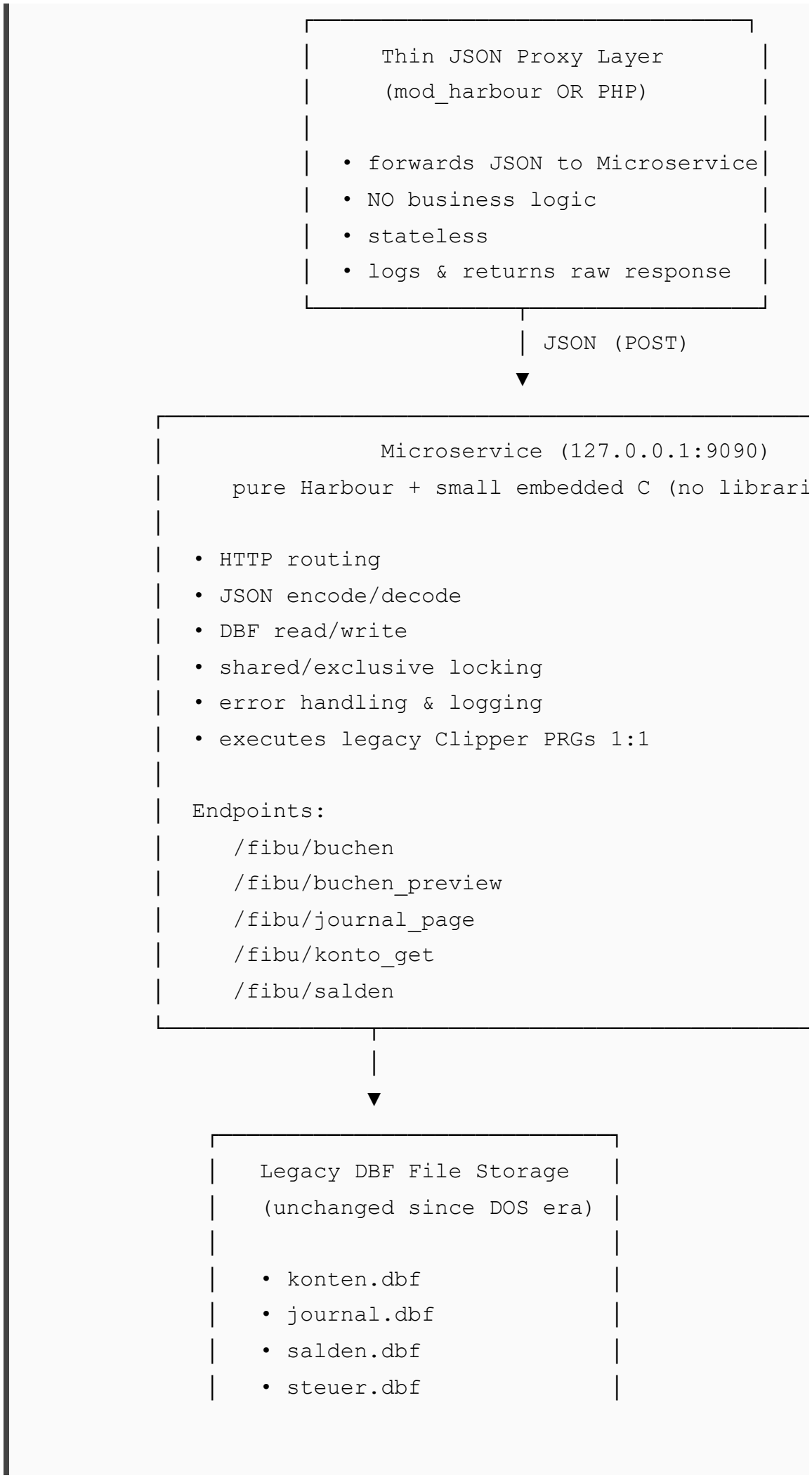
The UI sends JSON.
The Thin Proxy (mod_harbour or PHP) forwards it unchanged.
The Microservice executes pure Harbour logic against the DBF files.

No rewrite, no duplication, no frameworks.

Once this model is in place, turning an old Clipper application into a Web UI is literally a matter of hours, because the logic stays untouched and all that is needed is a requester and a few handlers.

Architecture Diagram




Thin JSON Proxy Layer
(mod_harbour OR PHP)
• forwards JSON to Microservice
• NO business logic
• stateless
• logs & returns raw response
JSON (POST)
Microservice (127.0.0.1:9090)
pure Harbour + small embedded C (no librari
• HTTP routing
• JSON encode/decode
• DBF read/write
• shared/exclusive locking
• error handling & logging
• executes legacy Clipper PRGs 1:1
Endpoints:
/fibu/buchen
/fibu/buchen_preview
/fibu/journal_page
/fibu/konto_get
/fibu/salden
Legacy DBF File Storage
(unchanged since DOS era)
• konten.dbf
• journal.dbf
• salden.dbf
• steuer.dbf

```
                    │    • ktobez.dbf                  │
                    └──────────────────────────────────┘
```

1. A Thin JSON Proxy (mod_harbour or PHP)

The proxy layer is intentionally minimal: it simply forwards JSON to the Microservice and returns the response unchanged.

- no business logic
- no state
- no duplication
- no side effects

For the forum I included a mod_harbour version of the proxy, because it fits naturally in the Harbour ecosystem.
I have not tested mod_harbour in production — my real setup uses a PHP-based JSON proxy, as it was already part of the infrastructure.

Architecturally both behave the same: a stateless JSON relay.

2. A powerful Microservice backend (pure Harbour, no external libraries)

The Microservice is written entirely in pure Harbour, with a tiny embedded C layer only for invoking Harbour VM functions.

There are:

- no external libraries
- no TipClientHttp
- no hbcurl
- no frameworks
- no foreign DBF drivers

Everything — DBF access, locking, JSON encode/decode, routing, logging, and execution of legacy Clipper PRGs — happens directly in Harbour. Latency is extremely low and stability is excellent.

3. AI-assisted preparation (DocChatty)

By supplying the full legacy source tree to the AI and defining the architectural line (“thin proxy → Microservice routing → legacy logic”), the system was automatically:

- analysed
- structured
- modularized
- rebuilt as a Web UI

The AI generated:

- HTML/CSS/JS UI
- project skeleton
- API route definitions
- proxy templates
- Microservice handler templates

This reduced what normally takes months to about an hour, because the business logic stayed untouched and the infrastructure already existed.

Result

- Modern Web UI (no frameworks)
- Original logic preserved
- Stateless JSON proxy
- Pure Harbour Microservice
- Clean separation of layers
- Zero rewrite of business rules
- Runs instantly with existing DBF files

Example Request (JavaScript)

```
fetch("/readdbf.prg", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ databasePath: "kunden.dbf" })
})
.then(r => r.json())
.then(console.log);
```

JSON Proxy (mod_harbour example)

This version is purely demonstrational and shows how a Thin Proxy could be implemented in mod_harbour. My production setup uses the PHP-based proxy instead.

```
procedure Main()

    local cBody      := mh_GetBody()
    local cUrl       := "http://127.0.0.1:9090/readdbf"
    local cResponse  := ""
    local oHttp, hIn

    if Empty(cBody)
       cBody := "{}"
    endif

    ? "Incoming JSON:", cBody

    hIn := hb_jsonDecode( cBody )

    oHttp := TipClientHttp():New( cUrl )
    oHttp:nConnTimeout := 5000
    oHttp:nRecvTimeout := 5000

    cResponse := oHttp:Post( cUrl, cBody, ;
                   { "Content-Type" => "application/json"

    if Empty(cResponse)
       mh_ResponseStatus( 502 )
       mh_Print( hb_jsonEncode( { ;
          "success" => .F., ;
          "message" => "Microservice not reachable", ;
          "error"   => oHttp:cError } ) )
       return
    endif

    mh_ResponseContentType( "application/json" )
    mh_Print( cResponse )
```

```
return
```

2. JSON Proxy (PHP version – PRODUCTION TESTED)

```
<?php
require_once($_SERVER['DOCUMENT_ROOT'] . '/shared/log_fun

header('Access-Control-Allow-Origin: *');
header('Content-Type: application/json; charset=utf-8');

$inputRaw = file_get_contents('php://input') ?: '{}';
logline('xwh_hub_readdbf payload', $inputRaw, __LINE__);

$msUrl = 'http://127.0.0.1:9090/readdbf';

$ctx = stream_context_create([
  'http' => [
    'method'  => 'POST',
    'header'  => "Content-Type: application/json\r\n",
    'content' => $inputRaw,
    'timeout' => 5
]]);
$resp = @file_get_contents($msUrl, false, $ctx);

if ($resp === false) {
  http_response_code(502);
  echo json_encode([
    'success' => false,
    'message' => 'Microservice not reachable',
    'error'   => error_get_last()['message'] ?? 'Unknown'
  ]);
  exit;
}

echo $resp;
```

Result

- Modern Web UI (no frameworks)
- Original logic preserved 1:1
- Stateless JSON proxy
- Pure Harbour Microservice backend
- Zero rewrite of business rules
- Runs instantly with existing DBF files