Understanding Tina4

Architecture, Philosophy, and the Four-Language Promise

v3.12.4 • tina4.com

The Intelligent Native Application 4ramework

Table of Contents

What Is Tina4?

The AI Framework Philosophy

TINA4 — The Intelligent Native Application 4ramework.

Four words. Each one carries weight.

Intelligent. The framework understands AI. Every project ships with a CLAUDE.md file that gives AI assistants complete knowledge of the framework's conventions, API, and structure. Your AI writes correct code on the first try because the framework told it how.

Native. Zero third-party dependencies. Every feature — the template engine, the JWT library, the SCSS compiler, the queue system, the GraphQL parser — is built from scratch using the language's standard library. No supply chain risk. No version conflicts. No surprises.

Application. This builds real production applications. Routing, ORM, authentication, queues, WebSocket, email, GraphQL, SOAP — 44 features, all included. One package. One install. Ship today.

4ramework. Four languages. One API. Python, PHP, Ruby, Node.js — learn the conventions once, build in any of them. The "4" is both the number and "for" — a framework *for* developers who value their time.

Here is a complete API endpoint:

```
<?php
// src/routes/greeting.php

Router::get("/api/greeting/{name}", function ($request, $response) {
    return $response->json([
        "message" => "Hello, " . $request->params["name"]
    ]);
});
```

No base controller. No service provider. No bootstrapping ritual. Drop that file into `src/routes/`, start the server, and it works. Your AI assistant knows this too — it reads the same conventions you do.

The philosophy fits in one sentence: **the framework that gets out of the way — for humans AND for AI.**

Routes go in `src/routes/`. Templates go in `src/templates/`. Models go in `src/orm/`. Learn the convention once. Your AI learns it once. Neither of you thinks about it again.

A decade of watching developers waste afternoons on configuration files, dependency conflicts, and framework upgrades that break everything. Then AI arrived and made the problem worse — every framework's ambiguity became the AI's confusion. Tina4 was born from both frustrations. One structure. One way to do things. The AI never guesses wrong because there is only one right answer.

Why Zero Dependencies Matters

Tina4 v3 has **zero third-party dependencies** for its core features. The template engine, the JWT library, the SCSS compiler, the queue system, the GraphQL parser, the logger, the rate limiter — every piece is built from scratch using the language's standard library.

This is a survival strategy — for you and for your AI assistant.

Security

Every dependency is an attack surface. When a package in your dependency tree gets compromised — and it will, ask the teams who trusted `event-stream`, `colors.js`, or `ua-parser-js` — your application is exposed.

Tina4's attack surface is the language runtime and your code. Nothing else sits between you and your users.

Size

A Laravel installation pulls in 70+ packages. A Rails app starts with 40+ gems. A Next.js project's `node_modules` folder is measured in hundreds of megabytes.

Tina4 installs **one package**. The framework runs to roughly **~26,000 lines of code** per language (Python ~26,000 | PHP ~35,000 | Ruby ~24,000 | Node.js ~32,000), all of it standard-library code you can read and audit. The Docker image fits in **40-80MB**. Your production container ships with what it needs. Nothing else tags along.

Portability

Zero dependencies means zero compatibility conflicts. You will never see this with Tina4:

```
Your requirements could not be resolved to an installable set of packages.
  Problem 1
    - package-a v2.1 requires other-package ^3.0
    - package-b v1.4 requires other-package ^2.0
```

No diamond dependency problem. No dependency tree to untangle. No Friday afternoon emergency because a transitive dependency released a breaking change.

Upgrades

Upgrading Tina4 means upgrading one package. No cascade of breaking changes. The framework team controls every line, so when something breaks, the fix lives in one place.

The One Exception

Database drivers are the exception. You cannot talk to PostgreSQL without a PostgreSQL driver. These are native connectors to external systems. They are optional — install only what you need. SQLite works out of the box with every language's standard library.

---

44 Features at 100% Parity

Tina4 ships with everything you need to build a production web application. 44 features, all implemented identically across Python, PHP, Ruby, and Node.js. 9,311 tests across 280 test files. Here is what every installation includes:

Core Web

- HTTP router with path parameters, typed params, middleware, and auth guards
- Request and Response objects with full HTTP access
- Static file serving, CORS, rate limiting, health checks
- Graceful shutdown, request ID tracking, structured logging
- Response compression, ETag support

Data Layer

- SQL-first ORM with Active Record pattern
- Five database drivers: SQLite, PostgreSQL, MySQL, MSSQL, Firebird
- Relationships: hasOne, hasMany, belongsTo with eager loading
- Migrations with rollback, seeders with 50+ fake data generators
- Query result caching with TTL, paginated results

Template and Frontend

- Frond: a Twig-compatible template engine with 55+ filters
- Template inheritance, includes, macros, pre-compilation
- SCSS compiler, tina4css (built-in CSS framework), frond.js (frontend helpers)

Auth and Sessions

- JWT (HS256/RS256) built from scratch
- Four session backends: file, Redis, Valkey, MongoDB
- CSRF protection, password hashing

Integration

- Queue system with retry, dead letters, and four backends (SQLite, RabbitMQ, Kafka, MongoDB)
- GraphQL parser and executor
- WebSocket server
- SOAP/WSDL support, HTTP API client, email messenger, i18n

Developer Experience

- Rust-based unified CLI with scaffolding, migrations, and testing
- Dev admin dashboard with 11 panels

- Error overlay with source code and stack traces
- Interactive gallery with 7 deployable examples
- Live reload, AI tool integration

All of this fits in a single package per language, with zero runtime dependencies. The biggest component — the Frond template engine — runs about 1,500 lines. Most features need fewer than 200.

Convention Over Configuration

Tina4 projects follow a predictable structure. Run `tina4 init` and you get:

```
my-project/
■■■ .env                    # Configuration
■■■ src/
■   ■■■ routes/             # Route handlers (auto-discovered)
■   ■■■ orm/                # ORM models (auto-discovered)
■   ■■■ templates/          # Frond templates
■   ■■■ public/             # Static files (served directly)
■   ■   ■■■ css/
■   ■   ■■■ js/
■   ■   ■■■ images/
■   ■■■ scss/               # SCSS source files (auto-compiled)
■■■ migrations/             # SQL migration files
■■■ data/                   # SQLite databases (gitignored)
■■■ logs/                   # Log files with rotation (gitignored)
■■■ tests/                  # Test files
```

Five rules. No exceptions:

- **Routes** go in `src/routes/`. Name the files however you want. Tina4 reads the route definitions inside them.
- **Models** go in `src/orm/`. Same auto-discovery.
- **Templates** go in `src/templates/`. Call `response.render("products/list.twig", data)` and Tina4 finds it.
- **Static files** go in `src/public/`. A file at `src/public/css/style.css` serves at `/css/style.css`.
- **Configuration** goes in `.env`. One file. Key-value pairs. No YAML. No TOML. No JSON config.

No routing table to maintain. No service container to wire up. No middleware stack to arrange in the right order. Drop files in the right directories. They work.

The Four-Language Paradigm

Tina4 is not one framework. It is four:

- **tina4-python** — Python 3.12+

- **tina4-php** — PHP 8.2+
- **tina4-ruby** — Ruby 3.1+
- **tina4-nodejs** — Node.js 20+ (TypeScript)

All four share the same project structure, the same `.env` variables, the same template syntax, the same CLI commands, and the same API contracts.

The only difference is naming convention:

Concept	Python / Ruby	PHP / Node.js
Method names	`snake_case`	`camelCase`
Fetch one row	`fetch_one()`	`fetchOne()`
Soft delete	`soft_delete()`	`softDelete()`

A team can prototype in Python and deploy in PHP without relearning the framework. Frontend developers using frond.js never need to know which backend language is running. DevOps deploys the same Docker structure, the same `.env`, the same health checks — regardless of language.

One Rust-based CLI binary auto-detects the project language and dispatches to the correct runtime:

```
tina4 init python ./my-app    # Scaffold a Python project
tina4 serve                   # Start dev server
tina4 generate model User     # Generate an ORM model
tina4 migrate                 # Run pending migrations
tina4 test                    # Run the test suite
```

What Tina4 Is Not

Tina4 does not replace Laravel, Django, Rails, or Next.js. Those are excellent frameworks for teams that want a full-stack opinion on everything.

Tina4 is for developers who want:

- **Control** — you see every line of code that runs your application
- **Simplicity** — one package, one import, predictable behaviour
- **Speed** — sub-millisecond framework overhead
- **Portability** — switch languages without switching paradigms
- **Security** — no supply chain risk from transitive dependencies

If you want a batteries-included platform with an ecosystem of plugins and a marketplace of themes, Tina4 is the wrong tool. If you want a sharp, minimal toolkit that does what you tell it and nothing else — keep reading.

The code you don't write is the code that never breaks.

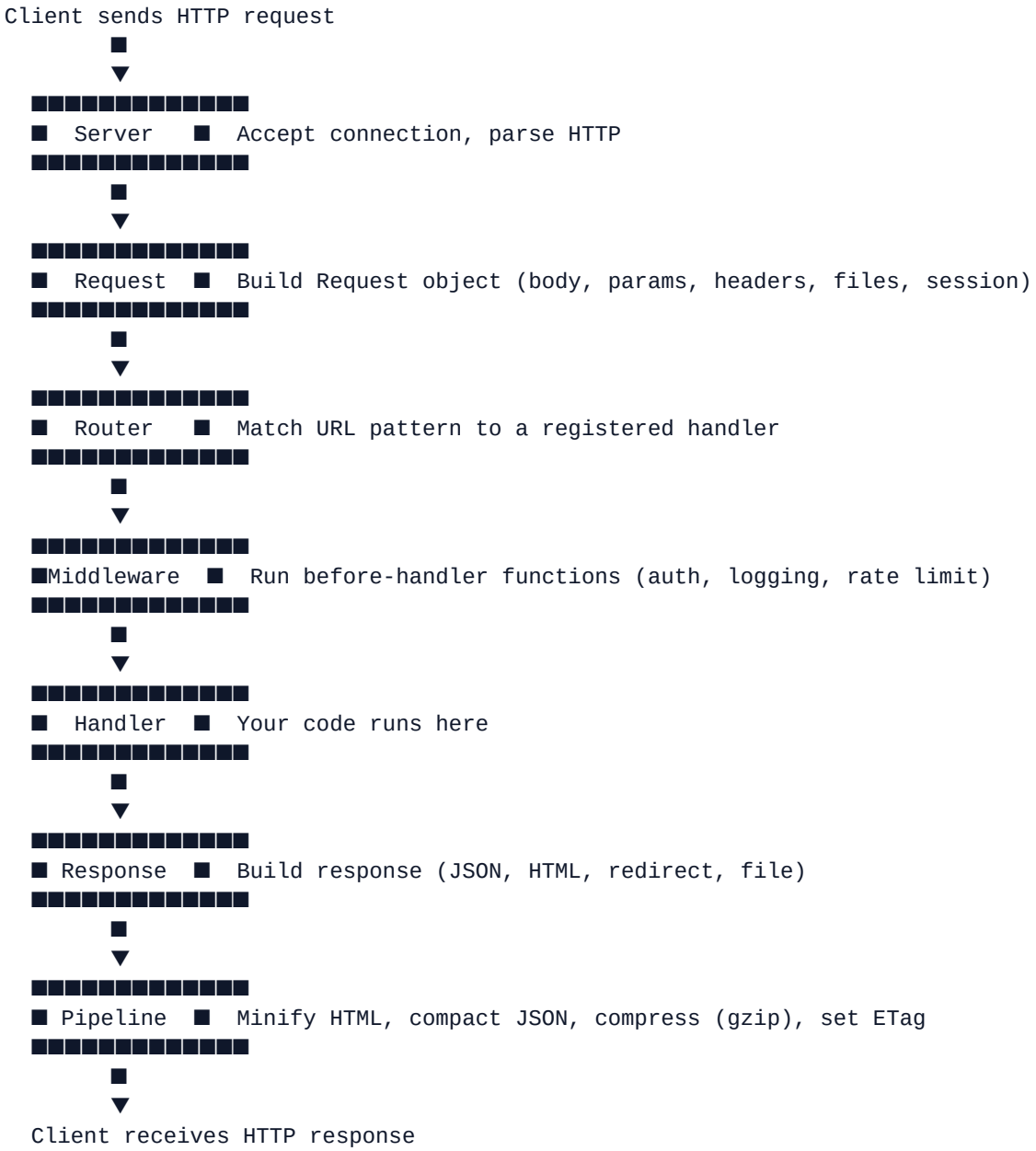
Summary

Aspect	Tina4
Philosophy	Toolkit, not a cathedral
Dependencies	Zero (core features)
Framework size	~26,000 lines per language (avg)
Languages	Python, PHP, Ruby, Node.js
Configuration	`.env` file only
Discovery	Automatic (routes, models, templates)
CLI	Unified Rust binary
Tests	9,311 across all four frameworks
Features	44 at 100% parity

Architecture

The Request Lifecycle

A request arrives. Seven stages later, a response leaves. Every Tina4 application follows this path -- Python, PHP, Ruby, Node.js. The language changes. The architecture does not.



Learn this diagram. Everything else in Tina4 is a footnote to it.

1. Server

The server accepts TCP connections and parses raw HTTP into structured data. Each language uses its native server:

Language	Server

Python	asyncio / ASGI
PHP	Built-in server / Swoole
Ruby	WEBrick / Puma
Node.js	`node:http`

You never touch this layer. Tina4 owns it. The server starts, listens, and hands off parsed requests. Your code lives further down the chain.

2. Request

The framework assembles a `Request` object. Everything the handler needs is already unpacked and waiting:

```
request.body        # Parsed JSON or form data
request.params      # Path parameters ({id} from /users/{id})
request.query       # Query string (?page=2&sort=name)
request.headers     # HTTP headers (case-insensitive)
request.files       # Uploaded files
request.session     # Session data (read/write)
request.method      # GET, POST, PUT, DELETE, etc.
request.path        # URL path without query string
request.ip          # Client IP address
request.request_id  # Unique ID for this request (for log correlation)
request.cookies     # Parsed cookies
request.is_json     # True if Content-Type contains "json"
```

No parsing. No extraction. No guessing where the data lives. One object. Twelve properties. Everything accounted for.

3. Router

The router listens. A request arrives. Method and path are matched against registered routes. Routes live in files under `src/routes/`:

```
// PHP
Router::get("/api/products/{id:int}", function ($request, $response) {
    // Only matches if {id} is an integer
});
```

```
# Python
@get("/api/products/{id:int}")
async def get_product(request, response):
    # Only matches if {id} is an integer
    pass
```

```
# Ruby
get "/api/products/{id:int}" do |request, response|
  # Only matches if {id} is an integer
end
```

```
// Node.js (file-based routing: src/routes/api/products/[id].ts)
export default function handler(request, response) {
    // id available via request.params.id
}
```

The router supports:

- **Basic parameters:** `/users/{id}` matches `/users/42`
- **Typed parameters:** `/users/{id:int}` only matches integers
- **Catch-all:** `/pages/{slug:.*}` matches `/pages/about/team/history`
- **Route groups:** Prefix multiple routes with a common path and middleware
- **Route caching:** Cache the response for a given TTL

No match on a registered route? The router checks `src/public/` for a static file. Still nothing? A 404 goes back to the client.

4. Middleware

Middleware functions form a pipeline. Each one inspects the request, decides whether to pass it forward, and optionally modifies what comes back. Think of it as a series of gates. A request must pass through every gate before it reaches your handler.

Tina4 ships four built-in middleware:

- **CORS** -- configured via environment variables, runs on every request
- **Rate limiting** -- 60 requests per minute per IP, out of the box
- **Auth gating** -- attach `.secure()` to a route to demand a valid JWT
- **Request ID tracking** -- generates or reads the `X-Request-ID` header

You write your own the same way:

```
// PHP
function logRequests($request, $response, $next) {
    $start = microtime(true);
    $result = $next($request, $response);
    $duration = round((microtime(true) - $start) * 1000, 2);
    Log::info("Request completed", [
        "method" => $request->method,
        "path" => $request->path,
        "duration_ms" => $duration
    ]);
    return $result;
}

Router::get("/api/users", $handler)->middleware([logRequests]);
```

A middleware receives the request, calls `$next` to continue the chain, and returns the result. Skip the `$next` call and the request stops right there. Short-circuit. The handler never runs. This is how auth guards work -- no valid token, no entry.

5. Handler

This is your territory. The handler receives a `Request` and a `Response`. What happens in between is your decision. Tina4 does not impose an application architecture. No base controllers. No service containers. No required inheritance. You receive two objects. You return a response.

6. Response

The `Response` object covers every common output:

```
response.json(data, statusCode)       # JSON with auto Content-Type
response.html(content, statusCode)    # HTML response
response.text(content, statusCode)    # Plain text
response.xml(content, statusCode)     # XML response
response.redirect(url, statusCode)    # HTTP redirect (302 or 301)
response.file(path)                   # File download with auto MIME type
response.render(template, data)       # Render a Frond template
response.status(code)                 # Set status code (chainable)
response.header(name, value)          # Set response header (chainable)
response.cookie(name, value, options) # Set a cookie
```

Ten methods. JSON, HTML, text, XML, redirects, files, templates, status codes, headers, cookies. Pick the one that fits. Chain what needs chaining.

7. Response Pipeline

Your handler finishes. The response is not done yet. It passes through an automatic pipeline -- five stages that optimize every response without a single line of configuration:

- **Frond rendering** -- if you called `response.render()`, the template compiles and executes
- **HTML minification** -- in production (`TINA4_DEBUG=false`), whitespace collapses, comments vanish. 15-25% smaller output.
- **JSON compaction** -- JSON ships compact. Add `?pretty=true` to the query string during development for readable output.
- **gzip compression** -- the client sends `Accept-Encoding: gzip` and the response exceeds 1KB? Compressed.
- **ETag generation** -- a hash of the response body becomes an `ETag` header. The next request with a matching `If-None-Match` gets a `304 Not Modified`. Zero bytes transferred.

Five optimizations. Zero configuration. Every response benefits.

Project Structure

Every Tina4 project follows the same directory layout. Python, PHP, Ruby, Node.js -- the folders are identical. A developer who has seen one Tina4 project has seen them all.

```
my-project/
■■■ .env                    # All configuration lives here
■■■ src/
■   ■■■ routes/             # Route handlers (auto-discovered)
■   ■■■ orm/                # ORM models (auto-discovered)
■   ■■■ migrations/         # SQL migration files
■   ■■■ seeds/              # Database seed files
■   ■■■ templates/          # Frond templates
■   ■   ■■■ errors/         # Custom error pages
■   ■■■ public/             # Static files (served at /)
■   ■   ■■■ js/
```

```
■   ■   ■   ■■■ frond.js    # Auto-provided
■   ■   ■■■ css/
■   ■   ■■■ scss/           # SCSS source files (auto-compiled)
■   ■   ■■■ images/
■   ■   ■■■ icons/
■   ■■■ locales/            # Translation files (JSON)
■       ■■■ en.json
■■■ data/                  # SQLite databases, .broken files
■■■ logs/                  # Log files with rotation
■■■ secrets/               # JWT keys
■■■ tests/                 # Test files
```

Fourteen directories. Each one has a single purpose. No overlap. No ambiguity.

`src/routes/` -- Where Your API Lives

Drop a file here. Tina4 finds the route definitions inside it. Organize however you want:

```
src/routes/
■■■ products.php        # All product routes
■■■ orders.php          # All order routes
■■■ admin/
    ■■■ users.php       # Admin user routes
    ■■■ reports.php     # Admin report routes
```

One file or twenty. Nested folders or flat. Tina4 reads them all. The file name does not affect the route path. Only the route definition inside the file matters. Name it `products.php` or `banana.php` -- the URL comes from the code, not the filename.

`src/orm/` -- Where Your Models Live

ORM model classes go here. Auto-discovered on startup:

```
src/orm/
■■■ Product.php
■■■ Order.php
■■■ OrderItem.php
■■■ User.php
```

Drop a class that extends the ORM base. Tina4 registers it. Auto-CRUD endpoints, route model binding, relationship resolution -- all of it flows from discovery.

`src/migrations/` -- Database Schema Changes

Migrations are SQL files with timestamps:

```
src/migrations/
■■■ 20260319100000_create_users_table.sql
■■■ 20260319100000_create_users_table.down.sql
■■■ 20260320090000_create_products_table.sql
■■■ 20260320090000_create_products_table.down.sql
```

The `.sql` file runs on `tina4 migrate`. The `.down.sql` file runs on `tina4 migrate --rollback`. Tina4 tracks which migrations have run in a `tina4_migrations` table. Forward and back. Always reversible.

`src/seeds/` -- Test Data

Seed files populate your database with test or default data. Run them with `tina4 seed`. Fifty built-in fake data generators handle the rest -- names, emails, addresses, phone numbers, dates.

`src/templates/` -- Frond Templates

Templates use the Frond engine. Inheritance, includes, filters, loops, conditionals -- all covered in detail later. The structure is yours to decide:

```
src/templates/
■■■ base.html           # Layout with blocks
■■■ index.html          # Extends base.html
■■■ products/
■   ■■■ list.html       # Product listing
■   ■■■ detail.html     # Single product
■■■ partials/
■   ■■■ header.html
■   ■■■ footer.html
■■■ errors/
    ■■■ 404.html        # Custom 404 page
    ■■■ 500.html        # Custom 500 page
```

`src/public/` -- Static Files

Files here are served at the root URL path. A file at `src/public/images/logo.png` appears at `/images/logo.png`. No route registration. No configuration. Drop it in. It serves.

The framework auto-provides `frond.js` in `src/public/js/` and keeps it in sync with the installed framework version. You never manage this file.

`data/` -- Runtime Data

SQLite databases live here by default (`data/app.db`). The `.broken/` subdirectory holds error marker files used by the health check. This entire directory belongs in `.gitignore`. Runtime data stays on the machine that runs the application.

`logs/` -- Log Files

Structured log files with automatic rotation. In `.gitignore`. Compressed after two days. Deleted after thirty. The framework handles all of it.

`secrets/` -- JWT Keys

Private and public keys for RS256 JWT signing. In `.gitignore`. Generated once, deployed with your application, never committed to source control.

.env Driven Configuration

One file controls everything. Not YAML. Not TOML. Not JSON config objects. A `.env` file at the project root. Key-value pairs. Plain text.

```
# .env
TINA4_DEBUG=true
```

```
TINA4_PORT=7145
TINA4_DATABASE_URL=sqlite:///data/app.db
JWT_SECRET=change-me-in-production
```

Four lines. A working application.

The Priority Chain

Tina4 resolves every configuration value through a three-level chain:

```
Constructor argument  >  .env file  >  Hardcoded default
```

The constructor wins. Always. The `.env` file is second. The hardcoded default is the safety net. This pattern applies to every configurable value in the framework. No exceptions.

Example: Priority Chain in Practice

```
# .env
TINA4_PORT=8080
```

```
// In code -- this overrides .env
$app = new Tina4\App(["port" => 9000]);
// Server starts on port 9000, not 8080
```

```
// No code override, no .env value
$app = new Tina4\App();
// Server starts on port 7145 (the default)
```

Three levels. Predictable resolution. Every time.

is_truthy() -- Boolean Environment Values

Environment variables are strings. Booleans do not exist in `.env` files. Tina4 bridges this gap with `is_truthy()`. Four values mean `true`:

- `true` (any case: `True`, `TRUE`, `tRuE`)
- `1`
- `yes` (any case)
- `on` (any case)

Everything else is `false`. Empty strings. Unset variables. Typos. If it is not on the list, it is `false`.

```
TINA4_DEBUG=true
TINA4_DEBUG=True
TINA4_DEBUG=1
TINA4_DEBUG=yes
TINA4_DEBUG=on
```

All equivalent. All enable debug mode.

```
TINA4_DEBUG=false
TINA4_DEBUG=0
TINA4_DEBUG=no
TINA4_DEBUG=off
TINA4_DEBUG=
# or simply omit the line
```

All disable it. No ambiguity.

Dev Mode vs. Production Mode

One variable. Two personalities. `TINA4_DEBUG` controls everything.

Dev Mode (`TINA4_DEBUG=true`)

The framework opens up. Every diagnostic tool activates:

- **Debug overlay** injects into every HTML response -- a toolbar showing request details, database queries, template render times, session data, and logs
- **Full stack traces** appear in the browser with source code context, the triggering request, and the queries that ran
- **Swagger UI** auto-registers at `/swagger`
- **Admin console** becomes available at `/__dev`
- **Live reload** watches for file changes and refreshes the browser
- **SQL query logging** writes every query to `logs/query.log`
- **Pretty JSON** is available via `?pretty=true` on any JSON endpoint
- **404 pages** show helpful route-not-found messages listing similar registered routes

Development mode assumes you want to see everything. It shows you everything.

Production Mode (`TINA4_DEBUG=false`)

The framework locks down. Every diagnostic tool disappears:

- **No debug overlay** -- responses ship clean
- **Generic error pages** -- no stack traces, no source code, no query details reach the browser
- **HTML minification** -- comments stripped, whitespace collapsed, 15-25% smaller output
- **.broken files** -- unhandled exceptions create marker files in `data/.broken/` that flip the health check to `503 Service Unavailable`, triggering container restarts
- **No Swagger UI** -- unless you force it on
- **No dev dashboard** -- only available when `TINA4_DEBUG=true`
- **No query logging** -- unless you enable it
- **Compact JSON only** -- no `?pretty=true`

Production mode assumes you want to expose nothing. It exposes nothing.

The default is `TINA4_DEBUG=false`. Forget to set it? The safe thing happens. Your application starts locked down.

One mistake will undo all of this: deploying with `TINA4_DEBUG=true`. Stack traces, database queries, session data -- visible to anyone with a browser. Set `TINA4_DEBUG=false` in production. Always.

The Frond Template Engine

Frond is Tina4's template engine. Zero dependencies. Built from scratch in each language. The syntax borrows from Twig -- developers who know Twig, Jinja2, or Nunjucks will recognize every construct. But no third-party template library runs underneath. Frond is Tina4's own.

Basic Syntax

Variables:

```
<h1>{{ title }}</h1>
<p>Welcome, {{ user.name }}</p>
<p>First item: {{ items[0] }}</p>
```

Filters (pipe syntax):

```
<p>{{ name | upper }}</p>           <!-- JOHN DOE -->
<p>{{ name | lower }}</p>           <!-- john doe -->
<p>{{ price | number_format(2) }}</p> <!-- 29.99 -->
<p>{{ text | truncate(100) }}</p>    <!-- First 100 chars... -->
<p>{{ description | raw }}</p>       <!-- No auto-escaping -->
<p>{{ items | length }}</p>          <!-- 5 -->
<p>{{ items | join(", ") }}</p>      <!-- apple, banana, cherry -->
```

Fifty-five filters. Strings, numbers, dates, arrays, encoding, formatting. If you need to transform data in a template, a filter exists.

Control structures:

```
{% if products | length > 0 %}
    {% for product in products %}
        <div class="product">
            <h2>{{ product.name }}</h2>
            <p>{{ product.price | number_format(2) }}</p>
            {% if loop.last %}
                <hr>
            {% endif %}
        </div>
    {% else %}
        <p>No products found.</p>
    {% endfor %}
{% endif %}
```

Template inheritance:

```
{# base.html #}
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}My App{% endblock %}</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
</html>

{# products/list.html #}
{% extends "base.html" %}
```

```
{% block title %}Products{% endblock %}

{% block content %}
    <h1>Our Products</h1>
    {% for product in products %}
        <p>{{ product.name }} - ${{ product.price | number_format(2) }}</p>
    {% endfor %}
{% endblock %}
```

Includes:

```
{% include "partials/header.html" %}
<main>
    {{ content }}
</main>
{% include "partials/footer.html" %}
```

Fragment caching:

```
{% cache "product-list" 300 %}
    {# This block is cached for 300 seconds #}
    {% for product in products %}
        <div>{{ product.name }}</div>
    {% endfor %}
{% endcache %}
```

One fact matters above all others: the template syntax is identical across all four languages. A template written for a Python backend works on PHP, Ruby, and Node.js without a single change. The backend is invisible to the frontend. Frond guarantees that.

How Auto-Discovery Works

Tina4 finds your code without being told where to look. No registration files. No import chains. No bootstrapping rituals. Drop files in the right directories. The framework discovers them.

Routes

On startup, Tina4 scans every file in `src/routes/` -- recursively, through every subdirectory. It finds route registration calls: `get()`, `post()`, `put()`, `delete()`, `any()`. Each call registers a route with the router.

```
Startup
  ■
  ■■■ Scan src/routes/
  ■   ■■■ products.php → registers GET /api/products, POST /api/products, ...
  ■   ■■■ orders.php → registers GET /api/orders, ...
  ■   ■■■ admin/users.php → registers GET /api/admin/users, ...
  ■
  ■■■ Router now has all routes in memory
```

The scan happens once at startup. In dev mode with live reload, the scan re-runs when files change. New route file saved? The router rebuilds. No restart needed.

The file name and path within `src/routes/` do not determine the URL. Only the route definition inside the file matters. Put all your routes in one file called `everything.php` and it

works. Spread them across fifty files in nested folders and it works. The directory structure is for your organization. The framework ignores it.

Models

ORM model classes in `src/orm/` follow the same pattern. The framework scans for classes that extend the ORM base class and registers them. This powers auto-CRUD -- REST endpoints generated from models. It powers route model binding -- URL parameters resolved to model instances. Discovery makes both possible without a single line of registration code.

Templates

Templates work differently. They are not discovered at startup -- they are loaded on demand when `response.render()` is called. But the framework knows where to look without being told. Reference `"products/list.html"` and Tina4 resolves it to `src/templates/products/list.html`. No path configuration. No template registry.

Static Files

When the router cannot match a request to a registered route, it falls back to the filesystem. The lookup follows a strict order:

```
1. Registered route?          → Run handler
2. File in src/public/?       → Serve static file
3. Framework built-in asset?  → Serve framework file
4. Nothing matches            → 404 response
```

Four steps. Tried in order. The first match wins. A file at `src/public/css/style.css` serves at `/css/style.css` with the correct MIME type. No route needed. No configuration needed. The file exists, so it serves.

Summary

Concept	How It Works in Tina4
Request lifecycle	Request > Router > Middleware > Handler > Response > Pipeline
Project structure	Fixed conventions: routes, orm, templates, public, migrations
Configuration	`.env` only, priority: constructor > .env > default
Dev vs. production	Single toggle: `TINA4_DEBUG`
Template engine	Frond: Twig-like syntax, zero dependencies, identical across languages
Auto-discovery	Files in `src/routes/` and `src/orm/` are found at startup
Static files	Files in `src/public/` are served at `/`
Response pipeline	Minification > compression > ETag, all automatic

Choosing Your Language

Tina4 runs on four languages. Same API. Same project structure. Same template syntax. Same CLI commands. The framework disappears -- what remains is your team, your hosting, and your problem.

This chapter gives you the information to choose.

Python

Best for: Data science teams, ML/AI integration, async-heavy applications, rapid prototyping.

```
# src/routes/products.py
from tina4_python import get, post

@get("/api/products")
async def list_products(request, response):
    products = Product.select(page=request.query.get("page", 1))
    return response.json(products)

@post("/api/products")
async def create_product(request, response):
    product = Product.create(request.body)
    return response.json(product, 201)
```

Why Choose Python

Async-native. Tina4 Python runs on asyncio. Every route handler is an async function. External API calls, database queries, file processing -- async means your server handles thousands of concurrent connections without spawning threads. One event loop. Thousands of requests.

Data science integration. Your web application sits next to a machine learning pipeline. Your API endpoint imports pandas, numpy, or scikit-learn directly. No inter-process communication. No REST calls to a separate service. The model runs in the same process as the route handler. Train in a notebook, deploy behind an endpoint.

The reference implementation. Tina4 Python was the first v3 framework to reach 100% completion. When ambiguity exists in the spec, Python's behavior is canonical. Every other implementation follows its lead.

Largest general-purpose community. Python is the most-taught programming language in universities worldwide. Finding Python developers is straightforward -- new graduates learn it first, data scientists think in it, and the library set grows faster than any other language.

Watch Out For

GIL limitations. CPU-bound work is single-threaded. The Global Interpreter Lock prevents true parallel execution of Python code. Image processing, PDF generation, heavy computation -- offload these to background tasks or a queue worker process. The event loop handles I/O. The GIL handles everything else, one thread at a time.

Deployment overhead. Python deployment demands more setup than PHP. Virtual environments, pip dependencies (Tina4 has zero, but your application might not), process managers. Docker simplifies this. Without Docker, plan for systemd units and environment isolation.

Package naming. Install with `pip install tina4-python`. The CLI command is `tina4` (auto-detects language). The import is `from tina4_python import ...`. Three different names for the same thing. Learn them once.

Quick Start

```
pip install tina4-python
tina4 init python my-project
cd my-project
tina4 serve
```

PHP

Best for: Existing PHP teams, shared hosting, CMS-adjacent projects, the widest hosting support on the planet.

```
<?php
// src/routes/products.php
use Tina4Router;

Router::get("/api/products", function ($request, $response) {
    $products = Product::select(page: $request->query["page"] ?? 1);
    return $response->json($products);
});

Router::post("/api/products", function ($request, $response) {
    $product = Product::create($request->body);
    return $response->json($product, 201);
});
```

Why Choose PHP

Hosting is everywhere. Shared hosting, VPS, dedicated servers, cloud platforms -- if it has a web server, it runs PHP. Every major hosting provider supports it. Every $5/month shared plan includes it. When your client's infrastructure is not negotiable, PHP fits.

Fastest Tina4. PHP 8.1+ with JIT compilation makes Tina4 PHP the fastest of the four implementations in raw request throughput. Add OPcache (compiled bytecode caching) and the framework overhead approaches zero. Requests arrive. Responses leave. The interpreter barely warms up.

Thirty years of libraries. Database drivers, payment processing, PDF generation, image manipulation -- PHP has well-tested libraries for all of it. Tina4 itself has zero dependencies. Your application can still pull in Composer packages when the need is real.

Monorepo simplicity. Tina4 PHP v3 is a single Composer package under the `Tina4\` namespace. No split packages. No sub-repositories. One `composer require`. Everything arrives.

Familiar ground. PHP was the first server-side language for a generation of web developers. If your team writes PHP, Tina4 speaks their language. The learning curve is the framework, not the language.

Watch Out For

No native async. Standard PHP is synchronous. One request, one thread, start to finish. For WebSocket support and true async behavior, you need the Swoole or OpenSwoole extension. Most web applications never need this. But if you plan to hold thousands of persistent connections, verify that Swoole is available on your hosting platform before you commit.

Extension management. PHP database drivers are C extensions: ext-pgsql, ext-mysqli, ext-sqlite3. On some hosting platforms, enabling these requires a support ticket or a PHP recompilation. Check your target platform's available extensions before you choose your database.

Case sensitivity. PHP uses `camelCase` for methods: `fetchOne()`, `softDelete()`, `hasMany()`. Coming from Python or Ruby? This is a style adjustment. The API is identical in capability -- the casing is different.

Quick Start

```
composer require tina4/tina4-php
tina4 init php my-project
cd my-project
tina4 serve
```

Ruby

Best for: Startups, elegant code, Rails refugees who want less framework and more control.

```
# src/routes/products.rb
get "/api/products" do |request, response|
  products = Product.select(page: request.query["page"] || 1)
  response.json(products)
end

post "/api/products" do |request, response|
  product = Product.create(request.body)
  response.json(product, 201)
end
```

Why Choose Ruby

Elegant syntax. Ruby was designed to make programmers happy. That philosophy shows. Route definitions read as English. Blocks feel natural. The DSL is clean. Code written in Tina4 Ruby looks the way you think about it.

Rails without the weight. You built Rails applications. You found yourself fighting the framework more than using it. Tina4 Ruby gives you the parts you valued -- convention over configuration, migrations, ORM, template engine -- and drops the parts you did not. No massive dependency tree. No opaque internals. No 15-minute boot time on a large codebase.

Startup velocity. Ruby's expressiveness means less code for the same functionality. Fewer lines. Fewer files. Faster iteration. For teams that need to ship and adjust, Ruby shrinks the distance between idea and production.

Strong testing culture. The Ruby community treats testing as a first-class concern. Tina4 Ruby integrates with Ruby's testing tools. The framework itself carries 2,333 tests. Testing is not an afterthought. It is the foundation.

Watch Out For

Smaller hosting pool. Ruby hosting is less common than PHP. Shared hosting with Ruby support is rare. Plan on a VPS, container platform, or PaaS -- Heroku, Render, Fly.io. The options exist. They are just not everywhere.

Performance. Ruby is not the fastest language. For raw throughput, PHP and Node.js outperform it. For most web applications, Ruby is fast enough. But if you are building a high-traffic API that must serve thousands of requests per second on a single instance, benchmark with your actual workload first. Do not assume.

Smaller talent pool. Ruby developers are harder to find than Python or JavaScript developers. The community is passionate but compact. If you are scaling a team, factor hiring timelines into the decision.

Quick Start

```
gem install tina4
tina4 init ruby my-project
cd my-project
tina4 serve
```

Node.js

Best for: JavaScript/TypeScript teams, file-based routing, real-time applications, highest raw throughput.

```
// src/routes/api/products/index.ts
export default function handler(request, response) {
    const products = Product.select({ page: request.query.page || 1 });
    return response.json(products);
}

// src/routes/api/products/index.post.ts
export default function handler(request, response) {
    const product = Product.create(request.body);
    return response.json(product, 201);
}
```

Why Choose Node.js

One language everywhere. Frontend and backend. Same developers. Same language. Same tooling. No context switching. A React component and its API endpoint live in the same mental model. JavaScript runs on both sides of the wire.

File-based routing. Tina4 Node.js maps file paths to URL paths. A file at `src/routes/api/products/[id].ts` handles `/api/products/42`. The filesystem is the routing table. If you have used Next.js, this pattern is already muscle memory.

Highest raw throughput. Node.js on V8 handles the most requests per second of the four implementations. For APIs that serve JSON and move data, Node.js delivers the highest ceiling. The event loop is purpose-built for I/O.

TypeScript from the ground up. Tina4 Node.js is written in TypeScript. Full type safety. Autocompletion in every editor. Compile-time error checking. Bugs surface before the server starts, not after.

WebSocket native. Node.js handles WebSocket connections without additional extensions. Real-time features -- chat, live dashboards, push notifications -- work out of the box. No Swoole. No extra gems. The runtime supports it.

Watch Out For

Async complexity. Modern async/await syntax is clean. The underlying tooling is not always clean. Unhandled promise rejections, callback-style legacy APIs, event loop blocking -- these traps exist. If your team is new to async JavaScript, expect a learning curve. The syntax is simple. The debugging is not.

Single-threaded for user code. Node.js runs your code on one thread. CPU-heavy operations block the event loop. Image processing, PDF generation, complex calculations -- offload these to worker threads or queue them for background processing. The event loop handles I/O brilliantly. It handles computation poorly.

Dependency gravity. The Node.js ecosystem gravitates toward micro-dependencies. Tina4 itself has zero core dependencies. Your application's other packages might pull in hundreds of transitive dependencies. Be deliberate about what enters your `node_modules`. Every `npm install` is a decision. Treat it as one.

Quick Start

```
npm install tina4
tina4 init nodejs my-project
cd my-project
tina4 serve
```

Comparison Table

Factor	Python	PHP	Ruby	Node.js
Install	`pip install tina4-python`	`composer require tina4/tina4-php`	`gem install tina4`	`npm install tina4`
CLI	`tina4`	`tina4`	`tina4`	`tina4`
Naming	`snake_case`	`camelCase`	`snake_case`	`camelCase`
Async	Native (asyncio)	Swoole extension	Rack-based	Native (event loop)
Hosting	VPS, containers, PaaS	Everywhere	VPS, containers, PaaS	VPS, containers, PaaS
Raw speed	Good	Very good (JIT)	Adequate	Best
Test count	2,112	2,220	2,333	2,646
Best for	Data/ML teams	Web agencies, existing PHP	Startups, clean code	JS/TS full-stack teams
WebSocket	Native async	Swoole required	Rack hijack / Puma	Native
Routing style	Decorator-based	Static method calls	DSL blocks	File-based + decorators
Learning curve	Low (if you know Python)	Low (if you know PHP)	Low (if you know Ruby)	Low (if you know JS/TS)
Framework LOC	~26,000	~35,000	~24,000	~32,000
Docker image	~60MB	~50MB	~70MB	~40MB
Community size	Largest (general)	Largest (web-specific)	Smallest	Large
Shared hosting	Rare	Universal	Rare	Rare
Package manager	pip	Composer	RubyGems	npm

Switching Languages

All four implementations share the same skeleton. Switching languages is not a rewrite. It is a translation.

Five things transfer without any changes:

- `src/templates/` -- Frond syntax is identical. Every template works on every backend.
- `.env` -- Same variables. Same defaults. Same priority chain.
- `src/migrations/` -- Same SQL. The database does not care what language talks to it.
- `src/public/` -- Same static files. CSS, JavaScript, images. Untouched.
- `frond.js` frontend code -- Same API. The backend language is invisible to the browser.

Three things need translation:

- `src/routes/` -- Route handlers rewritten in the new language's syntax.
- `src/orm/` -- Model definitions rewritten in the new language's class system.
- `tests/` -- Test files rewritten for the new language's test framework.

The framework knowledge transfers completely. The project structure stays. The templates stay. The migrations stay. The configuration stays. You translate your business logic. Nothing else.

How to Decide

Five questions. Answer them in order. Stop at the first clear answer.

- **What does your team already know?** Use that language. The Tina4 learning curve is the same across all four. The learning curve for a new programming language is not. Do not add two learning curves when one will do.
- **Where are you deploying?** Shared hosting means PHP. Containers mean anything. Serverless platforms -- check which runtimes are supported before you start writing code.
- **What sits next to the web app?** ML models mean Python. A React or Vue frontend team means Node.js. An existing PHP codebase means PHP. The web application is rarely the only software in the system. Choose the language that fits the neighborhood.
- **How many concurrent connections do you need?** Chat applications, live dashboards, thousands of WebSocket connections -- use Python or Node.js for native async. PHP needs Swoole. Ruby handles moderate concurrency with Puma. Know your ceiling before you build.
- **Does raw performance matter?** For most applications, all four are fast enough. The framework overhead is sub-millisecond in every language. The bottleneck is your database queries, not your router. But if you need 10,000+ requests per second on a single instance, benchmark with your actual workload. Numbers on paper are not numbers in production.

If none of these questions produce a clear answer, use Python. It is the reference implementation. It has the largest general-purpose community. It is the most-taught language in the world. When all else is equal, start where the most help exists.

Environment Variables

Every piece of Tina4 configuration lives in one file. A `.env` at the root of your project. All optional. All with sensible defaults. Identical across Python, PHP, Ruby, and Node.js. This chapter is the complete reference.

How .env Files Work

A `.env` file is plain text. Key-value pairs. Nothing more.

```
# This is a comment
TINA4_DATABASE_URL=sqlite:///data/app.db
TINA4_DEBUG=true

# Blank lines are ignored

# Values with spaces need quotes
TINA4_MAIL_FROM="My App <noreply@example.com>"

# No quotes needed for simple values
TINA4_SECRET=my-secret-key-change-in-production
```

Rules

Six rules govern the format:

- One variable per line.
- Format is `KEY=VALUE` -- no spaces around the `=`.
- Lines starting with `#` are comments.
- Blank lines are ignored.
- Values can be wrapped in double quotes (`"value"`) or single quotes (`'value'`).
- No variable interpolation -- `$OTHER_VAR` is treated as a literal string.

Simple format. No surprises. A developer who has never seen a `.env` file understands it in thirty seconds.

The .env File Is Not Committed to Git

The `.env` file holds secrets. Database passwords. JWT keys. API tokens. It belongs in `.gitignore`. When you run `tina4 init`, the generated `.gitignore` already excludes it.

Commit a `.env.example` instead. Placeholder values. A map for the next developer:

```
# .env.example -- copy to .env and fill in real values
TINA4_DATABASE_URL=sqlite:///data/app.db
TINA4_DEBUG=false
TINA4_SECRET=CHANGE_ME
TINA4_MAIL_HOST=smtp.example.com
TINA4_MAIL_USERNAME=
TINA4_MAIL_PASSWORD=
```

The example file documents what the application expects. The real file stays on the machine that runs it. Never in the repository.

The Priority Chain

Tina4 resolves every configurable value through a three-level chain:

```
Constructor argument  >  .env file  >  Hardcoded default
```

Three levels. Strict order. No exceptions.

- **Constructor argument wins.** Pass a value in code and it overrides everything.
- **.env file is second.** No code override? The `.env` value takes over.
- **Default is last.** Neither code nor `.env` specifies a value? The framework's built-in default applies.

This pattern holds across all variables and all four language implementations. Learn it once. Apply it everywhere.

Boolean Values

Environment variables are strings. The `.env` file has no concept of `true` or `false`. Tina4 recognises these values as truthy:

Value	Treated as
`true`, `True`, `TRUE`	`true`
`1`	`true`
`yes`, `Yes`, `YES`	`true`
`on`, `On`, `ON`	`true`

Everything else is `false`:

Value	Treated as
`false`, `0`, `no`, `off`	`false`
(empty string)	`false`
(variable not set)	`false`

Write whichever style your team prefers:

```
# All of these enable debug mode
TINA4_DEBUG=true
TINA4_DEBUG=1
TINA4_DEBUG=yes
```

Complete .env Reference

Server

Variable	Default	Description
`HOST`	`0.0.0.0`	Bind address. `0.0.0.0` listens on all interfaces (required for Docker). `127.0.0.1` restricts to localhost.
`PORT`	See below	HTTP server port. Each framework has a unique default to avoid conflicts when running side-by-side.
`TINA4_DEBUG`	`false`	Master toggle. Enables debug overlay, full stack traces, Swagger UI, live reload, query logging. **Never** `true` **in production.**

Default ports by framework:

Framework	Default Port
PHP	`7145`
Python	`7146`
Ruby	`7147`
Node.js	`7148`

Authentication

Variable	Default	Description
`TINA4_SECRET`	`tina4-default-secret`	Secret key for JWT signing (HMAC-SHA256). Long, random, never committed to git. **Change this in production.**
`TINA4_API_KEY`	*(none)*	Static API key for bearer token authentication. When set, requests with `Authorization: Bearer {TINA4_API_KEY}` are accepted.
`TINA4_TOKEN_LIMIT`	`60`	Token lifetime in minutes. Tokens issued by `get_token()` / `getToken()` expire after this many minutes.

CSRF Protection

Variable	Default	Description
`TINA4_CSRF`	`true`	Enable CSRF token validation on POST/PUT/PATCH/DELETE. Set to `false` to disable (e.g. for internal microservices behind a firewall).

CSRF is **on by default**. When enabled:

- POST/PUT/PATCH/DELETE requests must include a `formToken` in the request body or an `X-Form-Token` header.
- GET/HEAD/OPTIONS requests are not checked.
- Requests with a valid `Authorization: Bearer` token skip CSRF validation.
- Routes marked `@noauth()` skip CSRF validation.
- Tokens in query strings are **rejected** (security risk).

To disable for internal services:

```
TINA4_CSRF=false
```

Database

Variable	Default	Description
`TINA4_DATABASE_URL`	`sqlite:///data/app.db`	Connection string. The URL scheme selects the driver.
`TINA4_DATABASE_USERNAME`	*(from URL)*	Override the username in `TINA4_DATABASE_URL`. Useful when credentials contain special characters.
`TINA4_DATABASE_PASSWORD`	*(from URL)*	Override the password in `TINA4_DATABASE_URL`.
`TINA4_AUTOCOMMIT`	`false`	Enable auto-commit after every write operation. Default is off -- use explicit `commit()` calls.

Connection string formats:

```
# SQLite (default -- no credentials needed)
TINA4_DATABASE_URL=sqlite:///data/app.db

# PostgreSQL
TINA4_DATABASE_URL=postgresql://user:password@hostname:5432/database_name

# MySQL / MariaDB
TINA4_DATABASE_URL=mysql://user:password@hostname:3306/database_name

# Microsoft SQL Server
TINA4_DATABASE_URL=mssql://user:password@hostname:1433/database_name

# Firebird
TINA4_DATABASE_URL=firebird://user:password@hostname:3050/path/to/database.fdb

# MongoDB (SQL queries are auto-translated)
TINA4_DATABASE_URL=mongodb://user:password@hostname:27017/database_name
```

Gotcha: Special characters in your database password -- `@`, `#`, `:`, `/` -- will break URL parsing. URL-encode them (`@` becomes `%40`) or split the credentials into separate variables.

DB Query Cache

Variable	Default	Description
`TINA4_DB_CACHE`	`false`	Enable in-memory caching of query results.
`TINA4_DB_CACHE_TTL`	`30`	Cache time-to-live in seconds.

ORM

Variable	Default	Description
`TINA4_ORM_PLURAL_TABLE_NAMES`	`false`	Append "s" to auto-generated table names. When `false` (default), `Product` maps to `product`. When `true`, `Product` maps to `products`.

CORS (Cross-Origin Resource Sharing)

Variable	Default	Description
`TINA4_CORS_ORIGINS`	`*`	Comma-separated allowed origins. `*` allows all. In production, list your actual domains.
`TINA4_CORS_METHODS`	`GET,POST,PUT,PATCH,DELETE,OPTIONS`	Comma-separated HTTP methods allowed in cross-origin requests.
`TINA4_CORS_HEADERS`	`Content-Type,Authorization,X-Request-ID`	Comma-separated headers the client is allowed to send.
`TINA4_CORS_CREDENTIALS`	`true`	Whether the browser sends cookies and auth headers in cross-origin requests.
`TINA4_CORS_MAX_AGE`	`86400`	How long (seconds) the browser caches preflight responses. `86400` = 24 hours.

Gotcha: `TINA4_CORS_ORIGINS=*` combined with `TINA4_CORS_CREDENTIALS=true` is invalid per the CORS spec. Tina4 handles this automatically -- when origin is `*`, the credentials header is not sent.

Security Headers

Variable	Default	Description
`TINA4_FRAME_OPTIONS`	`SAMEORIGIN`	`X-Frame-Options` header. Prevents clickjacking. Options: `DENY`, `SAMEORIGIN`.
`TINA4_HSTS`	*(empty/off)*	`Strict-Transport-Security` max-age in seconds. Set to `31536000` (1 year) in production with HTTPS.
`TINA4_CSP`	`default-src 'self'`	`Content-Security-Policy` header. Controls which resources the browser is allowed to load.
`TINA4_REFERRER_POLICY`	`strict-origin-when-cross-origin`	`Referrer-Policy` header. Controls what referrer info is sent with requests.
`TINA4_PERMISSIONS_POLICY`	`camera=(), microphone=(), geolocation=()`	`Permissions-Policy` header. Disables browser features your app doesn't need.

The `X-Content-Type-Options: nosniff` and `X-XSS-Protection: 0` headers are always set (no env variable -- these are security best practices).

Rate Limiter

Variable	Default	Description
`TINA4_RATE_LIMIT`	`100`	Maximum requests per window per IP address.
`TINA4_RATE_WINDOW`	`60`	Window duration in seconds. Default: 100 requests per 60 seconds.

The rate limiter adds three headers to every response:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 85
X-RateLimit-Reset: 60
```

When the limit is exceeded, the server returns `429 Too Many Requests`.

Sessions

Variable	Default	Description
`TINA4_SESSION_BACKEND`	`file`	Session storage backend. Options: `file`, `redis`, `valkey`, `mongo`, `database`.
`TINA4_SESSION_PATH`	`data/sessions`	Directory for file-based sessions. Relative to the project root.
`TINA4_SESSION_TTL`	`3600`	Session expiry in seconds. Default: 1 hour.
`TINA4_SESSION_SAMESITE`	`Lax`	SameSite cookie attribute. Options: `Strict`, `Lax`, `None`.

Redis/Valkey Session Backend

Variable	Default	Description
`TINA4_SESSION_REDIS_HOST`	`127.0.0.1`	Redis host.
`TINA4_SESSION_REDIS_PORT`	`6379`	Redis port.
`TINA4_SESSION_REDIS_PASSWORD`	*(none)*	Redis password.
`TINA4_SESSION_REDIS_DB`	`0`	Redis database number.
`TINA4_SESSION_REDIS_PREFIX`	`tina4:session:`	Key prefix for session data.
`TINA4_SESSION_VALKEY_HOST`	`localhost`	Valkey host.
`TINA4_SESSION_VALKEY_PORT`	`6379`	Valkey port.
`TINA4_SESSION_VALKEY_PASSWORD`	*(none)*	Valkey password.
`TINA4_SESSION_VALKEY_DB`	`0`	Valkey database number.
`TINA4_SESSION_VALKEY_PREFIX`	`tina4:session:`	Key prefix for session data.

MongoDB Session Backend

Variable	Default	Description
`TINA4_SESSION_MONGO_URI`	*(none)*	Full MongoDB connection URI. Overrides host/port.
`TINA4_SESSION_MONGO_HOST`	`localhost`	MongoDB host.
`TINA4_SESSION_MONGO_PORT`	`27017`	MongoDB port.
`TINA4_SESSION_MONGO_USERNAME`	*(none)*	MongoDB username.
`TINA4_SESSION_MONGO_PASSWORD`	*(none)*	MongoDB password.
`TINA4_SESSION_MONGO_DB`	`tina4_sessions`	MongoDB database name.
`TINA4_SESSION_MONGO_COLLECTION`	`sessions`	MongoDB collection for session data.

Queue

Variable	Default	Description
`TINA4_QUEUE_BACKEND`	`file`	Queue storage backend. Options: `file`, `rabbitmq`, `kafka`, `mongodb`.
`TINA4_QUEUE_PATH`	`data/queue`	Directory for file-based queue storage (when using `file` backend).
`TINA4_QUEUE_URL`	*(none)*	Generic connection URL for queue backend.

RabbitMQ Queue Backend

Variable	Default	Description
`TINA4_RABBITMQ_HOST`	`localhost`	RabbitMQ host.
`TINA4_RABBITMQ_PORT`	`5672`	RabbitMQ port.
`TINA4_RABBITMQ_USERNAME`	`guest`	RabbitMQ username.
`TINA4_RABBITMQ_PASSWORD`	`guest`	RabbitMQ password.
`TINA4_RABBITMQ_VHOST`	`/`	RabbitMQ virtual host.

Kafka Queue Backend

Variable	Default	Description
`TINA4_KAFKA_BROKERS`	`localhost:9092`	Comma-separated Kafka broker addresses.
`TINA4_KAFKA_GROUP_ID`	`tina4_consumer_group`	Kafka consumer group ID.

MongoDB Queue Backend

Variable	Default	Description
`TINA4_MONGO_URI`	*(none)*	Full MongoDB connection URI. Overrides host/port.
`TINA4_MONGO_HOST`	`localhost`	MongoDB host.
`TINA4_MONGO_PORT`	`27017`	MongoDB port.
`TINA4_MONGO_USERNAME`	*(none)*	MongoDB username.
`TINA4_MONGO_PASSWORD`	*(none)*	MongoDB password.
`TINA4_MONGO_DB`	`tina4`	MongoDB database name.
`TINA4_MONGO_COLLECTION`	`tina4_queue`	MongoDB collection for queue messages.

Response Cache

Variable	Default	Description
`TINA4_CACHE_BACKEND`	`memory`	Cache storage. Options: `memory` (in-process), `redis`, `file`.
`TINA4_CACHE_TTL`	`60`	Default cache TTL in seconds for cached routes.
`TINA4_CACHE_MAX_ENTRIES`	`1000`	Maximum cached responses (memory backend). Oldest entries evicted at the limit.
`TINA4_CACHE_DIR`	`data/cache`	Directory for file-based cache.
`TINA4_CACHE_URL`	`redis://localhost:6379`	Redis connection URL for cache backend.

Logging

Variable	Default	Description
`TINA4_LOG_LEVEL`	`ERROR`	Minimum log level. Options: `ALL`, `DEBUG`, `INFO`, `WARNING`, `ERROR`.
`TINA4_LOG_MAX_SIZE`	`10`	Maximum log file size in MB before rotation.
`TINA4_LOG_KEEP`	`5`	Number of rotated log files to keep.

Messenger (Email / SMTP)

Tina4 supports two naming conventions for SMTP variables. The `SMTP_*` variables are the primary names. The `TINA4_MAIL_*` variables are aliases that take precedence when both are set.

Variable	Alias	Default	Description
`TINA4_MAIL_HOST`	`TINA4_MAIL_HOST`	`localhost`	SMTP server hostname.
`TINA4_MAIL_PORT`	`TINA4_MAIL_PORT`	`587`	SMTP port. `587` (TLS), `465` (SSL), `25` (unencrypted).

`TINA4_MAIL_USERNAME`	`TINA4_MAIL_USERNAME`	*(none)*	SMTP authentication username.
`TINA4_MAIL_PASSWORD`	`TINA4_MAIL_PASSWORD`	*(none)*	SMTP authentication password.
`TINA4_MAIL_FROM`	`TINA4_MAIL_FROM`	`noreply@localhost`	Default sender address.
`TINA4_MAIL_FROM_NAME`	`TINA4_MAIL_FROM_NAME`	*(none)*	Sender display name.
`TINA4_MAIL_ENCRYPTION`	—	`tls`	Connection encryption. `tls`, `ssl`, or `none`.

IMAP (for reading email)

Variable	Default	Description
`TINA4_MAIL_IMAP_HOST`	*(falls back to TINA4MAILHOST)*	IMAP server hostname.
`TINA4_MAIL_IMAP_PORT`	`993`	IMAP port (993 = SSL).

Localization (i18n)

Variable	Default	Description
`TINA4_LOCALE`	`en`	Default locale. Determines which translation file (`src/locales/{locale}.json`) is loaded.
`TINA4_LOCALE_DIR`	`src/locales`	Directory containing translation JSON files.

Swagger / OpenAPI

Variable	Default	Description
`TINA4_SWAGGER_TITLE`	`Tina4 API`	API title shown in Swagger UI.
`TINA4_SWAGGER_VERSION`	`1.0.0`	API version shown in Swagger UI.
`TINA4_SWAGGER_DESCRIPTION`	*(none)*	API description.

File Uploads

Variable	Default	Description
`TINA4_MAX_UPLOAD_SIZE`	`10485760`	Maximum upload size in bytes. Default: 10 MB.

WebSocket

Variable	Default	Description
`TINA4_WS_PORT`	`8080`	WebSocket server port (when running as separate process).
`TINA4_WS_BACKPLANE`	*(none)*	WebSocket backplane type. Set to `redis` to relay broadcasts across instances.
`TINA4_WS_BACKPLANE_URL`	`redis://localhost:6379`	Connection URL for the WebSocket backplane.

Services (Background Workers)

Variable	Default	Description
`TINA4_SERVICE_DIR`	`src/services`	Directory for service worker scripts.
`TINA4_SERVICE_SLEEP`	`5`	Seconds between service worker ticks.

Dev Mailbox

Variable	Default	Description
`TINA4_MAILBOX_DIR`	`data/mailbox`	Directory for captured dev emails.

Minimal .env for Development

Getting started? One line:

```
TINA4_DEBUG=true
```

Everything else uses sensible defaults:

- Binds to `0.0.0.0` on the framework's default port
- SQLite database at `data/app.db`

- File-based sessions (1 hour TTL)
- File-based queue
- In-memory response cache
- CORS allows all origins
- 100 requests per minute rate limit
- CSRF protection enabled
- Security headers active

One line. A working development environment. Add variables when you need them. Not before.

Minimal .env for Production

```
TINA4_DEBUG=false
TINA4_SECRET=a-very-long-random-string-at-least-32-characters
TINA4_DATABASE_URL=postgresql://app_user:strong_password@db-host:5432/myapp
TINA4_CORS_ORIGINS=https://myapp.com
TINA4_HSTS=31536000
TINA4_MAIL_HOST=smtp.sendgrid.net
TINA4_MAIL_PORT=587
TINA4_MAIL_USERNAME=apikey
TINA4_MAIL_PASSWORD=SG.xxxxx
TINA4_MAIL_FROM=noreply@myapp.com
```

Ten lines. A production application. Debug disabled. Real database. Signed tokens. Locked CORS. HSTS enabled. Email configured. Everything else keeps its defaults.

Docker .env

When running in Docker, `HOST` must be `0.0.0.0` so the container accepts connections from outside. This is already the default, but if you override it, keep this in mind:

```
# Required for Docker -- do NOT set to 127.0.0.1
HOST=0.0.0.0
PORT=7145
TINA4_DEBUG=false
```

Full .env Template

Copy this to your `.env.example` as a starting point:

```
# ==============================================================================
# Tina4 Environment Configuration
# Copy this file to .env and fill in your values
# ==============================================================================

# --- Server ---
HOST=0.0.0.0
# PORT=7145
TINA4_DEBUG=false

# --- Authentication ---
TINA4_SECRET=CHANGE_ME
# TINA4_API_KEY=
```

```
TINA4_TOKEN_LIMIT=60

# --- CSRF ---
TINA4_CSRF=true

# --- Database ---
TINA4_DATABASE_URL=sqlite:///data/app.db
# TINA4_DATABASE_USERNAME=
# TINA4_DATABASE_PASSWORD=
# TINA4_AUTOCOMMIT=false

# --- DB Query Cache ---
# TINA4_DB_CACHE=false
# TINA4_DB_CACHE_TTL=30

# --- ORM ---
# TINA4_ORM_PLURAL_TABLE_NAMES=false

# --- CORS ---
TINA4_CORS_ORIGINS=*
TINA4_CORS_METHODS=GET,POST,PUT,PATCH,DELETE,OPTIONS
TINA4_CORS_HEADERS=Content-Type,Authorization,X-Request-ID
TINA4_CORS_CREDENTIALS=true
TINA4_CORS_MAX_AGE=86400

# --- Security Headers ---
# TINA4_FRAME_OPTIONS=SAMEORIGIN
# TINA4_HSTS=
# TINA4_CSP=default-src 'self'
# TINA4_REFERRER_POLICY=strict-origin-when-cross-origin
# TINA4_PERMISSIONS_POLICY=camera=(), microphone=(), geolocation=()

# --- Rate Limiting ---
TINA4_RATE_LIMIT=100
TINA4_RATE_WINDOW=60

# --- Logging ---
TINA4_LOG_LEVEL=ERROR
# TINA4_LOG_MAX_SIZE=10
# TINA4_LOG_KEEP=5

# --- Sessions ---
TINA4_SESSION_BACKEND=file
# TINA4_SESSION_PATH=data/sessions
TINA4_SESSION_TTL=3600
# TINA4_SESSION_REDIS_HOST=127.0.0.1
# TINA4_SESSION_REDIS_PORT=6379
# TINA4_SESSION_VALKEY_HOST=localhost
# TINA4_SESSION_VALKEY_PORT=6379

# --- Queue ---
TINA4_QUEUE_BACKEND=file
# TINA4_RABBITMQ_HOST=localhost
# TINA4_RABBITMQ_PORT=5672
# TINA4_RABBITMQ_USERNAME=guest
# TINA4_RABBITMQ_PASSWORD=guest
# TINA4_KAFKA_BROKERS=localhost:9092
# TINA4_KAFKA_GROUP_ID=tina4_consumer_group
# TINA4_MONGO_HOST=localhost
```

```
# TINA4_MONGO_PORT=27017
# TINA4_MONGO_DB=tina4
# TINA4_MONGO_COLLECTION=tina4_queue

# --- Response Cache ---
TINA4_CACHE_BACKEND=memory
TINA4_CACHE_TTL=60
TINA4_CACHE_MAX_ENTRIES=1000

# --- Email (SMTP) ---
# TINA4_MAIL_HOST=smtp.example.com
# TINA4_MAIL_PORT=587
# TINA4_MAIL_USERNAME=
# TINA4_MAIL_PASSWORD=
# TINA4_MAIL_FROM=noreply@example.com

# --- Localization ---
TINA4_LOCALE=en

# --- Swagger ---
TINA4_SWAGGER_TITLE=Tina4 API
TINA4_SWAGGER_VERSION=1.0.0
# TINA4_SWAGGER_DESCRIPTION=

# --- File Uploads ---
# TINA4_MAX_UPLOAD_SIZE=10485760

# --- Services ---
# TINA4_SERVICE_DIR=src/services
# TINA4_SERVICE_SLEEP=5
```

Summary

Count	Category
3	Server (HOST, PORT, TINA4_DEBUG)
3	Authentication (TINA4*SECRET, TINA4*API*KEY, TINA4*TOKEN_LIMIT)
1	CSRF (TINA4_CSRF)
4	Database (TINA4*DATABASE*URL, USERNAME, PASSWORD, AUTOCOMMIT)
2	DB query cache
5	CORS
5	Security headers
2	Rate limiter
3	Logging
14	Sessions (base + Redis + Valkey + MongoDB)
15	Queue (base + RabbitMQ + Kafka + MongoDB)
5	Response cache
8	Messenger (SMTP + IMAP)
2	Localization
3	Swagger
1	File uploads
3	WebSocket
2	Services
1	Dev mailbox
82	**Total**

Every variable follows the same priority chain: constructor > `.env` > default. Every boolean is interpreted consistently across all four frameworks. Every variable has a sensible default that works for development without any configuration.

One file. Eighty-two knobs. Turn what you need. Leave the rest alone.