



**راهنمای جامع چهارچوب ترکیبی ماهی  
آموزش ساخت صفحات وب و API**

نویسنده مهدی خلیل زاده

در ابتدا، از پدر و مادر عزیزم تشکر می‌کنم که همیشه با عشق و حمایت بی‌قید و شرط خود، به من قدرت تفکر، آزادی ایده‌پردازی و فضایی برای رشد دادند. قدردان زحمات و فداکاری‌های آن‌ها هستم و این کتاب را به آن‌ها تقدیم می‌کنم.

فهرست مطالب

[فصل ۱: ماهی چیست 9](#_Toc187838758)

[چرا ماهی ؟ 9](#_Toc187838759)

[آشنایی با فریمورک ماهی 9](#_Toc187838760)

[ماهی چیست ؟ 9](#_Toc187838761)

[ویژگی های کلیدی ماهی چیست ؟ 9](#_Toc187838762)

[چرا Lua در توسعه وب؟ 9](#_Toc187838763)

[مزایای ماهی برای توسعه‌دهندگان 10](#_Toc187838764)

[استفاده از Lua به‌جای زبان‌های پیچیده‌تر 10](#_Toc187838765)

[سادگی و قابلیت شخصی‌سازی 10](#_Toc187838766)

[سرعت و عملکرد 10](#_Toc187838767)

[سبک و کم حجم 11](#_Toc187838768)

[مناسب برای پروژه‌های خاص و اختصاصی 11](#_Toc187838769)

[پشتیبانی از APIهای سریع و قابل گسترش 11](#_Toc187838770)

[یادگیری سریع 11](#_Toc187838771)

[فصل 2: نصب و راه اندازی 12](#_Toc187838772)

[پیش‌نیازها 12](#_Toc187838773)

[.NET 8.0 یا بالاتر 12](#_Toc187838774)

[دانلود و نصب ماهی 12](#_Toc187838775)

[نصب ماهی در ویندوز 12](#_Toc187838776)

[نصب ماهی در لینوکس 13](#_Toc187838777)

[اجرای اولین پروژه 13](#_Toc187838778)

[فصل 3: ساختار پروژه در ماهی 14](#_Toc187838779)

[ساختار فایل پیکر بندی (Config) 14](#_Toc187838780)

[ساختار پوشه ها و فایل ها در ماهی 17](#_Toc187838781)

[پوشه اصلی پروژه: wwwapp 18](#_Toc187838782)

[تغییر نام پوشه اصلی در فایل کانفیگ 18](#_Toc187838783)

[توضیحات فایل ها 18](#_Toc187838784)

[توضیحات پوشه ها 18](#_Toc187838785)

[مسیریابی در ماهی 19](#_Toc187838786)

[ساختار Route 20](#_Toc187838787)

[ساختار فایل HTMLUA در ماهی 21](#_Toc187838788)

[یک فایل Htmlua شامل بخش‌های زیر است: 21](#_Toc187838789)

[چگونه از کدهای Lua در HTML استفاده کنیم؟ 21](#_Toc187838790)

[نمونه‌ی کامل 22](#_Toc187838791)

[فصل ۴: توابع داخلی ماهی 23](#_Toc187838792)

[فصل ۵: مدیریت درخواست‌های HTTP 26](#_Toc187838793)

[ماژول های HTTP 26](#_Toc187838794)

[کاربرد ماژول ها 26](#_Toc187838795)

[ایجاد ماژول 26](#_Toc187838796)

[نحوه کار ماژول 27](#_Toc187838797)

[فصل ۶: کار با کنترلرها 27](#_Toc187838798)

[فصل ۷: ساخت API با ماهی 27](#_Toc187838799)

[فصل ۸: پروژه‌های عملی 27](#_Toc187838800)

[فصل ۹: اشکال‌زدایی و بهینه‌سازی 27](#_Toc187838801)

فهرست مطالب

فصل ۱: ماهی چیست

* آشنایی با فریمورک ماهی
* چرا Lua در توسعه وب؟
* مزایای ماهی برای توسعه‌دهندگان

فصل ۲: نصب و راه‌اندازی

* پیش‌نیازها
* دانلود و نصب ماهی
* اجرای اولین پروژه

فصل ۳: ساختار پروژه در ماهی

* فایل‌ها و پوشه‌های اصلی
* نحوه مدیریت منابع ثابت
* تنظیمات اولیه پروژه

فصل ۴: مفاهیم پایه ماهی

* معرفی زبان Lua
* متغیرها، انواع داده و عملگرها
* توابع و ساختارهای کنترلی
* ماژول‌ها در Lua

فصل ۵: مدیریت درخواست‌های HTTP

* ساختار درخواست‌ها و پاسخ‌ها در Mahi
* تعریف مسیرها (Routes)
* استفاده از ماژول‌های HTTP

فصل ۶: کار با کنترلرها

* معرفی کنترلرها
* نوشتن کنترلرهای ساده
* ارسال و دریافت داده‌ها

فصل ۷: ساخت API با ماهی

* تعریف مسیرهای API
* ارسال پاسخ به صورت JSON
* مدیریت خطاها در API

فصل ۸: پروژه‌های عملی

* پروژه اول: ساخت یک وبلاگ ساده
* پروژه دوم: ایجاد یک سیستم مدیریت وظایف (To-Do List)
* پروژه سوم: طراحی API احراز هویت کاربران

فصل ۹: اشکال‌زدایی و بهینه‌سازی

* ابزارهای اشکال‌زدایی در Mahi
* بهبود عملکرد و امنیت
* نکات پیشرفته برای مدیریت پروژه‌های بزرگ

فصل ۱۰: منابع و مراجع

* مستندات رسمی فریمورک Mahi
* منابع یادگیری زبان Lua
* لینک‌های مفید و انجمن‌های آنلاین

# فصل ۱: ماهی چیست

## چرا ماهی ؟

نام "ماهی" الهام گرفته از سادگی، سبکی و روان بودن این فریمورک است. همان‌طور که ماهی در آب به راحتی و آزادانه حرکت می‌کند، این فریمورک هم با استفاده از Lua و ساختار ساده و آزادش، فرایند توسعه وب را روان‌تر و سریع‌تر و آسان‌تر می‌کند.

## آشنایی با فریمورک ماهی

### ماهی چیست ؟

ماهی یک فریمورک وب و مفسر سبک و انعطاف‌پذیر است که با زبان C# توسعه داده شده و به شما این امکان را می‌دهد تا صفحات وب، APIها، و ماژول‌های HTTP را با استفاده از زبان Lua طراحی و مدیریت کنید.

### ویژگی های کلیدی ماهی چیست ؟

قابلیت‌های وب:

پشتیبانی از مدیریت درخواست‌های HTTP، ساخت API و ماژول‌های HTTP .

ساختار ساده و کاربردی:

برخلاف فریمورک‌های سنتی MVC، در ماهی خبری از مدل و ویو نیست، و تمرکز بیشتر بر کنترلرها و ماژول‌هاست.

## چرا Lua در توسعه وب؟

ماهی به کمک [NLua](https://github.com/nlua/nlua) به‌ عنوان یک مفسر داخلی Lua عمل می‌کند، بنابراین شما می‌توانید کد Lua را به‌طور مستقیم اجرا کرده و از آن برای کنترل رفتار صفحات وب استفاده کنید همچنین این زبان بسیار سبک، ساده و سریع است که در ماهی از آن برای تعریف منطق برنامه و طراحی صفحات وب استفاده می‌شود.

## مزایای ماهی برای توسعه‌دهندگان

**ماهی** در مقایسه با فریمورک‌های دیگر مانند  **ASP.NET** و **PHP** مزایای خاصی دارد که به ویژگی‌های منحصر به فرد آن و فلسفه طراحی‌اش مربوط می‌شود. در ادامه به مزایای ماهی نسبت به این فریمورک‌ها می‌پردازیم:

### استفاده از Lua به‌جای زبان‌های پیچیده‌تر

شما می‌توانید به‌سادگی از قدرت C# برای توسعه قسمت‌های دیگر برنامه و از Lua برای منطق کسب‌وکار استفاده کنید، که این ترکیب باعث **انعطاف‌پذیری بیشتر** می‌شود.

### سادگی و قابلیت شخصی‌سازی

برخلاف ASP.NET که معماری MVC را برای ساختاردهی به برنامه‌ها استفاده می‌کند، ماهی فقط از کنترلرها و ماژول‌ها استفاده می‌کند، که این باعث می‌شود کدنویسی و پیکربندی آن برای پروژه‌های کوچک و متوسط **ساده‌تر و سریع‌تر** باشد.

### سرعت و عملکرد

زبان Lua از سرعت بالایی برخوردار است و به همین دلیل **در پروژه‌هایی که نیاز به پردازش سریع دارند، مناسب‌تر** است. همچنین، چون ماهی تنها از Lua برای منطق برنامه استفاده می‌کند، **سرعت بارگذاری و پردازش درخواست‌ها** نسبت به فریمورک‌های سنگین‌تر مانند ASP.NET یا PHP می‌تواند بهتر باشد.

### سبک و کم حجم

رخلاف ASP.NET که به‌طور پیش‌فرض وابستگی‌های زیادی به کتابخانه‌ها و چارچوب‌های دیگر دارد، ماهی فریمورکی سبک و کم‌حجم است که تنها به یک مفسر Lua و .NET نیاز دارد. این ویژگی باعث می‌شود که پروژه‌های مبتنی بر ماهی **کمتر منابع مصرف کنند** و **راه‌اندازی آن سریع‌تر** باشد.

### مناسب برای پروژه‌های خاص و اختصاصی

ماهی برای پروژه‌های خاص یا اختصاصی که ممکن است نیاز به سادگی و **کنترل دقیق بر روی کد** داشته باشند، ایده‌آل است. این ویژگی در صورتی که نیاز به یک **فریمورک بسیار سبک و شخصی‌سازی‌شده** باشد، خیلی مفید خواهد بود.

### پشتیبانی از APIهای سریع و قابل گسترش

ماهی به‌طور طبیعی به شما این امکان را می‌دهد که APIهای RESTful بسازید. در مقابل، در ASP.NET و PHP شما ممکن است نیاز به پیچیدگی‌های بیشتری برای ساخت APIها داشته باشید (مثلاً در ASP.NET، شما باید Web API را تنظیم کنید یا در PHP از فریمورک‌هایی مانند Laravel استفاده کنید.)

### یادگیری سریع

برخلاف ASP.NET یا PHP که ممکن است برای مبتدیان پیچیده به نظر برسند و نیاز به دانشی گسترده از سیستم‌ها و الگوهای مختلف (مانند MVC) داشته باشند، ماهی به دلیل سادگی زبان Lua و معماری ساده‌تر، برای افرادی که تازه‌کار هستند **یادگیری سریع‌تری دارد**.

# فصل 2: نصب و راه اندازی

## پیش‌نیازها

### .NET 8.0 یا بالاتر

همانطور که گفتیم ماهی بر پایه C# و Dotnet توسعه داده شده است، بنابراین نیاز است که .NET 8.0 یا نسخه‌های بالاتر روی سیستم شما نصب باشد.

## دانلود و نصب ماهی

### نصب ماهی در ویندوز

برای نصب ماهی برای سیستم عامل ویندوز میشود از دو راه اقدام کرد:

* 1. دانلود نسخه از پیش کامپایل شده

برای این کار کافی است به آدرس گیت هاب مخزن ماهی به آدرس <https://github.com/mehdika2/Mahi/releases> مراجعه کنید و آخرین نسخه از پیش کامپایل شده را از قسمت Assets دانلود و از حالت فشرده خارج، سپس فایل Mahi.exe را اجرا کنید.

* 1. دانلود سورس کد برنامه و کامپایل کردن آن

در این حالت شما میبایست [گیت](https://git-scm.com/downloads/guis) را روی سیستم خود نصب کنید و با اجرا دستورات زیر، مخزن ماهی و فردین که یک مخزن مورد نیاز ماهی میباشد را نصب کنید:

git clone <https://github.com/mehdika2/mahi>

git clone <https://github.com/mehdika2/fardin>

سپس با دستورات زیر به ترتیب به محل دانلود مخزن بروید و کد ها را کامپایل کنید:

cd mahi

dotnet build

### نصب ماهی در لینوکس

توجه داشته باشید که قبل از شروع شما باید dotnet را روی لینوکس خود نصب داشته باشید، سپس برای نصب ماهی روی لینوکس کافیست با دستورات زیر اقدام به نصب برنامه گیت روی لینوکس کنید:

sudo apt-get update

sudo apt-get install git

و از اینجا به بعد باید همانند [مرحله 2 نصب ماهی در لینوکس](#_نصب_ماهی_در) اقدام به دانلود کد مخزن کنید و آن را کامپایل کنید.

## اجرای اولین پروژه

بعد از دانلود و کامپایل کردن برنامه به مسیر زیر در پوشه ماهی بروید:

bin > Debug > {.net version}   
و یک فایل با نام appconfig.yml ایجاد کنید(این فایل فایل پیکر بندی اصلی ماهی میباشد) و محتوایت زیر را در آن بریزید:

DefaultPages:

- index.htmlua

سپس یک پوشه با نام wwwapp ایجاد کنید (این پوشه حاوی تمام محتویات و فایل های منطقی برنامه تحت وب شما خواهد بود) و در آن یک فایل با نام index.htmlua ایجاد کنید(فایل های با پسوند .htmlua فایل های دارای منطق و کد های html هستند) و محتوایت زیر را در آن بریزید:

$go("Hello world!")

سپس مسیر <http://127.0.0.1:1010> را در مرورگر خود باز کنید.

\* تبریک میگوییم، شما اولین صفحه وب خود را با ماهی ایجاد کردید! \*

# فصل 3: ساختار پروژه در ماهی

## ساختار فایل پیکر بندی (Config)

همانطور که مشاهده کردید، فایل اصلی پیکربندی ماهی یک فایل با پسوند .yml و از نوع YAML میباشد که مانند Json پرکاربرد و مانند INI ساده میباشد.

فایل پیکربندی این اجازه را به شما میدهد تا رفتار ماهی را هرطور که لازم است تایین کنید، این فایل شامل بخش های زیر میباشد:

#### BaseDirectory (پوشه اصلی)

این کلید برای تعریف پوشه اصلی برنامه تنظیم میشود که یک رشته به عنوان مقدار میگیرد.

مقدار پیش فرض: wwwapp

#### DefaultPages (صفحات پیش فرض)

#### این کلید برای تعریف صفحات پیش‌فرض استفاده می‌شود و مقادیری از نوع آرایه‌ای از رشته‌ها می‌پذیرد. زمانی که مسیر / درخواست شود، فریمورک به ترتیب از اولین مقدار آرایه شروع کرده و به دنبال فایل مربوطه می‌گردد. اگر فایل موردنظر پیدا شود(افزودن پسوند فایل ضروری میباشد):

* در صورت داشتن پسوند .html، محتویات فایل به‌طور مستقیم به کاربر ارسال میشود.
* در صورت داشتن پسوند .htmlua، ابتدا منطق برنامه اجرا شده و سپس خروجی تولید شده به کاربر پساخ داده میشود.

و در صورت پیدا نشدن فایل مربوطه، درخواست دهنده با خطای 404 مواجه میشود.

#### ExtentionRequired (ضرورت وجود .htmlua انتهای مسیر درخواست)

این کلید برای تایین این است که آیا در مسیر درخواست که یک به یک فایل منطقی اشاره دارد ، وجود پسوند فایل (.htmlua) ضرورتی دارد یا خیر و یک دو ارزشی یا Boolean به عنوان مقدار میگیرد.

مقدار پیش فرض: true

#### NotExtentionInUrl (ضرورت نبودن .htmlua انتهای مسیر درخواست)

این کلید زمانی تاثیر گذار است که مقدار ExtentionRequired برابر با false باشد و درصورتی که مقدار true به این کلید بدهید، دیگر نباید پسوند فایل منطقی (.htmlua) را در انتهای درخواست یک فایل منطقی قرار دهید.

مقدار پیش فرض: false

#### ConnectionStriongs (رشته های اتصال)

این کلید برای ذخیره سازی رشته های اتصال و استفاده ساده از آنها در کد ها میباشد، مقدار ورودی این کلید باید یک **فهرست نگاشت** یا Hash Map که هر دو مقدار آن باید رشته باشند است.

#### HttpModules (ماژول های درخواست ها)

این کلید برای تعریف ماژول های درخواست تنظیم میشود، فایل های ماژول باید در پوشه .modules در پوشه اصلی برنامه (یا همان wwwapp پیشفرض) با پسوند .lua تعریف شوند و یک نام برای ماژول به عنوان کلید و نام فایل آن را به عنوان مقدار به عنوان یک **فهرست نگاشت** برای مقدار این کلید تعریف کنید.

#### Routes (مسیر ها)

این کلید برای تعریف رفتار مسیریابی برنامه تنظیم میشود، ساختار این کلید از دیگر کلید ها پیچیده تر است و همینطور امکانات جالبتر و بیشتری در اختیار شما قرار میدهد.

این کلید فهرستی از Route به عنوان مقدار میگیرد، ساختار Route دارای کلید های زیر میباشد:

* **Type (نوع):** که دارای سه نوع Static(ثابت)، Dynamic(متغیر) و Regex میباشد.
* **URL (مسیر درخواست):** که یک رشته میباشد که تایین میکند آیا مسیر درخواست شده شامل Route مربوطه میباشد یا خیر.
* **Route (مسیر):** درصورت شامل شدن Route مربوطه، این کلید تعین میکند که پاسخ توسط کدام فایل منطقی یا کد پردازش و پاسخ داده شود.

#### ErrorPages (صفحات تعریف خطا)

این بخش برای مدیریت کدهای وضعیت HTTP که نشان‌دهنده خطا هستند، استفاده می‌شود. اگر پس از پردازش درخواست، کد وضعیت HTTP برگردانده شده متفاوت از 200 (کد موفقیت در پروتکل HTTP) باشد، سیستم به‌طور خودکار کاربر را به صفحه خطای مربوطه هدایت می‌کند. این صفحات خطا به‌صورت سفارشی تعریف شده‌اند و برای هر کد وضعیت می‌توان یک صفحه خاص در نظر گرفت، مقدار این کلید باید از نوع **فهرست نگارشت** باشد که کلید آن عدد و مقدار آن رشته ای حاوی مسیر فایل تعریف شده برای کد وضعیت باشد.

#### FrobbidenPaths (مسیر های غیر مجاز)

این کلید به منظور غیر مجاز شناختن یا مخفی کردن مسیر هایی در سرور تعریف میشود و مقدار آن باید آرایه ای از نوع رشته باشد.

#### DirectoryBrowsing (پیمایش پوشه ها)

درصورتی که مقدار این کلید را صحیح (true) تعریف کنید کاربر میتواند در پوشه های مجاز برنامه وب جابه جا شود و محتویات درون پوشه ها را ببیند، توصیه میشود برای امنیت بیشتر این ویژگی غیر فعال باشد مگر در موارد خاصی که نیاز به دسترسی عمومی به فایل ها وجود دارد،  
مقدار این کلید باید از نوع دو ارزشی باشد.

## ساختار پوشه ها و فایل ها در ماهی

ماهی مانند بسیاری از چهارچوب‌های طراحی وب، یک ساختار ساده و قابل فهم برای مدیریت فایل‌ها و سازماندهی پروژه ارائه می‌دهد. این ساختار به کاربران کمک می‌کند تا به‌سادگی فایل‌های مربوط به منطق برنامه، صفحات وب، تنظیمات و سایر بخش‌های پروژه را مدیریت کنند.

### پوشه اصلی پروژه: wwwapp

پوشه‌ی اصلی که تمام فایل‌های مرتبط با پروژه را شامل می‌شود، به‌صورت پیش‌فرض با نام wwwapp تعریف شده است. نام این پوشه از طریق فایل تنظیمات قابل تغییر است.

### تغییر نام پوشه اصلی در فایل کانفیگ

اگر بخواهید نام پوشه اصلی را تغییر دهید، می‌توانید در فایل پیکربندی (مثلاً appconfig.yml) مقدار مربوط به کلید BaseDirectory را تغییر دهید:

BaseDirectory: myApp

پس از تغییر، فریمورک به‌جای پوشه‌ی پیش‌فرض wwwapp، از پوشه‌ی مشخص‌شده (در اینجا myApp) استفاده می‌کند.

### توضیحات فایل ها

تمام فایل های منطقی و ثابت در این پوشه قرار میگرند که فایل های ثابت با پسوند پیشفرض خود و فایل های منطقی با پسوند .htmlua که تلفیق Html Lua میباشد .

### توضیحات پوشه ها

#### پوشه .controller

این پوشه شامل فایل های مربوط به کنترلر ها است که مسئول مدیریت درخواست هایی که توسط تنظیمات مسیریابی به سمت آنها هدایت میشوند و ارسال پاسخ مناسب هستند. کنترلر ها درخواست های ورودی را پردازش و داده ها یا صفحات مناسب را به کاربر برمیگردانند.

#### پوشه .modules

در این پوشه برای ذخیره ماژول ها برای مدیریت درخواست های HTTP قبل از پردازش استفاده میشود.

ماژول ها باید در فایل appconfig.yml تعریف شوند.

#### پوشه .libraries

از این پوشه برای ذخیره سازی کتابخانه‌های خارجی یا توابع اضافی Lua که در پروژه استفاده میشوند طراحی شده است.

این کتابخانه ها میتوانند از منابع خارجی باشند یا توسط تیم شما توسعه داده شوند.

هدف از ساخت این پوشه، جداسازی کتابخانه ها از کد های اصلی است.

نکته: دسترسی کاربران به این سه پوشه ذکر شده به هیچ وجه ممکن نیست.

## مسیریابی در ماهی

مسیریابی (Routing) یکی از بخش‌های کلیدی در فریمورک ماهی است که تعیین می‌کند هر درخواست (Request) به چه کنترلری ارسال یا به چه مسیری هدایت شده و چگونه پردازش شود. این سیستم ساده و منعطف طراحی شده است تا با استفاده از فایل پیکربندی یا کدنویسی مستقیم، مسیرهای مختلف را مدیریت کند.

پیکربندی مسیریابی در فایل پیکربندی اصلی appconfig.yml و کلید **Routes** انجام میشود که میتواند دارای یک یا بینهایت **Route** (مسیر) باشد.

### ساختار Route

هر مسیر دارای سه خصوصیت اصلی میباشد:

#### Type (نوع)

نوع دارای سه حالت است که نشان میدهد مسیر درخواست شده چه الگویی داشته باشد:

* **Static:** ثابت است و هر مسیر درخواستی که دقیقا شبه مسیر الگوی آن باشد به مسیر هدایت ارسال میشود.
* **Dynamic:** متغیر است و میتوان بخش های مختلف مسیر درخواست را به کمک مسیر الگو جدا کرده و از آن در مسیر هدایت استفاده کرد.

**مثال:** با فرض اینکه مسیر درخواست "/admin/user/delete" و مسیر الگو "/admin/{role}/{action}" باشد، میتوانید کاربر را با این **"مسیر هدایت"** هدایت کنید: "=/{role}/{action}.lua".

* **Regex:** این نوع مسیر الگو را به الگوی رجکس تبدیل میکند.

#### URL (مسیر الگو)

مسیر الگو مسیری است که با توجه به نوع Route که در بالا توضیح داده داده اگر شبیه مسیر درخواست بود **مسیر هدایت** عمل میکند.

#### Route (مسیر هدایت)

مسیر هدایت در صورتی که با توجه به نوع، مسیر الگوی آن شبیه مسیر درخواست باشد عمل میکند و میتواند درخواست کاربر را توسط یک فایل منطقی پردازش یا به یک مسیر هدایت کند به اینصورت که:

اگر در ابتدای مسیر هدایت علامت "=" بگذارید، علامت به مسیر کنترلر ها تبدیل میشود و اینگونه شما میتوانید مسیر درخواست ها را به سمت کنترل ها هدایت و توسط آنها پردازش کنید و پاسخ دهید.

برای هدایت یا به اصطلاح Redirect کردن کاربر به یک مسیر یا URL کافی است قبل از مسیر از علامت ">" استفاده کنید، اینگونه ماهی متوجه میشود که شما قصد هدایت کردن کاربر به مسیر دیگری را دارید.

یا برای ارسال یک فایل از آن طریق کافی است تنها مسیر آنرا قرار دهید.

## ساختار فایل HTMLUA در ماهی

Htmlua یک فرمت ترکیبی در فریمورک ماهی است که به شما اجازه می‌دهد کد Lua را در کنار HTML بنویسید. این فرمت شباهت بسیاری به ساختار Razor در ASP.NET دارد اما با تغییراتی ساده و قابل فهم طراحی شده است. مهم‌ترین تغییر، استفاده از علامت **$** به جای **@** است که به‌عنوان پیشوند دستورات Lua در قالب HTML عمل می‌کند.

### یک فایل Htmlua شامل بخش‌های زیر است:

**HTML خام:** برای طراحی رابط کاربری.

**دستورات Lua:** برای اضافه کردن منطق پویا به قالب.

**ادغام Lua و HTML:** با استفاده از **$** برای جایگذاری یا اجرای کد Lua.

### چگونه از کدهای Lua در HTML استفاده کنیم؟

در Htmlua، زمانی که بخواهید کد Lua را داخل یک فایل HTML بنویسید، باید از علامت $ استفاده کنید. این علامت به فریمورک **ماهی** می‌گوید که کدی که بعد از آن قرار دارد، یک دستور Lua است.

اگر بخواهید چندین خط کد منطقی بنویسید، کافی است بعد از علامت $ یک آکولاد باز { قرار دهید. با این کار شما یک بلوک کد Lua را شروع کرده‌اید و تمام متنی که داخل آن بنویسید به‌عنوان کد Lua شناسایی می‌شود.

اما اگر بخواهید در این بلوک‌های کد از تگ‌های HTML استفاده کنید، **ماهی** این امکان را به‌صورت هوشمند برای شما فراهم می‌کند. کافی است که تگ‌های HTML خود را بدون تغییر بنویسید. **ماهی** به‌طور خودکار تشخیص می‌دهد که تگ‌های HTML در وسط کد Lua قرار دارند.

اگر تگ HTML شما جفتی باشد (به عنوان مثال <div> که باید با </div> بسته شود)، **ماهی** به‌طور هوشمند انتهای آن را پیدا کرده و کد Lua را بعد از آن ادامه می‌دهد. به این ترتیب، شما می‌توانید به راحتی HTML و Lua را در یک فایل ترکیب کنید بدون اینکه نیاز به نگرانی در مورد بسته شدن تگ‌ها داشته باشید.

### نمونه‌ی کامل

بیایید یک نمونه‌ی کامل را بررسی کنیم که از همه این موارد استفاده می‌کند:

<html>

<head>

<title>نمونه Htmlua</title>

</head>

<body>

<h1>خوش آمدید!</h1>

<!-- شرط: اگر کاربر وارد شده باشد -->

${

**if** user.isAuthenticated **then**

<p>سلام، $user.name!</p>

**else**

<p>لطفاً وارد شوید.</p>

end

}

<!-- استفاده از تابع -->

<p>جمع دو عدد: $add(3, **5**)</p>

</body>

</html>

# فصل ۴: توابع داخلی ماهی

توابع داخلی ماهی ابزارهای کاربردی هستند که در فریمورک Mahi برای ساده‌سازی برخی از کارهای رایج و پیچیده توسعه وب فراهم شده‌اند. این توابع به توسعه‌دهندگان کمک می‌کنند تا بسیاری از عملیات‌های معمول را بدون نیاز به کدنویسی دستی و پیچیده انجام دهند. در ادامه، به توضیح توابع داخلی اصلی Mahi خواهیم پرداخت:

#### go

تابع go برای افزودن یک رشته به محتوای پاسخ استفاده میشود. این تابع میتواند برای اضافه کردن محتوای پویا به صفحه وب در زمان پردازش کردن استفاده شود.

#### safe

مانند تابع go عمل می‌کند، با این تفاوت که در اینجا محتوای اضافه شده به‌صورت امن در HTML انکد می‌شود تا از حملات XSS جلوگیری شود. این تابع برای جلوگیری از تزریق کدهای مخرب به محتوای وب استفاده می‌شود.

#### setTemp

با استفاده از setTemp می‌توانید داده‌ای را به‌طور موقت ذخیره کنید که در درخواست بعدی قابل دسترسی باشد. این داده‌ها معمولاً برای نگهداری اطلاعات موقتی بین درخواست‌ها استفاده می‌شوند.

#### getTemp

این تابع برای دریافت داده‌هایی است که قبلاً با استفاده از setTemp ذخیره شده‌اند.

#### setItem

تابع setItem متغیر ورودی را تا زمان پایان درخواست در حافظه نگهداری میکند و به کمک آن میشود محتوا را به ماژول ها و فایل های کنترل یا htmlua فرستاد.

#### getItem

این تابع برای دریافت داده‌های ذخیره شده با استفاده از setItem استفاده می‌شود.

#### Log

این تابع برای ثبت لاگ‌ها در فایل استفاده می‌شود. می‌توانید از این تابع برای ذخیره‌سازی اطلاعات خطاها یا اطلاعات دیگر در فایل‌های لاگ استفاده کنید.

#### setStatus

این تابع برای تغییر کد وضعیت پاسخ HTTP استفاده می‌شود. به عنوان مثال، می‌توانید وضعیت پاسخ را به 404 برای "یافت نشد" یا 500 برای "خطای سرور" تنظیم کنید.

#### redirect

این تابع برای هدایت کاربر به یک URL دیگر استفاده می‌شود. معمولاً در صورت نیاز به هدایت کاربر به صفحه‌ای دیگر پس از انجام یک عملیات (مانند ثبت‌نام یا ورود به سیستم) از آن استفاده می‌شود.

#### addHeader

این تابع برای اضافه کردن هدرهای HTTP به پاسخ استفاده می‌شود. می‌توانید از آن برای اضافه کردن هدرهای امنیتی، کش، یا هر هدر دیگری که نیاز دارید استفاده کنید.

#### setCookie

این تابع برای تنظیم یک Cookie در پاسخ HTTP استفاده می‌شود. می‌توانید از آن برای ذخیره اطلاعات در مرورگر کاربر استفاده کنید.

#### deleteCookie

این تابع برای حذف یک Cookie از مرورگر کاربر استفاده می‌شود.

#### isNullOrEmpty

این تابع برای بررسی اینکه آیا یک متغیر یا رشته تهی یا خالی است یا نه استفاده می‌شود.

#### getError

این تابع برای دریافت آخرین خطای بوجود آمده در برنامه استفاده میشود. میتوانید از آن برای نمایش خطاهای خاص در صفحه استفاده کنید.

#### clearError

این تابع برای پاک کردن خطای بوجود آمده در برنامه از حافظه استفاده میشود.

#### match

این تابع برای انجام عملیات Regex و تطبیق الگو با رشته ها استفاده میشود.

#### matchs

این تابع برای دریافت گروه های مختلف تطبیق در یک عملیات Regex استفاده میشود.

#### base64\_encode

این تابع برای کدگذاری داده ها از نوع آرایه بایت به فرمت Base64 استفاده میشود، معمولاً برای ارسال داده های باینری در قالب متنی مفید است.

#### base64\_decode

این تابع برای رمزگشایی داده های Base64 به آرایه بایت استفاده میشود.

#### utf\_encode

این تابع برای کدگذاری رشته متنون Unicode به آرایه بایت از نوع UTF-8 استفاده میشود.

#### utf\_decode

این تابع برای رمزگشایی آرایه بایت و تببدیل آن به رشته Unicode استفاده میشود.

# فصل ۵: مدیریت درخواست‌های HTTP

## ماژول های HTTP

### کاربرد ماژول ها

ماژول های HTTP در ماهی با هدف گسترش قابلیت های ماهی و انجام وظایف مختلف در قبل از پردازش درخواست ها ایجاد شده‌اند. ماژول ها در ماهی این امکان را به شما میدهند تا کدهایی که برای انجام عملیات هایی مانند احراز هویت، مجوزدهی، پردازش داده ها و بسیاری دیگر از قالبیت ها را به راحتی در پروژه خود اضافه کنید.

### ایجاد ماژول

مراحل ایجاد ماژول در ماهی به شرح زیر میباشد:

* 1. ایجاد ماژول: در این مرحله شما باید ماژول خود و کد های منطقی آن را در یک فایل با پسوند .lua در پوشه .modules در پوشه برنامه که به صورت پیشفرض wwwapp تعریف شده ذخیره کنید.
  2. ثبت نام ماژول: برای آنکه ماژول طراحی شده شروع به کار کند ابتدا باید توسط ماهی شناسایی شود، برای این کار باید یک نام و نام فایل یا آدرس ماژول در پوشه .modules را به عنوان مسیر ماژول به کلید HttpModules فایل پیکربندی برنامه اضافه کنید.

### نحوه کار ماژول

ماژول ها قبل از هر مسیر یابی یا ارسال یا پردازش فایل اجرا میشوند و وظایف آنها توسط برنامه نویس مشخص میشود، ماژول ها میتوانند یک متغیر دو عرضشی یا Boolean را بازگردانند که درصورتی که مقدار صحیح (true) توسط یکی از ماژول ها برگردانده شود، ماهی از ادامه پردازش خودداری میکند و پاسخ ایجاد شده تا اینجا را به سمت درخواست دهنده باز میگرداند، این کار کمک میکند تا اگر درخواست صادره به هر دلیلی مجاز به مشاهده ادامه درخواست نبود جلوی آن گرفته شود.

# فصل ۶: کار با کنترلرها

# فصل ۷: ساخت API با ماهی

# فصل ۸: پروژه‌های عملی

# فصل ۹: اشکال‌زدایی و بهینه‌سازی