



# Ambiorix

Build for the web, the way the  
web works

Creator: John Coene, [john@opifex.org](mailto:john@opifex.org)

Maintainer: Kennedy Mwavu, [mwavukennedy@gmail.com](mailto:mwavukennedy@gmail.com)

# Introduction to Ambiorix

## Web Framework for R

Ambiorix lets you build web applications and APIs entirely in your favorite programming language: R, giving you direct access to HTTP mechanisms.

# Introduction to Ambiorix

## Web Framework for R

Ambiorix lets you build web applications and APIs entirely in your favorite programming language: R, giving you direct access to HTTP mechanisms.

## Inspired by Express JS

Following the principles of the popular Node.js framework, Ambiorix brings request-response architecture to the R ecosystem.

- ❏ A web framework for R developers to build multi-page applications and APIs with the full power of HTTP.

# Is Ambiorix Shiny 2.0?

**NO.**

Ambiorix isn't a Shiny replacement or upgrade. It's a fundamentally different approach to web development in R, based on the HTTP request-response cycle rather than reactivity.





# My Journey to Ambiorix



# The Problem

After building huge Shiny applications, I hit limitations:

- **Multi-page navigation** - Needed proper routing and shareable links
- **API endpoints** - Clients wanted to build their own frontends



# The Dilemma

Didn't want to leave the R ecosystem, but needed web functionality beyond Shiny's capabilities



# The Solution

Discovered Ambiorix - an R web framework that handles both multi-page apps and APIs!

*"And he lived happily ever after"*



# Why should you care?

## True multi-page support

Build applications with multiple pages and proper routing out of the box.

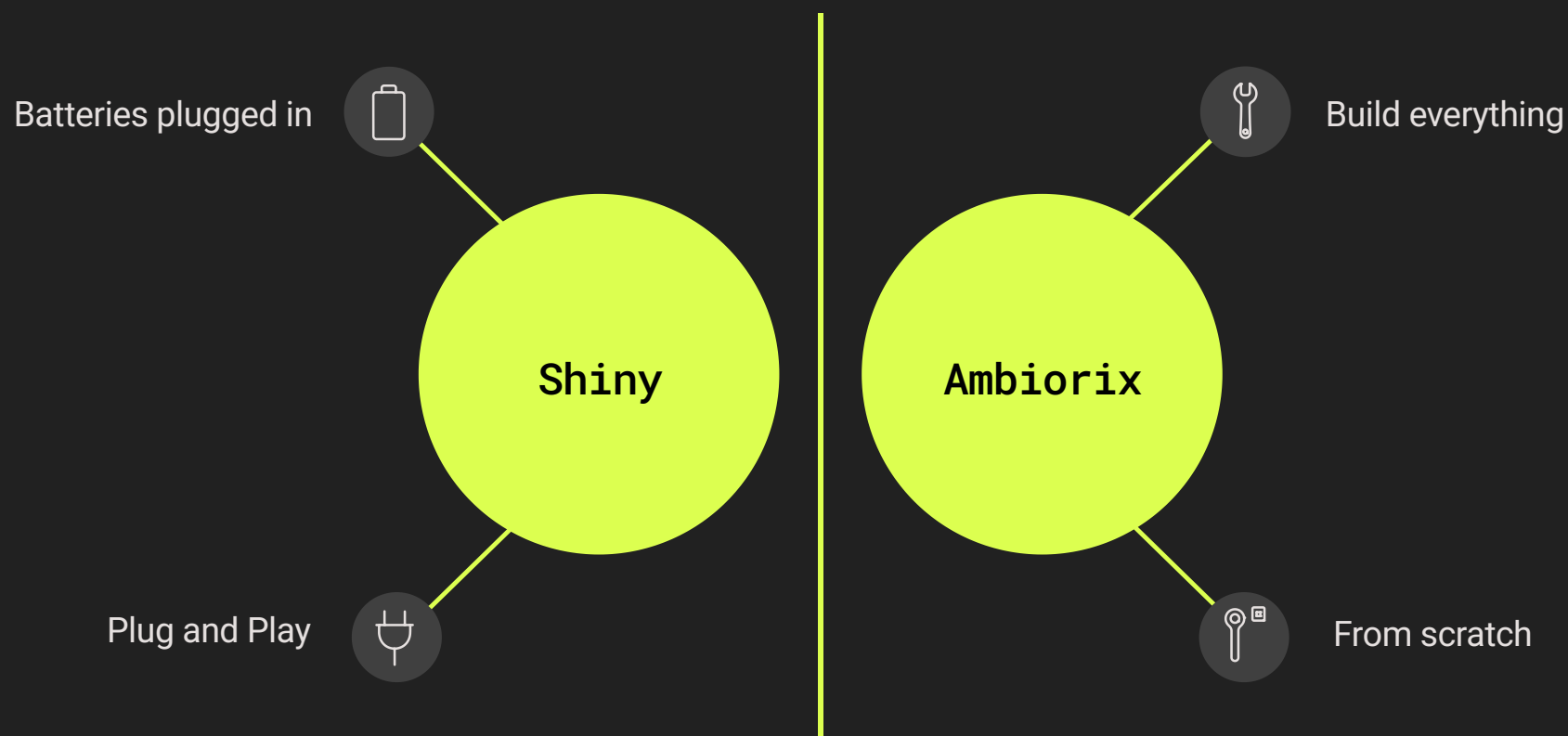
## API capabilities

Create JSON endpoints alongside HTML pages using the same syntax.

## Fine-grained HTTP control

Direct access to request and response objects gives you complete flexibility.

# Shiny vs. Ambiorix: A Comparison



# Shiny vs. Ambiorix: A Comparison

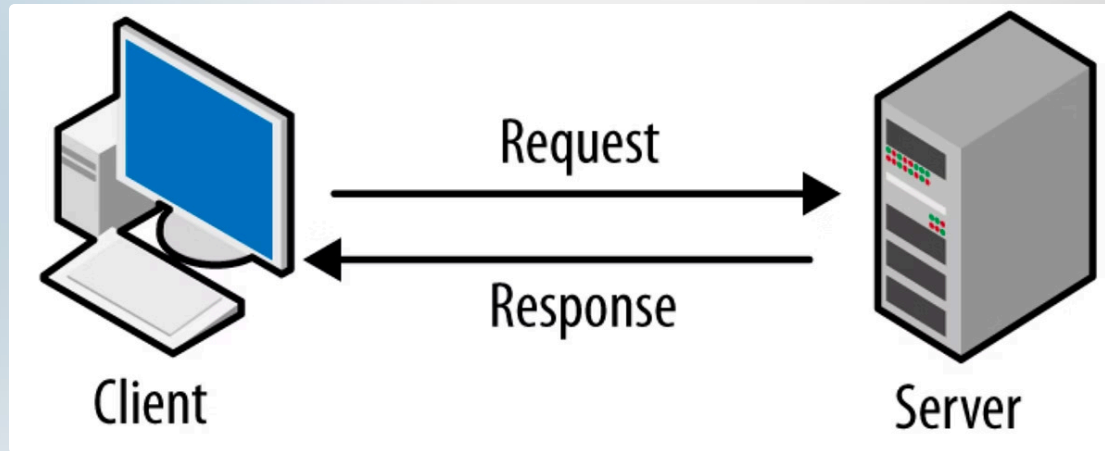
Shiny	
Single Page Applications	
Pre-built UI components	
Bootstrap & jQuery by default	
Reactive framework built-in	
WebSockets by default	

# Shiny vs. Ambiorix: A Comparison

Shiny	Ambiorix
Single Page Applications	Multi-Page Applications
Pre-built UI components	No pre-built components (build from scratch)
Bootstrap & jQuery by default	Use whatever frontend libraries you wish
Reactive framework built-in	No reactivity by default
WebSockets by default	HTTP requests by default

# Best & Worst Thing





# The Request-Response Cycle

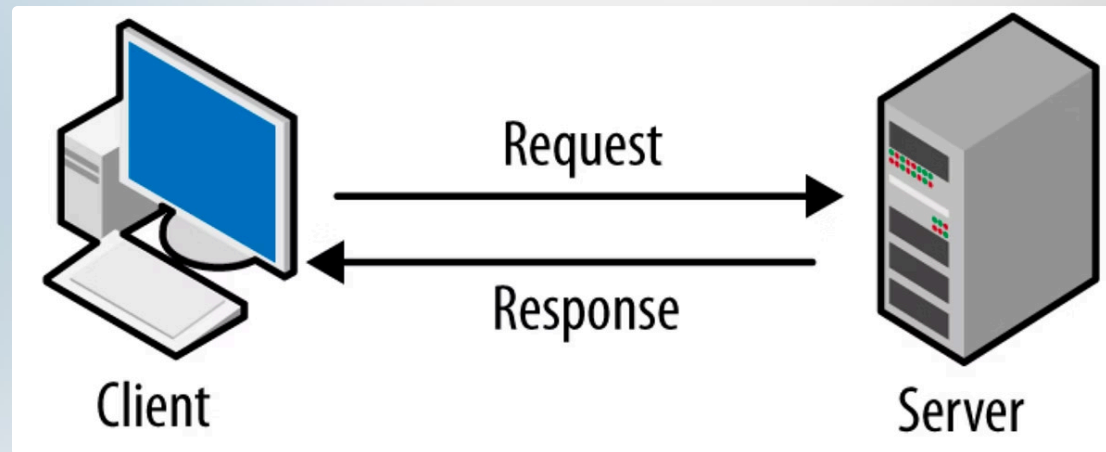
Every interaction on the web, from loading a page to submitting a form, follows this fundamental cycle:



## Client Initiates

Your web browser or application sends a request to a server.

This continuous **HTTP Cycle** forms the backbone of how web applications, including those built with Ambiorix, communicate.



# The Request-Response Cycle

Every interaction on the web, from loading a page to submitting a form, follows this fundamental cycle:

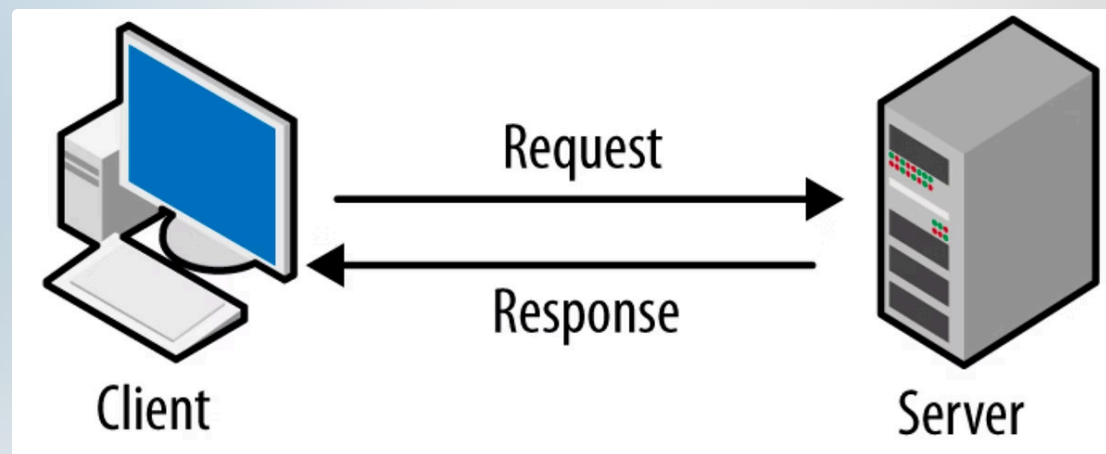
## Client Initiates

Your web browser or application sends a request to a server.

## The Request

A message detailing what the client wants (e.g., a specific web page, data) is sent.

This continuous **HTTP Cycle** forms the backbone of how web applications, including those built with Ambiorix, communicate.



# The Request-Response Cycle

Every interaction on the web, from loading a page to submitting a form, follows this fundamental cycle:

## Client Initiates

Your web browser or application sends a request to a server.

## The Request

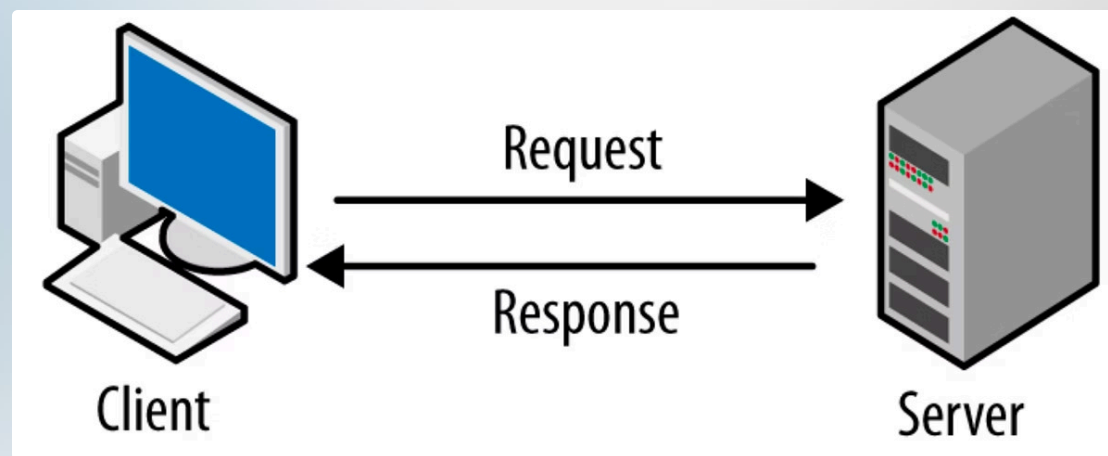
A message detailing what the client wants (e.g., a specific web page, data) is sent.

## Server Processes

The server receives the request, processes it, and prepares a response.

This continuous **HTTP Cycle** forms the backbone of how web applications, including those built with Ambiorix, communicate.





# The Request-Response Cycle

Every interaction on the web, from loading a page to submitting a form, follows this fundamental cycle:

## Client Initiates

Your web browser or application sends a request to a server.

## The Request

A message detailing what the client wants (e.g., a specific web page, data) is sent.

## Server Processes

The server receives the request, processes it, and prepares a response.

## The Response

The server sends back the requested data, an HTML page, or an action confirmation.

This continuous **HTTP Cycle** forms the backbone of how web applications, including those built with Ambiorix, communicate.

# Data Dashboard Demo

## Explore R datasets

mtcars, iris, airquality

## HTML interface for humans

Beautiful tables and  
visualizations

## JSON API for machines

Programmatic access to the  
same data

[Live Demo Time!]

# Hello World: Request → Response

## The cycle in action:

01

---

Browser requests GET /

02

---

Handler function receives req, res objects

03

---

We send response with res.send()

hello-world.R

R

```
library(ambiorix)

app <- Ambiorix$new()

app$get("/", function(req, res) {
  res$send("Hello from Ambiorix!")
})

app$start(port = 3000)
```

# Brief Overview of HTTP Methods



GET



POST



PUT



DELETE

Ambiorix gives you handlers for all standard HTTP methods, making it perfect for RESTful APIs.

# Brief Overview of HTTP Methods



## GET

Retrieve a resource from the server

```
app$get("/data", handler)
```

# Brief Overview of HTTP Methods



## POST

Create a new resource on the server

```
app$post("/data", handler)
```

# Brief Overview of HTTP Methods



## PUT

Update an existing resource

```
app$put("/data/:id", handler)
```

# Brief Overview of HTTP Methods



## DELETE

Remove a resource from the server

```
app.delete("/data/:id", handler)
```



# Let's Now Talk About The Demo App

Time to explore the demo application. We'll break down its core components and the Ambiorix concepts behind them.

## Concepts & Core Ideas



Serve  
Multiple  
Formats



Request  
Parameters



Query  
Parameters



HTTP Error  
Responses

# Same Data, Different Formats

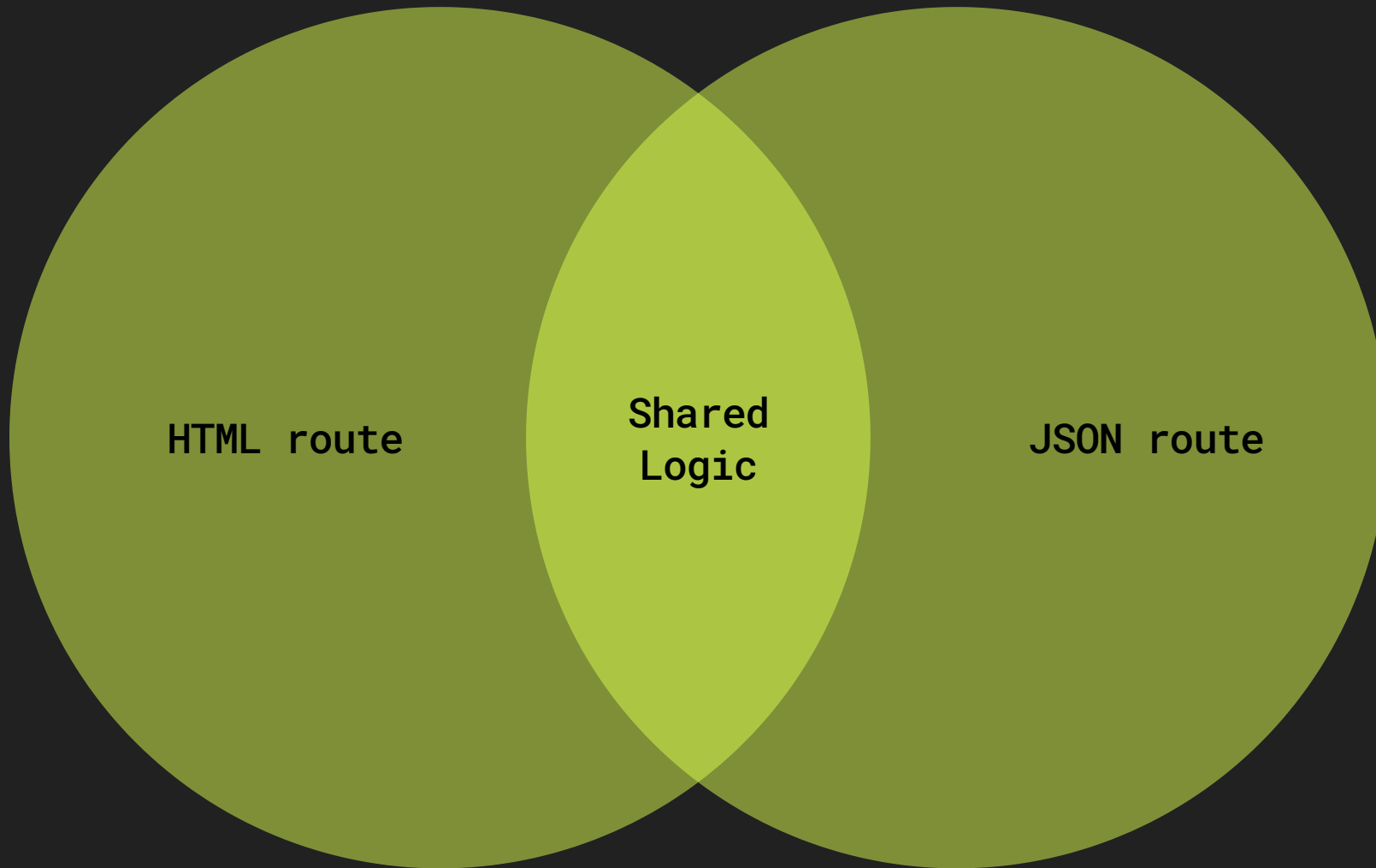
## Two Interfaces:

- **Browser:** Beautiful HTML tables and charts
- **API:** Raw JSON for programmatic access

## One Core R Function:

```
get_dataset_summary("mtcars")  
# Returns: list with statistics,  
# sample data, etc.
```

# Same Data, Different Formats



**Key insight:** Write business logic once, serve it multiple ways!

# Same Data, Different Formats

app.R

R

```
# ----HTML route----
```

```
app$get("/datasets/:name", function(req, res) {  
  data <- get_dataset_summary(req$params$name)  
  res$send(create_html_page(data)) # Returns HTML  
})
```

```
# ----JSON route----
```

```
app$get("/api/datasets/:name/summary", function(req, res) {  
  data <- get_dataset_summary(req$params$name) # Same function!  
  res$json(data) # Returns JSON  
})
```

# Route Parameters

1

**`/datasets/mtcars`**

Motor Trend Car Road Tests

2

**`/datasets/iris`**

Edgar Anderson's Iris Data

3

**`/datasets/airquality`**

New York Air Quality

# Route Parameters

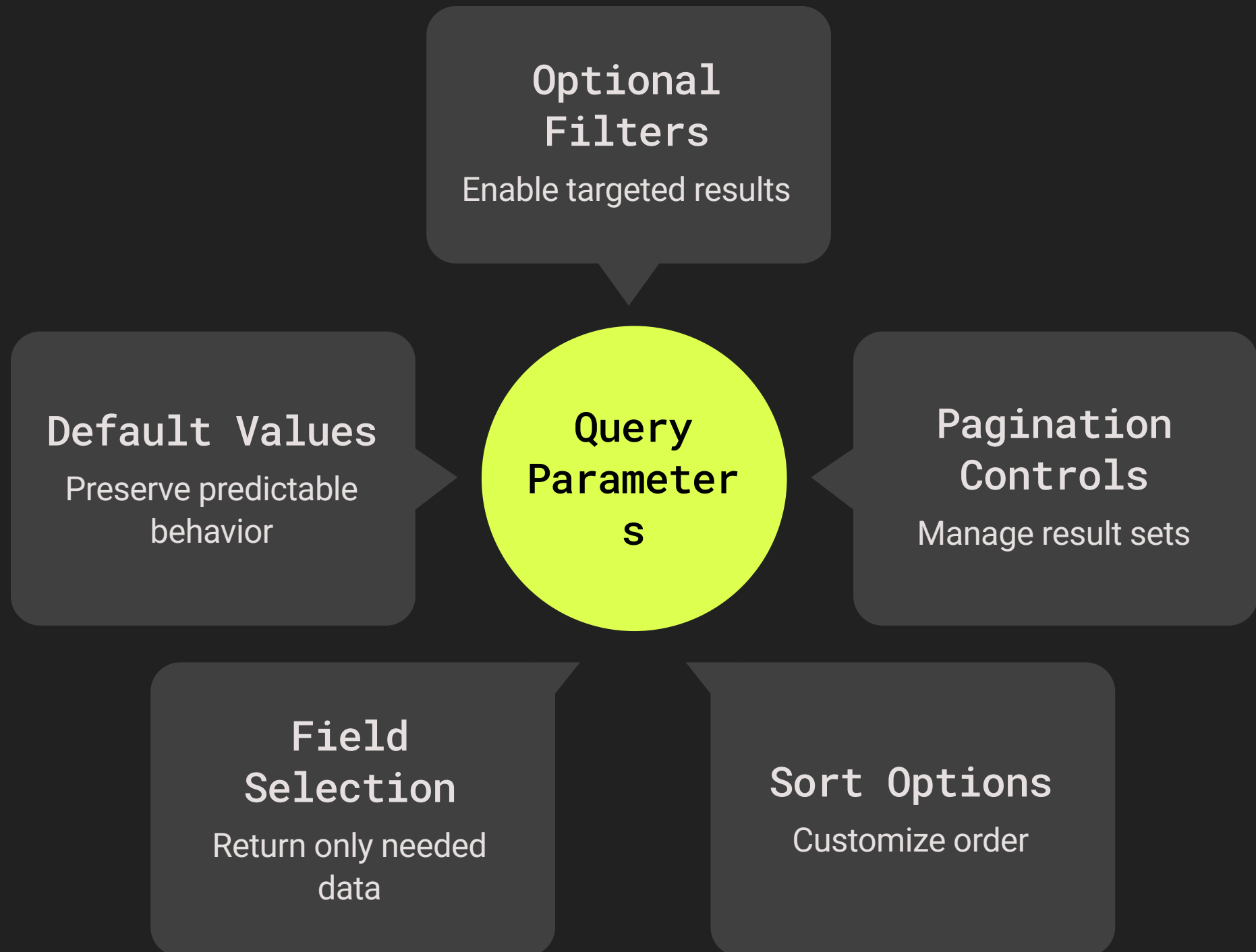
app.R

R

```
app$get("/datasets/:name", function(req, res) {  
  dataset_name <- req$params$name  
  summary <- get_dataset_summary(dataset_name)  
  
  res$send(create_dataset_page(summary))  
})
```

Creating **dynamic URLs** is effortless!

# Query Parameters



# Query Parameters

## Example API Calls:

- `/api/datasets/mtcars/data` → All rows
- `/api/datasets/mtcars/data?limit=5` → First 5 rows
- `/api/datasets/iris/data?limit=10` → First 10 rows

**Standard web patterns** that developers expect



# Query Parameters

app.R

R

```
app$get("/api/datasets/:name/data", function(req, res) {  
  dataset_name <- req$params$name  
  limit <- req$query$limit # ?limit=10  
  
  data <- get_dataset_data(dataset_name, limit)  
  res$json(data)  
})
```

# Common HTTP Error Responses

Understanding and communicating errors effectively is crucial for building robust web applications. Ambiorix allows precise control over these responses.

## ✔ Proper HTTP status codes & helpful error messages

Ambiorix enables you to set custom status codes and provide helpful error messages, guiding clients in debugging and ensuring proper web standards are followed.

# Common HTTP Error Responses

400 Bad Request

404 Not Found

500 Internal Server Error

# Common HTTP Error Responses

## 400 Bad Request

The server cannot process the request due to client error, such as malformed syntax or invalid request parameters.

# Common HTTP Error Responses

## 404 Not Found

The requested resource could not be found on the server. This indicates a valid request for an unknown URL.

# Common HTTP Error Responses

## 500 Internal Server Error

A generic error message indicating an unexpected condition or server-side issue that prevented the fulfillment of the request.

# HTTP Error Responses

app.R

R

```
api_dataset_summary_get <- function(req, res) {  
  dataset_name <- req$params$name  
  
  if (!is_valid_dataset(dataset_name)) {  
    response <- list(  
      error = "Dataset not found",  
      message = sprintf(  
        "Dataset '%s' not found. Available datasets: mtcars, iris, airquality",  
        dataset_name  
      )  
    )  
  
    res$status <- 404L  
    return(res$json(response))  
  }  
  
  summary_data <- get_dataset_summary(dataset_name)  
  res$json(summary_data)  
}
```

# Try Ambiorix Today



GitHub

[ambiorix-web/ambiorix](https://github.com/ambiorix-web/ambiorix)



Website

[ambiorix.dev](https://ambiorix.dev)



Demo Code

[ambiorix-web/positconf2025](https://ambiorix-web/positconf2025)

## Get started:

```
install.packages("ambiorix")  
# Run the demo from /demo folder
```



# Thanks!

## Questions?



Creator: John Coene (2020)



Maintainer: Kennedy Mwavu