

Unlock Your Potential and Master Web Development

PROGRAMMING STEP BY STEP AND MORE

AN INSTRUCTIVE GUIDE

BOOK 6

Full-Stack Engineering with JavaScript:
From Modern Frontend to Modular
Architecture and Cloud Deployment.

**JUMP TO THE BACKEND
HIDDEN ENGINEERING**

FIRST MODULE: From Frontend
Development to Modern Service-Based
Architecture.

René F. Ruano Domínguez

Copyright © 2026 René F. Ruano Domínguez
All rights reserved.

DEDICATION

TO THE GENERATIONS OF THE FUTURE, MILLENNIALS, BABY BOOMERS AND BEYOND.

This How-To is dedicated to everyone: To you, the brilliant young minds shaping the world with your ingenuity and creativity. To the millennials, who navigate the digital age with ease and constantly seek new ways to innovate. To the baby boomers, who pioneered the technological revolution and now contribute their experience and wisdom. To the retirees who dream of staying up-to-date, filling their free time with useful hours, and exercising their minds.

OTHER BOOKS BY THE AUTHOR

If you would like to continue exploring the Programming Step by Step and More series, you can access all related volumes, updates, and additional books available on Amazon through the QR code below.

All books in this series are available on Amazon.

Scan the QR code to discover more:



PROLOGUE

This Sixth Book, titled **"Full-Stack Engineering with JavaScript: From Modern Frontend to Modular Architecture and Cloud Deployment,"** marks a turning point in the "Programming Step-by-Step and More" series. It represents not only the culmination of a progressive learning journey but also a significant contribution to contemporary software development and an up-to-date understanding of the technologies underpinning modern programming. This volume marks the transition from visible frontend code to the deep, underlying logic of the backend, expanding language learning into a comprehensive view of software engineering as applied to real-world systems and professional environments.

Given its broad scope and highly specialized content, Book 6 is presented in two complementary modules that structure the learning process in a clear, progressive manner:

a) The first module, **"From Frontend Development to Modern Service-Based Architecture,"** guides the reader from building dynamic interfaces to understanding the principles underpinning distributed applications and API-based systems.

b) The second module, **"Modern Backend with JavaScript: Data Persistence, REST APIs, Modular Architecture, and Cloud Deployment,"** delves into server-side engineering, covering everything from professional data management to the construction of scalable architectures and their deployment on state-of-the-art cloud platforms.

More than just a technical volume, this book is the result of a continuous journey of knowledge, practice, and critical reflection regarding the construction of modern applications. Behind these pages lie years of study, experimentation, and systematic work with JavaScript, coupled with the conviction that knowledge attains its greatest value when shared with clarity, rigor, and a pedagogical approach. This philosophy, which runs through the entire series, is solidified in this sixth book as a mature expression of the author's commitment to teaching and the structured transmission of technical expertise.

This volume reaffirms an educational approach that transcends the mere presentation of tools. Its purpose is to cultivate professional judgment: the ability to analyze problems, design coherent solutions, structure scalable systems, and make responsible decisions within real-world development contexts. Rather than simply teaching how to program, it seeks to teach how to think using engineering logic.

In this regard, the book guides the reader through the transition toward a software engineering mindset—one that is critical, methodical, and creative, and committed to quality, security, and ethics in technological development. Each chapter is designed to reinforce this perspective by integrating practice, analysis, and technical reflection.

Driven by this conviction, we present **Book 6 of the Programming Step by Step and More series**. This work marks the transition from writing code to building systems, transforming accumulated knowledge into solid professional expertise and preparing the reader for the challenges of modern development in the era of Full Stack and cloud computing.

With gratitude and admiration for the author:

Idalmy Baluja Conde

CONTENT

ID	TITLE	PAG
1	DEDICATION	iii
2	OTHER BOOKS BY THE AUTHOR	iv
3	PROLOGUE	v
4	GRATITUDE	xv
5	BOOK 6 — FULL STACK ENGINEERING WITH JAVASCRIPT	1
6	INTRODUCTION – FIRST MODULE	3
7	JSON. DATA PROCESSING	5
8	JSON – Brief introduction	5
9	JSON – Structure, Serialization, and Deserialization	5
10	Basic JSON Structure	5
11	Serialization (Object → JSON)	6
12	Deserialization (from JSON to object)	7
13	Importance of JSON in modern systems	8
14	JSON vs Objetos JavaScript	8
15	JSON in APIs and storage	10
16	JSON as Frontend → Backend communication	10
17	Validation of JSON in Backend (PHP)	12
18	Frontend Responses	13
19	Handling HTTP codes (200, 400, 500)	14
20	Solutions for storing JSON	17
21	LocalStorage	17
22	JSON in modern databases	18
23	Conclusions of JSON Chapter	18
24	TEMPLATES	20
25	Purpose of the Chapter	20
26	Template Literals	21
27	Dynamic Rendering	22

ID	TITLE	PAG
28	Basic operation of dynamic rendering	23
29	Dynamic rendering using arrays	23
30	Importance of rendering in SaaS systems	24
31	Evolution towards modern frameworks	25
32	Key Principles in Templates	26
33	Conclusions	30
34	ASYNCHRONY AND SYNCHRONY IN JAVASCRIPT	31
35	Fetch	32
36	Try Block	38
37	Functions of the try block	38
38	Analysis of try block examples	39
39	APIs	40
40	Conclusions	41
41	PROMISES IN JAVASCRIPT	42
42	Definitions and characteristics	42
43	Basic Creation of a Promise	43
44	How promises work	44
45	Checking username availability	45
46	Access-Verify-Server Project	46
47	Scripts involved in execution flow	47
48	User Verification Module	54
49	Chaining of Promises	54
50	Operation of chaining	55
51	Advantages of chaining	56
52	Relationship with fetch() and APIs	56
53	Modern Error Handling	56
54	Error handling with catch()	57
55	Error handling in architecture	59

ID	TITLE	PAG
56	async/await	59
57	Asynchronous function	60
58	Modern error handling with async/await	63
59	async/await and distributed systems	64
60	Scaling Access-verify-server	64
61	Structure of access-promises-chain	65
62	Scripts of new files	66
63	Conclusions	72
64	COMMUNICATION WITH SERVERS	73
65	Communication process	74
66	Distributed Communication	75
67	APIs as intermediaries	76
68	Modern Asynchronous Communication	77
69	JSON as universal language	77
70	Dynamic Rendering	78
71	Towards decoupled architecture	78
72	Anatomy of HTTP Communication	79
73	HTTP Request	79
74	Server Response	80
75	HTTP Methods	80
76	URLs	81
77	Endpoints	82
78	Query Parameters	82
79	HTTP Headers	83
80	HTTP Status Codes	84
81	Complete Technical Flow	84
82	Asynchronous Communication	85
83	Asynchronicity	87

ID	TITLE	PAG
84	Promises	87
85	fetch()	88
86	Distributed Flow	89
87	await	89
88	Dynamic Rendering	90
89	Error Handling	90
90	Promise.all()	90
91	Customized Events	91
92	JSON and Distributed Communication	91
93	Rest APIs Decoupled Architecture	92
94	Decoupled Frontend	93
95	REST APIs	94
96	Query Parameters	94
97	Backend API-First	95
98	Backend as Service Provider	95
99	JSON as Universal Language	96
100	Service-Based Architecture	97
101	Headers and CORS	97
102	SaaS and Cloud-Native Architecture	98
103	Microservices	98
104	PWAs	99
105	Towards Modern Architecture	99
106	Security, Authentication and Protection	100
107	Safety in Modern Architecture	100
108	API Keys	101
109	Risks of exposing keys	101
110	Best Practices with API Keys	102
111	Tokens and Authentication	103

ID	TITLE	PAG
112	Bearer Authentication	103
113	Authorization Headers	104
114	Endpoint Protection	104
115	CORS and Distributed Security	105
116	Security in SaaS	105
117	Safety and Decoupling	106
118	Complete Frontend ↔ Backend Flow	106
119	General Flow	108
120	Forms	108
121	Data Capture and Validation	109
122	Sending POST	109
123	POST Method	110
124	JSON as Exchange Mechanism	110
125	Headers	110
126	Local Backend	111
127	Response JSON	111
128	Response Processing	112
129	Dynamic Rendering	112
130	Backend as Decoupled API	113
131	Distributed Systems	113
132	Towards Professional Systems	114
133	Conclusions	114
134	MULTI USER – CONTROLLED TEST	116
135	Local server installation	116
136	Wampserver	116
137	Installation process	117
138	User manuals	119
139	Local development of access control	120

ID	TITLE	PAG
140	New Generation Template	129
141	Main characteristics	130
142	Armored Pro LMS architecture	130
143	Project structure	131
144	Asset organization	131
145	Main scripts	132
146	Access Control — Multi User	159
147	Users Table	160
148	LOG Table	161
149	Registration Form	162
150	Access Form	163
151	Strategic substitute for CAPTCHA	163
152	Authentication flow	163
153	System structure	164
154	Backend functionalities	165
155	register-user.php	166
156	login-user.php	166
157	Frontend functionalities	167
158	saas-login.js	167
159	saas-register.js	168
160	Universal Access System	168
161	Conclusions	169
162	GENERAL CONCLUSIONS	171
163	REFERENCES	173
164	MOTIVATIONAL PHRASES	177
165	ABOUT THE AUTHOR	179

GRATITUDE

To my family, my colleagues, my friends.

Thank you so much for the unwavering support you've given me since I started this project. And to those who didn't believe in it, I thank you for motivating me to prove that it was worthwhile.

Without the encouragement, suggestions, trust they placed in me, and the personal challenge to create a useful product, this project would not have been possible.

BOOK 6

**Full-Stack Engineering with JavaScript:
From Modern Frontend to Modular
Architecture and Cloud Deployment.**

First Module:

**From Frontend Development to
Modern Service-Based Architecture.**

Rene F. Ruano Domínguez

I. BOOK 6 — FULL STACK ENGINEERING WITH JAVASCRIPT: FROM MODERN FRONTEND TO PROFESSIONAL BACKEND AND SYSTEM SECURITY.

Book 6 comprises a training package made up of three complementary modules, designed to guide the student from modern frontend development to professional service-based architecture, JavaScript-based backend development, and quality and security engineering. Each module is published as a standalone edition to balance content volume, reduce costs for the student-reader, and maximize the practical value added to their professional profile and quality of life.

The First Module, "From Frontend Development to Modern Service-Based Architecture," introduces the reader to the fundamentals required to evolve from browser-based code to distributed systems. Across six sections, students work with JSON, templates, asynchrony, Promises, and client-server communication, ultimately building the foundation of a multi-user SaaS system—a controlled test case—by integrating modularity, data handling, and remote services.

The Second Module, "Modern Backend with JavaScript: Data Persistence, REST APIs, Modular Architecture, and Cloud Deployment" covers the fundamentals of modern backend development through a highly practical approach. It begins with the principles of data persistence, examining in-memory storage, JSON files, and SQL and NoSQL databases; proceeds to explore the Node.js ecosystem, npm, and Express.js for building servers and REST APIs; moves on to integrating PostgreSQL as a data source using CRUD operations; and concludes with professional software organization—employing modular architecture, the MVC pattern, and service and repository layers—as well as cloud application deployment via Git, GitHub, and Vercel, and the optimization and protection of JavaScript code through minification and obfuscation techniques.

Each module follows a methodology in which students do not simply study theoretical concepts but immediately apply them through practical development. Throughout the learning process, they complete hundreds of exercises and progressively build real-world projects by writing every line of code in Visual Studio Code, creating project structures, executing scripts, and monitoring application behavior in real time. This hands-on approach transforms learning into an active experience in which students

observe how their functions, classes, APIs, and modules are rendered in the browser, analyze their execution using developer tools, and validate backend functionality through a local development server. The result is technical training that closely mirrors the workflow of a professional software development environment.

Furthermore, each module is further complemented by practical examples published on the Energyner learning platform (<https://energyner.net/jscript/>), where students can compare their own solutions, validate results, and explore alternative implementations. Together, the books, the online platform, the development environment, browser developer tools, local servers, version control, and cloud deployment form a comprehensive learning ecosystem that guides students through the complete software development lifecycle—from writing source code to building APIs, managing databases, deploying cloud applications, and organizing professional software architecture. Rather than focusing on isolated technologies, this ecosystem helps students develop the analytical thinking and engineering practices required for modern Full Stack software development.

Upon completing the two modules that comprise Book 6, readers will have developed a comprehensive understanding of modern Full Stack software engineering with JavaScript. They will understand how frontend applications communicate with backend services, how APIs integrate with databases, how software is organized using professional architectural patterns, how applications are deployed to cloud platforms, and how code is optimized and protected before production release. Equipped with both practical experience and architectural insight, they will be prepared to design, build, deploy, and maintain real-world applications using the technologies, workflows, and engineering practices adopted throughout today's software industry.

II- INTRODUCTION – FIRST MODULE.

The First Module of Book 6 marks a fundamental transition point in the student's learning journey. Up to this point, much of the learning has focused on building interfaces, client-side programming, and manipulating data within the browser. However, modern applications demand far more advanced capabilities: distributed processing, inter-system communication, consumption of remote services, user management, and decoupled architecture capable of operating in cloud environments and interconnected digital ecosystems.

With this objective, the module introduces a series of technologies and concepts that form the operational basis of current professional applications. The journey begins with the study of JSON and data processing, understanding how information can be structured, exchanged, and transported between different components of an application. JSON thus becomes the universal language that enables communication between frontend, backend, APIs, and distributed services.

Next, the course addresses the development of dynamic templates, a fundamental technique for building modular and reusable interfaces. Through this approach, students begin to understand how modern applications separate content, structure, and logic, facilitating maintenance, scalability, and component reuse within increasingly complex architecture.

The next step involves studying synchronization and asynchronicity in JavaScript, essential concepts for understanding how modern systems connected to external resources function. Unlike traditional programs that execute tasks sequentially, today's web applications continuously interact with servers, APIs, and remote services, requiring mechanisms capable of managing concurrent processes without disrupting the user experience.

Building on this foundation, students are introduced to working with Promises in JavaScript, one of the most important tools for modern development. Through Promises, students learn to coordinate distributed processes, manage deferred responses, handle errors, and construct more robust and efficient execution flows. This knowledge

forms the technical basis for technologies widely used in web applications, APIs, microservices, and cloud-native platforms.

The module then progresses to server communication, integrating all previous knowledge within client-server architecture. Concepts such as HTTP, Request/Response, REST APIs, JSON, `date()`, authentication, and the complete frontend/backend flow are studied. Through practical examples, students understand how modern browsers communicate with remote services, consume APIs, exchange data, and dynamically render the received information.

Finally, all this knowledge converges in the development of the foundation for a professional Multi-User Controlled Test (MMC) SaaS system, integrating validation mechanisms, user control, authentication, data processing, and distributed communication. This project represents much more than an academic exercise: it constitutes a first approach to the architecture currently used in enterprise platforms, multi-user systems, and service-based applications.

Taken together, the six chapters of this module should not be understood as independent topics, but rather as complementary pieces of the same technological evolution. JSON provides the data exchange format; templates allow for the construction of modular interfaces; asynchronicity and Promises manage execution flows; communication with servers connects applications and services; and the SaaS system integrates all these elements within a functional architecture geared towards real users.

Upon completing this first module, the student will have taken a decisive step: they will no longer conceive of applications like simple web pages running in a browser but will begin to understand them as distributed systems composed of multiple interconnected services, components, and resources. This new perspective will serve as the foundation for the following volumes, which will cover modern APIs with Node.js, serverless architecture, microservices, OpenAPI, automated testing, cloud-native deployment, and scalable solutions on platforms such as Render and Google Cloud.

1. JSON. DATA PROCESSING.

1.1 Brief introduction

JSON (JavaScript Object Notation) is a lightweight and universal format for exchanging data between systems. Although it originated within the JavaScript ecosystem, today it is the de facto standard for communication between frontend and backend, web APIs, modern databases, and cloud services.

In the context of Book 6, JSON constitutes the common language of distributed systems: it is the bridge that connects user interfaces, servers, databases, and external services. A precise understanding of it is essential for working with APIs, data persistence, messaging, microservices, and modern architecture based on REST or events.

This chapter does not treat JSON merely as a syntax, but as a fundamental interoperability mechanism that allows information to be transformed into data that can be transported, stored, and processed by different systems and languages.

1.2. Structure, Serialization, and Deserialization

1.2.1. Basic JSON Structure

Table 1. Objects and Arrays – Characteristics.

1.	Objetos (clave–valor)
2.	{
3.	"ride_id": 21,
4.	"client_name": "Ana Pérez",
5.	"total_price": 125.50,
6.	"status": "completed"
7.	}
8.	Arreglos (listas ordenadas)
9.	[
10.	{"step": 1, "event": "send_info"},
11.	{"step": 2, "event": "arrive_location"},
12.	{"step": 3, "event": "waiting"}
13.	]

Essential characteristics of JSON:

- Use double quotes for keys and strings.
- It is strict in its syntax (unlike JavaScript literal objects).
- Supports basic types: string, number, boolean, null, object, array.

This simplicity makes it ideal for APIs, logs, data storage and transmission.

1.2.2. Serialization (Object → JSON)

Serialization is the process of converting a JavaScript object into a JSON string for transmission over a network or for storage.

Table 1 in JavaScript: Object

```
const ride = {  
  ride_id: 21,  
  client: "Ana Pérez",  
  total_price: 125.50,  
  status: "completed"  
};
```

```
const jsonString = JSON.stringify(ride);
```

Result:

Table 2: Serialized Object.

```
"{\\"ride_id\\":21,\\"client\\":\\"Ana Pérez\\",\\"total_price\\":  
125.5,\\"status\\":\\"completed\\"}"
```

Key uses in real-world applications:

- Sending data with `fetch()`
- Registration in databases
- Saved to `localStorage`
- Communication between microservices

Let's study the examples for each application

Table 3 Using fetch to send a request to the backend

```
fetch("/api/registerRide", {  
  method: "POST",  
  headers: { "Content-Type": "application/json" },  
  body: JSON.stringify(ride) });
```

In this example, JSON is the vehicle for transporting information between the frontend and backend.

1.2.3. Deserialization (from JSON to object)

```
const rideObj = JSON.parse(jsonResponse);
```

Deserialization is the reverse process: converting a JSON string into a usable JavaScript object.

Table 1 Deserialization of a JSON string

```
const jsonResponse = '{"ride_id":21,"status":"completed"}';  
const rideObj = JSON.parse(jsonResponse);
```

Now rideObj is a real object:

Table2: Printing the contents of the rideObj object to the console

```
console.log(rideObj.ride_id); // 21  
console.log(rideObj.status); // "completed"
```

This is fundamental when the backend responds to a request:

Table 3: Capturing the Backend Response

```
fetch("/api/getRide?ride_id=21")  
.then(res =>res.json())  
.then(data => {  
  console.log(data.status);  
});
```

In Book 4 of the Programming Step by Step and More series (Mastering JavaScript: Classes, Advanced Inheritance, Polymorphism, Patterns, Modern Principles, Refactoring, and Lifecycle Management), Section 10.4, Serialization and Deserialization of Class Objects, discussed the

serialization and deserialization processes in JavaScript and how they can be performed in two different ways:

- using the language's native methods (`JSON.stringify` and `JSON.parse`)
- using custom methods defined by the programmer (`toJSON` and `fromJSON`).
- Each one fulfills a different role within the object → JSON → object cycle.

See Example 1: Implementing custom methods such as `toJSON` or parsing with `Object.assign`. Professional management.

1.3 Importance of JSON in modern systems

In the context of this Book 6, JSON will be the common thread of: Frontend-backend communication

- REST API Responses
- Persistence in databases
- Structured logs
- Interoperability between services

In short, without JSON there are no modern systems.

1.4 JSON vs Objetos JavaScript

Although JSON and JavaScript objects look similar, they are not the same. Understanding this difference is key to working correctly with APIs, databases, and browser storage.

a) Fundamental conceptual differences

Table 1 Differences between a JavaScript object and JSON

Aspect	JavaScript object	JSON
Nature	Living structure in memory	Text string
Main use	Programming logic	Data transport and storage

Aspect	JavaScript object	JSON
Syntax	Flexible	Strict
Quotation marks	Optional keys	Required in keys
Functions	Yes, they can contain functions	It does NOT allow functions
Methods	Yes (e.g., map, filter)	It has no methods
Comments	Allowed	NOT allowed

Table 2: Comparing the Structure of Both

JSON	OBJECT
If we want to send this object to a server, we must convert it to JSON : <pre>const rideJSON = JSON.stringify(ride); console.log(rideJSON);</pre> Result: <pre>{"ride_id":21,"client":"Ana Pérez","total_price":125.5, "created_at":"2025-02-10T10:15:00.000Z"}</pre> What was lost in the conversion: • The summary() function disappeared. • new Date() was converted to text. This reveals a key truth: JSON carries data, NOT behavior. JavaScript handles data and behavior.	JavaScript object <pre>const ride = { ride_id: 21, client: "Ana Pérez", total_price: 125.50, created_at: new Date(), summary() { return `Ride \${this.ride_id} for \$\$\${this.total_price}`; } };</pre> This is a live object in JavaScript: • It may have methods (summary()), • It can have actual dates (new Date()), It can execute logic.

When We Receive JSON from the Server

Table 3: Reconstructing the JSON

```
const received = JSON.parse(rideJSON);
console.log(received.created_at);
Resultado:
"2025-02-10T10:15:00.000Z"
```

If we want to use it as the actual date, we must reconstruct it:

```
received.created_at = new Date( received.created_at );
```

This is a common pattern in real-world systems.

1.5. JSON in APIs and storage (real-world examples)

1.5.1. JSON as a Frontend → Backend communication language

Table 1. Sending the JSON to the backend with fetch. Typica

```
const ride = {
  ride_id: 21,
  client: "Ana Pérez",
  total_price: 125.50
};

fetch("/api/registerRide", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(ride)
});
```

Here:

- The frontend constructs a JavaScript object:
- The browser, from the frontend, sends **JSON** to the server.
- `Content-Type: "application/json"` tells the server that what is being sent is a JSON object, not a form, not plain text. Without this header, PHP wouldn't know how to interpret the body."
- `body: JSON.stringify( ride )` converts the JavaScript object into a JSON string.
- This string is the one that actually travels over the network.

- The shipment is made by **POST** .
- This allows the backend to receive the JSON in the raw input stream
- The server receives **JSON** text and converts it into an object.

Table 2. In the backend, the JSON is received with PHP

```
$data = json_decode(file_get_contents("php://input"), true);
$data = [
    "ride_id" => 21,
    "client" => "Ana Pérez",
    "total_price" => 125.50
];
```

What the backend does:

- In PHP, the JSON sent by the frontend doesn't arrive as `$_POST` because it's not a form. It arrives as the raw request body. That's why the following is used:
- `$data = json_decode(file_get_contents("php://input"), true);`

Where:

- `php://input` - contains the full body of the HTTP request.
- `file_get_contents()` reads as text
- `json_decode( ...true)` - converts that JSON text into a PHP associative array.
- An example of the result is the array:

```
$data = [
    "ride_id" => 21,
    "client" => "Ana Pérez",
    "total_price" => 125.50
];
```

This pattern is the basis of many stages of the projects we have developed and will carry out later in this course.

1.5.2. How to validate the JSON that is sent from the frontend and received in the backend.

1.5.2.1. Validation of the JSON received in the backend (PHP)

When the frontend sends a JSON object using `fetch()`, the backend receives it as the raw request body.

The first step is to decode it and validate that:

- The JSON is valid
- Required fields exist
- The values have the correct type
- There should be no empty or malformed data
- Table 1. Validating the JSON Object received in the backend

```
//Object sent via fetch
const ride = {
ride_id: 21,
client: "Ana Pérez",
total_price: 125.50
};
//Standard validation procedure
// 1. Read the JSON body sent by the frontend
$json = file_get_contents("php://input");
// 2. Decode the JSON to an associative array
$data = json_decode($json, true);
// 3. Validate that the JSON is valid
if (!$data) {
echo json_encode([
"success" => false,
"message" => "Invalid JSON received"
]);
exit;
}
// 4. Validate required fields
$required = ["ride_id", "client", "total_price"];
foreach ($required as $field) {
if (!isset($data[$field])) {
echo json_encode([
"success" => false,
```

```

    "message" => "Missing required field: $field"
  });
  exit;
}
}
// 5. Validate data types
if (!is_numeric($data["total_price"])) {
  echo json_encode([
    "success" => false,
    "message" => "Invalid price format"
  ]);
  exit;
}
}

```

Technical explanation of the process

- a) Reading of the raw body
 - php://input contains exactly the JSON sent by the frontend.
- b) Decoding
 - json_decode(..., true); transforms the JSON into an associative PHP array.
- c) Structural validation
 - It is verified that the JSON is valid and not empty.
- d) Validation of required fields
 - It is checked that ride_id, client and total_price exist.
- e) Type validation
 - Ensure that total_price is numeric before using it.
 - This process prevents silent errors and ensures that the backend only processes correct data.

Once the ride object has been validated and processed, the backend must also respond to the frontend in JSON format to maintain API consistency.

1.5.2.2. Frontend Responses

The answer can be successful or incorrect.

Table 1. Successful response

```

header("Content-Type: application/json; charset=UTF-8");

echo json_encode([
  "success" => true,

```

```
"message" => "Ride registered successfully",
"ride_id" => $data["ride_id"]
]);
```

Table2. Response in case of error

```
header("Content-Type: application/json; charset=UTF-8");

echo json_encode([
    "success" => false,
    "message" => "Database error: could not register ride"
]);
```

What does this structure achieve?

- The frontend always receives a valid JSON.
- You can easily interpret whether the operation was successful or not.
- A uniform standard is maintained throughout the API.
- It allows for clear and predictable error handling.

1.5.3 Handling HTTP codes in the Backend (200, 400, 500)

In a professional API, it is not enough to send responses in JSON format: it is also essential to return correct HTTP codes, as these allow the frontend to interpret the result of the operation without ambiguity.

An API uses three main codes:

- a) 200 – OK
- b) 400 – Bad Request
- c) 500 – Internal Server Error

The following documents how they should be used in the backend when processing the JSON object:

Table 1. Response “Successful Operation”

```
http_response_code(200);
header("Content-Type: application/json; charset=UTF-8");

echo json_encode([
    "success" => true,
    "message" => "Ride registered successfully",
    "ride_id" => $data["ride_id"]
]);
```

What does this mean for the frontend?

The frontend knows that:

- a) The request was processed correctly
- b) You can continue with the flow (show confirmation, send emails, etc.)

Table 2. HTTP Code 400 – Bad Request (Client Error)

```
if (!$data) {
    http_response_code(400);
    echo json_encode([
        "success" => false,
        "message" => "Invalid JSON received"
    ]);
    exit;
}

$required = ["ride_id", "client", "total_price"];

foreach ($required as $field) {
    if (!isset($data[$field])) {
        http_response_code(400);
        echo json_encode([
            "success" => false,
            "message" => "Missing required field: $field"
        ]);
        exit;
    }
}

if (!is_numeric($data["total_price"])) {
    http_response_code(400);
    echo json_encode([
        "success" => false,
        "message" => "Invalid price format"
    ]);
    exit;
}
```

What does this mean for the frontend?

The frontend knows that:

- a) The request was sent with errors
- b) You must correct the data before resubmitting
- c) You should not attempt to continue the flow

Table 3. HTTP Code 500 – Internal Server Error

```
http_response_code(200);  
header("Content-Type: application/json; charset=UTF-8");  
  
echo json_encode([  
    "success" => true,  
    "message" => "Ride registered successfully",  
    "ride_id" => $data["ride_id"]  
]);
```

What does this mean for the frontend?

The frontend understands that:

- a) The error is not the user's fault
- b) It should display a generic message (“Internal error, please try again later”)
- c) You must not modify the submitted data

The backend of a professional API uses HTTP codes to communicate the actual status of each operation. Code 200 indicates success; 400 indicates errors in the data sent by the client; and 500 reflects internal server failures.

Each response includes a JSON object that details the result, allowing the frontend to properly manage the application flow.

In summary:

The frontend sends the ride object as JSON using fetch.

The backend receives it from `php://input`, decodes it with `json_decode()` and validates its structure, fields and types.

After processing it, the backend also responds in JSON format, indicating success or error.

This mechanism guarantees modern, clean and decoupled communication between client and server, following the standard of professional REST APIs.

1.6. Solutions for storing a JSON Object

1.6.1. Local (localStorage)

We can store the JSON **ride** object directly in the browser using `localStorage`. If we choose to save this information under the **searchFormData** key, the correct instruction to persist the content is:

```
localStorage.setItem(" searchFormData ", JSON.stringify( ride ));
```

We can then retrieve those stored values using:

```
const stored = JSON.parse(localStorage.getItem(" searchFormData "));
```

Once the object is obtained, its properties can be used as variables within the application flow. For example, to access the value of `total_price` and print it to the browser console:

```
console.log(stored.total_price);
```

In this way, the data previously saved in `localStorage` remains available for any stage of the process, allowing consistency and continuity in the user experience.

Table 1. Saving JSON and retrieving properties in `localStorage`

```
const ride = {  
  ride_id: 21,  
  client: "Ana Pérez",  
  total_price: 125.50  
};  
//Storing the JSON object in localStorage:  
localStorage.setItem(" searchFormData ", JSON.stringify( ride ));  
//Recovering the properties:  
const stored = JSON.parse(localStorage.getItem(" searchFormData "));  
//Using one of its properties within the application flow, we  
can access its value and print it.  
console.log( stored .total_price);
```

Real advantages of this procedure:

- Data persists between pages.
- It allows the user flow to continue.
- Avoid reloading entire forms.

1.6.2. JSON in modern databases

In many current applications:

- MySQL can store native JSON.
- MongoDB uses JSON as its base format.

Table 1. Conceptual Instruction

```
//En MySQL:
INSERT INTO rides (ride_data)
VALUES ('{"ride_id":21,"client":"Ana Pérez"}');

// en MongoDB:

db.rides.insertOne({
  ride_id: 21,
  client: "Ana Pérez",
  total_price: 125.5
});
```

This shows how JSON becomes a **universal data language** .

1.7 Conclusions of this Chapter

This chapter solidified the understanding of JSON as the standard data exchange format in modern systems, highlighting its simple structure based on objects and arrays, as well as the fundamental processes of serialization (converting objects into JSON strings) and deserialization (converting JSON into usable objects). Through practical examples in JavaScript and PHP, the student observed how JSON objects are constructed, sent, and received within a real Frontend → Backend flow, including validations, error handling, and the interpretation of server responses. The chapter also delved into the differences between JSON and JavaScript objects, clarifying their relationship, their limitations, and the importance of adhering to the format's strict syntax.

The chapter also highlighted the essential role of JSON in APIs, local storage, and modern databases, reinforcing its value as a universal communication language between decoupled systems. It covered the proper handling of HTTP status codes (200, 400, 500) for transmitting semantic state to the frontend, as well as techniques for validating JSON

on the backend before processing. Finally, it explored storage solutions, from `localStorage` to databases that accept JSON columns, demonstrating how this format enables the development of scalable, secure applications that adhere to current web development standards.

7. GENERAL CONCLUSIONS OF BOOK 6 – FIRST MODULE

From Frontend Development to Modern Service-Based Architecture

The development of this first module allowed us to progress from fundamental data processing concepts to the construction of web systems capable of communicating with servers, consuming remote services, and managing users within modern, decoupled architecture. The journey began with the study of JSON as a universal information exchange mechanism, understanding its role in serialization, deserialization, storage, and communication between the frontend and backend. This foundation proved essential for understanding how modern applications exchange data in a structured and efficient manner.

Subsequently, working with templates and dynamic rendering allowed students to understand how modern interfaces generate content in a flexible, reusable way that adapts to information received from APIs and external services. This approach introduced students to concepts currently used by SaaS frameworks and platforms, where the separation of structure, logic, and presentation is a fundamental principle for building scalable applications.

The study of synchronization and asynchronicity in JavaScript, along with the use of `fetch()`, `try/catch` blocks, and APIs, allowed us to understand how applications connected to external resources operate. Through these mechanisms, it was demonstrated that modern systems depend on processes capable of running concurrently, enabling them to maintain responsive interfaces while exchanging information with servers and distributed services.

The incorporation of Promises, operation chaining, and `async/await` consolidated a deeper understanding of the asynchronous flows used in professional applications. The projects developed demonstrated how to coordinate multiple processes, robustly manage errors, and build modular architecture capable of progressively evolving into more complex distributed systems.

One of the module's most important contributions was the formalization of frontend-backend communication. The analysis of HTTP, Request/Response, REST APIs, JSON, authentication, security, resource protection, and the complete client-server flow revealed that modern applications are not isolated programs, but rather ecosystems

composed of multiple services that collaborate using standardized protocols. This knowledge forms the technical basis for modern cloud-native architecture, microservices, PWAs, and SaaS systems.

The practical culmination of the module materialized in the development of multi-user access systems, authentication controls, log management, event auditing, and the construction of reusable SaaS structures. These implementations allowed the student to experiment with real-world software engineering elements, including dynamic forms, databases, user validation, session management, and modular design oriented toward enterprise applications.

From an architectural perspective, the main takeaway from Book 6 - Module One is understanding that modern web development is no longer limited to creating visual pages or running scripts in a browser. Students have begun working with applications that exchange data, consume services, manage users, coordinate asynchronous processes, and operate using decoupled components capable of evolving into distributed, professional-grade platforms.

Furthermore, the module allowed for the gradual introduction of concepts that will be essential in subsequent volumes: modern APIs, advanced authentication, API-First architecture, SaaS systems, modularity, security, distributed services, and cloud communication. In this way, the exercises developed cease to be isolated practices and become the first building blocks of more complex and scalable platforms.

Finally, it can be said that this Book 6 - Module One constitutes the bridge between traditional frontend development and modern service-based software engineering. Upon completion of the module, the student possesses the necessary foundation to tackle the challenges of the next level: the design and construction of professional APIs with Node.js, serverless architecture, OpenAPI, automated testing, cloud-native deployments, and scalable systems on distributed infrastructures.

8. REFERENCES.

1. Eloquent JavaScript

Author: Marijn Haverbeke

One of the best texts for understanding:

- Modern JavaScript
- asynchrony
- Promises
- `fetch()`
- APIs
- data structures

Level: Intermediate.

2. JavaScript: The Definitive Guide

Author: David Flanagan

Reference work for:

- Advanced JavaScript
- JSON
- asynchrony
- Promises
- modern web programming

Level: Intermediate – Advanced.

3. You Don't Know JS Yet

Author: Kyle Simpson

Especially recommended for:

- Event Loop
- asynchrony
- Promises
- `async/await`
- JavaScript architecture

Level: Intermediate.

4. Learning JavaScript Design Patterns

Author: Addy Osmani

- Excellent introduction to:
- decoupled architecture
- modularity
- scalability
- patterns used in SaaS

Level: Intermediate.

5. RESTful Web APIs

Authors: Leonard Richardson, Mike Amundsen

Ideal for exploring in depth:

- REST
- HTTP
- endpoints
- resources
- API-first design

Level: Advanced.

6. Designing Web APIs

Authors: Brenda Jin, Saurabh Sahni

- Focused on:
- professional API design
- REST contracts
- interoperability

7. Building Microservices

Author: Sam Newman

Enter:

- distributed services
- decoupling
- SaaS
- scalability

8. Clean Architecture

Author: Robert C. Martin

Fundamental reference on:

- modern architecture
- separation of responsibilities
- decoupled backend

Level: Advanced.

9. Web Development with Node and Express

Author: Ethan Brown

Excellent bridge to:

- Node.js
- REST APIs
- middleware
- backend services

10. Cloud Native Patterns

Author: Cornelia Davis

Enter:

- cloud-native
- microservices
- Distributed APIs
- scalable systems

Level: Advanced.

11.MDN Web Docs JavaScript Guide

The most important documentation for:

- Modern JavaScript
- `fetch()`
- JSON
- Promises
- `async/await`

12. MDN HTTP Overview

Reference on:

- HTTP
- Request/Response
- Headers
- Status Codes

13. JSON.org

Official website for:

- JSON syntax
- structures
- data exchange

14. OpenAPI Initiative

- Preparation for:
- API documentation
- REST contracts

15. Node.js Documentation

Official base for:

- Node.js
- Modern APIs
- JavaScript backend

16. Express.js Documentation

Essential for:

- REST API development
- middleware
- routes

17. Google Cloud Architecture Center

Excellent reference for:

- cloud-native
- microservices
- distributed systems

18. Render Documentation

Preparation for:

- cloud deployment
- API hosting

19. OWASP Foundation

Essential reference on:

- web security
- authentication
- API protection
- goodpractices

20. Google Web Dev PWA Guides

Ideal for exploring in depth:

- PWAs
- Service Workers
- offline applications
- modern architecture frontend

9. MOTIVATIONAL PHRASES.

"Success is the sum of small efforts repeated day after day."

Robert Collier

"The only limit to your fulfillment tomorrow will be your doubts today."

Franklin D. Roosevelt

"It doesn't matter how slow you go as long as you don't stop."

Confucius

"The path to success and greatness is to develop what you already have."

ZigZiglar

"The secret of success lies in knowing more than others."

Aristotle Onassis

"It's not about how many times you fall, but how many times you get up."

Vince Lombardi

"Success is not the key to happiness. Happiness is the key to success. If you love what you do, you will be successful."

Albert Schweitzer

"The only way to do great work is to love what you do."

Steve Jobs

"The biggest risk is not taking any risks. In a world that's changing so fast, the only strategy that's guaranteed to fail is not taking risks."

Mark Zuckerberg

"Motivation drives us to begin, and habit keeps us going."

Jim Ryun

"It's not enough to tell the truth; you have to prove that you have it."

Baltasar Gracián

If you can't fly, run; if you can't run, walk; if you can't walk, crawl, but keep moving toward your goal."

Martin Luther King Jr

"No matter your physical age, you can always motivate your mind to face tomorrow."

From the Author

OTHER BOOKS BY THE AUTHOR

If you would like to continue exploring the *Programming Step by Step and More* series, you can access all related volumes, updates, and additional books available on Amazon through the QR code below.

All books in this series are available on Amazon.

Scan the QR code to discover more:



10. ABOUT THE AUTHOR.

After several decades dedicated to improving the energy efficiency of heating and cooling equipment and systems, I faced a major change: retirement. Upon retiring, I felt disoriented and without purpose, which affected my emotional well-being. To combat this feeling, I sought a new occupation that would keep me active and motivated.

I decided to share my experiences and accumulated knowledge from over four decades of working in the industry. This led to the creation of this project, with the goal of publishing articles, calculation tools, and tutorials on the various disciplines I've worked in throughout my engineering career. I've acquired these computer skills and knowledge through self-study, delving into technical, theoretical, and practical topics. This includes web programming, which I used as an auxiliary tool to optimize project management, investments, finance, and marketing.

Based on lived experience, web programming is a powerful tool key to solving complex problems in industrial environments, technical services, and everyday life.

My new perspective on the future, focused on the information technology revolution, has been key to keeping my mind active, staying up-to-date, and learning new technologies empirically. In this context of constant and accelerated change, my vision is expanding toward a horizon where AI not only optimizes processes but also redefines industries and creates previously unimaginable opportunities.

This guide highlights the importance of programming languages as essential tools in the modern world and is designed to support people of all ages in developing their professional skills and achieving their career goals. It provides valuable resources for both those starting their careers and those seeking new ways to apply their expertise.

The Author

PROGRAMMING STEP BY STEP AND MORE ®

An Instructive Guide

Book 6- Full-Stack Engineering with JavaScript: From Modern Frontend to Modular Architecture and Cloud Deployment.
Part One: From Frontend Development to Modern Service-Based Architecture.

Published in Paperback, Hardcover, and Digital

Miami, Florida, USA • 2026

English-language edition © 2026 René F. Ruano Domínguez

Translated from Spanish by René F. Ruano Domínguez

Originally published in Spanish as:

PROGRAMANDO PASO A PASO Y MÁS

INSTRUCTIVO – LIBRO 6.

Ingeniería Full-Stack con JavaScript: Del Frontend Moderno a la Arquitectura Modular y el Despliegue en la Nube

Primera Parte: Del Desarrollo Frontend a la Arquitectura Moderna basada en Servicios.

Copyright © 2026 René F. Ruano Domínguez

All rights reserved.

Publisher: Corporate Luxury Group, LLC