

Introduction to Back-End Development

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-05

Table des matières

1 Session Structure	1
2 What is Back-End Development?	1
2.1 PHP	1
2.2 JavaScript	2
2.3 Other Languages and Frameworks ..	2
3 Why Focus on JavaScript Frameworks? ..	2
3.1 Front-End and Full-Stack Development	2
3.2 Node.js	2
3.3 Why Focus on React and Remix?	2
4 Web Servers (Apache, Nginx)	3
4.1 Web Servers (Apache, Nginx)	3
5 Data Storage	3
6 Machine-to-Machine vs. Human-to- Machine Communication	3
6.1 Comparison	4
7 Key Takeaways	4

1 Session Structure

- Introduction to Back-End Development Concepts
- Types of Back-End Solutions
- Web Servers and Data Storage
- Rest and WebSockets for Communication

2 What is Back-End Development?

- The server-side of web development, handling data processing, logic, and database communication.
- Responsibilities of Back-End Developers
 - Building APIs (Application Programming Interfaces) to communicate with the front-end.
 - Managing databases and data security.
 - Implementing server-side logic and business rules.
- Various programming languages and frameworks can be used for back-end development.

2.1 PHP

- **PHP:** A widely-used open-source scripting language especially suited for web development.
- **Frameworks:**
 - **Laravel:** A PHP framework known for its elegant syntax and robust features.
 - **Symfony:** A PHP framework for web applications and a set of reusable PHP components.

2.2 JavaScript

- **Node.js:** An open-source, JavaScript run-time environment built on Chrome's V8 engine.
- **Frameworks:**
 - **Express.js:** A minimal and flexible Node.js web application framework.
 - **NestJS:** A progressive Node.js framework for building efficient, reliable, and scalable server-side applications.

2.3 Other Languages and Frameworks

- **Python:** Django, Flask
- **Ruby:** Ruby on Rails
- **Java:** Spring Boot, JakartaEE
- **C#:** ASP.NET

3 Why Focus on JavaScript Frameworks?

- **Unified Language:** JavaScript can be used for both front-end and back-end development, reducing complexity.
- **Large Ecosystem:** Extensive libraries and tools available for JavaScript.
- **Performance:** Node.js is known for its non-blocking, event-driven architecture, making it efficient for I/O operations.

3.1 Front-End and Full-Stack Development

- **React for Front-End:**
 - React is a popular JavaScript library for building user interfaces.
 - It allows developers to create reusable UI components, making front-end development more efficient and maintainable.
 - React's component-based architecture promotes reusability and scalability.
- **Remix for Full-Stack:**
 - Remix is a full-stack web framework built on top of React.

- It simplifies full-stack development by providing tools for both front-end and back-end development.
- Remix focuses on performance and user experience, offering features like server-side rendering (SSR) and static site generation (SSG).
- By using Remix, developers can build modern web applications with a unified language and framework.

3.2 Node.js

- **Event-Driven Architecture:**
 - Non-blocking I/O operations.
 - Efficient handling of multiple concurrent connections.
 - Ideal for I/O-heavy applications.
- **Single-Threaded:**
 - Uses a single-threaded event loop for asynchronous operations.
 - Simplifies concurrency management.
- **Package Ecosystem:**
 - npm (Node Package Manager) provides access to thousands of libraries and modules.
 - Facilitates rapid development and code reuse.

3.3 Why Focus on React and Remix?

- **React:** A JavaScript library for building user interfaces, known for its component-based architecture and virtual DOM.
- **Remix:** A full-stack web framework built on React, focusing on performance and user experience.
- **Integration:** Seamless integration between front-end (React) and back-end (Node.js) using Remix.
- **Modern Development:** Emphasizes modern web development practices, including server-side rendering and static site generation.

4 Web Servers (Apache, Nginx)

- **Key Functionalities:**

- ▶ **Serving Static Content:**

- Delivering files such as HTML, CSS, JavaScript, images, and videos directly to the client.
 - Both Apache and Nginx are optimized for fast and efficient static content delivery.

- ▶ **Routing Requests:**

- Directing incoming HTTP requests to the appropriate application or service.
 - Supports URL rewriting, redirection, and proxying to backend servers.
 - Essential for handling dynamic content generated by web applications.

- ▶ **Plugins and Modules:**

- **Apache:**
 - Highly configurable with `.htaccess` files.
 - Supports a wide range of modules for additional functionality, such as `mod_php` for running PHP scripts.
 - **Nginx:**
 - Uses a modular architecture with dynamic modules.
 - Can be configured to work with PHP via `php-fpm` (FastCGI Process Manager) for efficient PHP processing.

4.1 Web Servers (Apache, Nginx)

- **Overview of Common Web Servers:**

- ▶ **Apache:**

- One of the most widely used web servers.
 - Known for its flexibility and extensive module support.
 - Suitable for serving both static and dynamic content.
 - Highly configurable with `.htaccess` files.

- ▶ **Nginx:**

- Popular for its high performance and low resource consumption.
 - Excels at handling concurrent connections.
 - Often used as a reverse proxy and load balancer.
 - Efficient for serving static content.

5 Data Storage

- **Relational Databases (MySQL, PostgreSQL):** Structured data with tables and relationships.

- ▶ Benefits: strong consistency, complex queries.
 - ▶ Drawback: schema rigidity.

- **Non-Relational Databases (MongoDB):** Flexible schema, stores data in JSON-like documents.

- ▶ Benefits: scalability, rapid development.
 - ▶ Drawback: complex querying for relational data.

6 Machine-to-Machine vs. Human-to-Machine Communication

- **Human-to-Machine (HTML):**

- ▶ Traditional web interactions where users request and receive data.
 - ▶ Typically involves web browsers and servers communicating using HTTP/HTTPS.
 - ▶ Example: A user fills out a form on a website and submits it to the server.

- **Machine-to-Machine Communication:**

- ▶ **REST (Representational State Transfer):**

- A stateless architectural style for designing networked applications.
 - Uses standard HTTP methods (GET, POST, PUT, DELETE) for communication.

- Ideal for CRUD (Create, Read, Update, Delete) operations.
- Example: A client application sends a GET request to a RESTful API to retrieve data.
- **WebSockets:**
 - A protocol for real-time communication between systems.
 - Establishes a persistent connection between the client and server.
 - Allows for bidirectional communication, enabling real-time updates.
 - Example: A chat application where messages are instantly sent and received without refreshing the page.

6.1 Comparison

- **REST:**
 - **Pros:**
 - Simple and easy to implement.
 - Uses standard HTTP methods and status codes.
 - Well-suited for CRUD operations and stateless interactions.
 - **Cons:**
 - Not ideal for real-time communication.
 - Requires multiple HTTP requests for continuous updates.
- **WebSockets:**
 - **Pros:**
 - Enables real-time, bidirectional communication.
 - Reduces latency by maintaining a persistent connection.
 - Ideal for applications requiring instant updates (e.g., chat apps, live notifications).
 - **Cons:**
 - More complex to implement compared to REST.
 - Requires additional considerations for scalability and security.

7 Key Takeaways

- **Understanding Back-End Development:**
 - Grasp the fundamental concepts and responsibilities of back-end development.
 - Learn about different types of back-end solutions and their use cases.
- **JavaScript and Node.js:**
 - Explore the advantages of using JavaScript for both front-end and back-end development.
 - Understand the event-driven architecture and single-threaded nature of Node.js.
- **PHP:**
 - Understand the role of PHP in web development.
 - Learn about popular PHP frameworks like Laravel and Symfony for building robust web applications.
- **Frameworks and Tools:**
 - Get familiar with popular frameworks like Express.js and NestJS for building scalable and efficient server-side applications.
 - Learn about the integration of React for front-end and Remix for full-stack development.
- **Web Servers and Data Storage:**
 - Understand the roles of web servers like Apache and Nginx in serving static and dynamic content.
 - Learn about different data storage solutions, including relational and non-relational databases.
- **Communication Methods:**
 - Compare REST and WebSockets for machine-to-machine communication.
 - Understand the pros and cons of each method and their suitable use cases.