

DEV-500 – Area

# **Stack Benchmark**



# Table of content

• Team .....	1
• Context .....	2
• Backend .....	3
• Comparison table .....	3
• Decision .....	4
• Security risks and Mitigations .....	5
• Database .....	6
• Comparison table .....	6
• Decision .....	7
• Security risks and Mitigations .....	8
• Frontend (web) .....	9
• Comparison table .....	9
• Decision .....	10
• Security risks and Mitigations .....	11
• Frontend (mobile) .....	12
• Comparison table .....	12
• Decision .....	13
• Security risks and Mitigations .....	14
• Conclusion .....	15



# Team

- **Backend**

- Pierre MARGUERIE (*pierre.marguerie@epitech.eu*)
- Arthur DRAHY (*arthur.drahy@epitech.eu*)
- Timéo TREGAROT (*timeo.tregarot@epitech.eu*)

- **Frontend (web)**

- Esteban BOUYAULT--YVANEZ  
(*esteban.bouyault-yvanez@epitech.eu*)

- **Frontend (mobile)**

- Lysandre BOURSETTE (*lysandre.boursette@epitech.eu*)



# Context

The goal is to develop a web application similar to IFTTT or Zapier, where users can create, configure, and run automated workflows. The requirements will be the following:

- A **scalable backend** capable of handling many requests at once.
- A **database** able to manage both structured and semi-structured data with strong scalability.
- A **frontend** that can provide an interactive workflow builder (drag-and-drop, event triggers, visual flows).
- A stack that balances **performance, scalability, developer productivity, and security**.



# 1. Backend

	<u>NestJS</u> (Node.js/TS)	<u>FastAPI</u> (Python)	<u>Go</u> (Golang)
Performance	<b>Good</b> (compiled language)	<b>Average</b> (Python interpreter)	<b>Excellent</b> (compiled, close to C performance)
Ease of Development	<b>Medium to high</b> (MVC structure, decorators)	<b>Very high</b> (clear syntax, little boilerplate)	<b>Medium</b> (simple syntax but lower level)
Ecosystem	<b>Large</b> ( <u>npm</u> , frontend/backend, modern libs)	<b>Rich</b> (web, ORM)	<b>Decent</b> (solid stdlib, but smaller ecosystem)
Learning Curve	<b>Medium</b> (TypeScript + Nest concepts)	<b>Low</b>	<b>Medium</b> (simple syntax but different paradigm)
Scalability	<b>Good</b> (microservices-oriented, WS)	<b>Limited</b> , good for I/O APIs	<b>Very good</b> (native concurrency support)
Community	<b>Large</b> (backed by the JS ecosystem)	<b>Large</b>	<b>Large</b> (DevOps/ backend focus)
Typical Use Case	Full web apps, Node-based microservices	Quick APIs, AI integrations	High-performance services, distributed systems
Experience	Backend team has experience with JS	<b>Everyone</b> (Python)	– <b>Esteban B.</b> (since 2024)



# 1. Backend

## Decision

**NestJS** was chosen for the backend because:

- Provides **structured and opinionated (strict) architecture**, crucial for complex workflow applications.
- Excellent **ecosystem** with npm and TypeScript, enabling fast integration of external services.
- Built-in support for **microservices and websockets**, aligned with event-driven workflow execution.
- Good balance between **performance and developer productivity** compared to FastAPI (simpler but less scalable) and Go (faster but more low-level).
- Due to the far due date, we have decided that this project would be a good opportunity to learn a new framework, that is widely used in the real-world.



# 1. Backend

## Security risks and Mitigations

- **Validation misconfiguration:** Always enable global validation (ValidationPipe with whitelist, forbidNonWhitelisted).
- **CORS misconfigurations:** Restrict allowed origins in production.
- **Weak JWT/auth handling:** Use short-lived tokens, refresh strategies, secure secret storage.
- **NoSQL/SQL injection:** Always use parameterized queries and sanitize inputs.
- **Supply chain attacks:** Audit npm dependencies regularly and pin versions.
- **Rate limiting:** Use throttling to prevent DoS.



## 2. Database

	<u>MongoDB</u>	<u>PostgreSQL</u>	<u>MariaDB</u>
Type	<b>NoSQL</b> (schema-less document-oriented)	<b>Relational</b>	<b>Relational</b>
Performance	<b>Excellent</b> for massive R/W operations	<b>Very good</b> (especially for complex queries)	<b>Good</b> (optimized for simple queries)
Scalability	<b>Excellent</b> (sharding, native replication)	<b>Very good</b> (vertical and horizontal)	<b>Good</b> , but less advanced than MongoDB
Data Flexibility	<b>Very flexible</b>	<b>Strict schema</b> , but supports JSON/JSONB	<b>Low flexibility</b> (SQL schema)
Transactions	<b>Limited multi-document transactions</b>	<b>Full ACID</b>	<b>Full ACID</b>
Learning Curve	<b>Medium</b> (document paradigm is not the same as SQL)	<b>Medium</b> (basically MySQL with more features)	<b>Low</b>
Typical Use Case	Big Data, logs, flexible apps	Critical apps, complex analytics	Classic web apps
Experience	No experience.	- <u>Arthur D.</u> (since 2025)	- <u>Lysandre B.</u> (since 2019) - <u>Pierre M.</u> (since 2024) - <u>Timéo T.</u> (since 2024)





## 2. Database

### Decision

**MongoDB** was chosen for the database because:

- Provides **schema flexibility**, essential to store workflow definitions, user configurations, and unstructured payloads.
- Built-in **sharding and replication**, ensuring horizontal scalability for millions of events.
- **Strong ecosystem** for **real-time data handling**.
- **Easier** to iterate quickly on new features compared to rigid relational schemas.
- We yearn to learn it.



## 2. Database

### Security risks and Mitigations

- **Open instances without authentication:** Always enforce authentication and TLS.
- **NoSQL injection:** Sanitize inputs, restrict operators.
- **Overly permissive roles:** Use least-privilege roles per service.
- **Unencrypted data at rest:** Enable disk encryption and TLS for all connections.
- **Backup exposure:** Secure backup storage, rotate credentials.



## 3. Frontend (Web)

	<u>React/TypeScript/Vite</u>	<u>Angular</u>	<u>Svelte</u>
<b>Architecture</b>	UI library, highly flexible	Full framework (opinionated, MVC-like)	Compiled framework, component-driven
<b>Performance</b>	<b>Good</b>	<b>Good</b> , but heavy bundles	<b>Excellent</b> (compiled to optimized JS)
<b>Learning curve</b>	<b>Medium</b> (some ecosystem learning)	<b>High</b> (complex concepts, <u>RxJS</u> , modules)	<b>Low-medium</b> (syntax close to native HTML)
<b>Ecosystem</b>	<b>Very large</b> ( <u>npm</u> , many third-party tools)	<b>Large</b> , structured	<b>Smaller</b> , but rapidly growing
<b>Community</b>	<b>Very large and dominant</b> (Meta)	<b>Large and active</b> (Google)	<b>Smaller</b> but very active
<b>Scalability</b>	<b>Good</b> , requires strong architecture discipline	<b>Very good</b> for large enterprise projects	Less suited for very large projects
<b>Typical Use Case</b>	Versatile web apps, flexible frontends	Enterprise apps, complex projects	Lightweight apps, fast prototypes
<b>Experience</b>	– <u>Esteban B.</u> (since 2024)	No experience.	No experience.



## 3. Frontend (Web)

### Decision

**React/TypeScript** was chosen for the web frontend because:

- It is the **industry standard**, ensuring long-term support.
- **Huge ecosystem** (state management, routing, visualization).
- **Great flexibility** to build a complex workflow builder UI with drag-and-drop, real-time updates, and reusable components.
- **Large and active community**, ensuring better sustainability than Svelte.
- **Esteban is already experienced with React**, reducing the risk of beginner mistakes and speeding the development process.



## 3. Frontend (Web)

### Security risks and Mitigations

- **XSS via dangerous rendering:** Avoid direct HTML injection, sanitize content.
- **Insecure state management (token leaks):** Store tokens in HttpOnly cookies, avoid localStorage for sensitive data.
- **Dependency vulnerabilities:** Audit npm packages regularly.
- **Clickjacking:** Use CSP and X-Frame-Options headers.
- **Leakage in source maps:** Disable source maps in production builds.



## 4. Frontend (Mobile)

	<u>Flutter</u>	<u>React Native / Expo</u>	<u>Native (Kotlin for Android)</u>
<b>Cross-platform</b>	<b>Yes</b> (Android, iOS, Web, Desktop)	<b>Yes</b> (Android and iOS)	<b>No</b>
<b>Performance</b>	<b>Very good</b> (near-native, own render engine)	<b>Good</b> , depends on native bridges	<b>Excellent</b>
<b>UI/UX consistency</b>	<b>Very strong</b>	<b>Medium</b> (relies on native components)	<b>Perfect</b> (native look & feel)
<b>Ecosystem</b>	<b>Growing</b> , <u>rich widget library and plugins</u>	<b>Large</b> (npm ecosystem, many libraries)	<b>Mature</b>
<b>Development speed</b>	<b>Very high</b> (hot reload, community packages)	<b>High</b> (hot reload, JS familiarity)	<b>Medium</b> (requires platform-specific code)
<b>App size</b>	<b>Larger binaries</b> (~40–50MB for small apps)	<b>Smaller than Flutter, larger than native</b>	<b>Smallest possible</b>
<b>Typical Use Case</b>	Cross-platform apps with complex UI/UX	Cross-platform mobile apps, quick MVPs	High-performance apps
<b>Experience</b>	– <u>Lysandre B.</u> (since 2022)	No experience.	– <u>Lysandre B.</u> (2019); Very little



## 4. Frontend (Mobile)

### Decision

**Flutter** was chosen for the mobile frontend because:

- Our mobile developer is already experienced with Flutter, reducing the risk of beginner mistakes.
- Flutter provides a **cross-platform solution** with **consistent UI/UX** across Android and iOS.
- It offers **fast development cycles** (hot reload, widget ecosystem) which aligns with the project's need for **rapid iteration**.
- While app size is larger compared to native, this trade-off is acceptable given the productivity and consistency benefits.



## 4. Frontend (Mobile)

### Security risks and Mitigations

- **Risk of large bundle size exposing unused packages:**  
Mitigate by tree-shaking and package audits.
- **Dependency ecosystem still maturing:** Select only well-maintained packages with good reputation.
- **Insecure state management (token leaks):** Store tokens in secure storage, avoid direct clear disk-write for sensitive data.



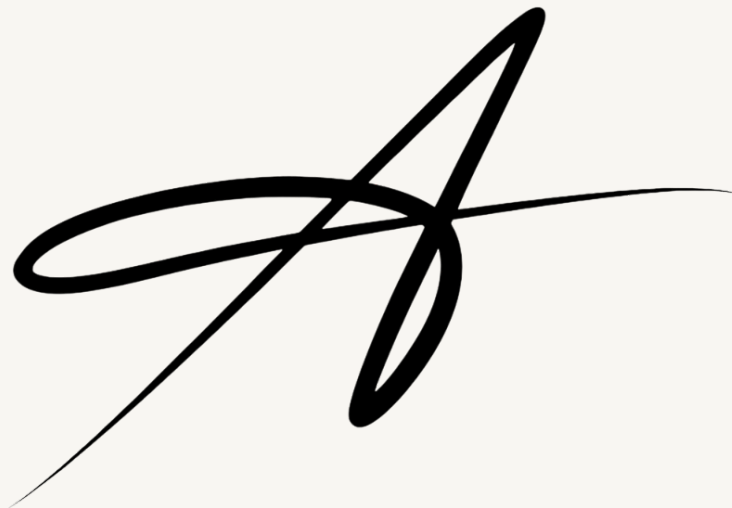


# Conclusion

**After evaluation, the following choices were made:**

- **Backend:** NestJS (TS)
- **Database:** MongoDB
- **Web Frontend:** React/Vite (TypeScript)
- **Mobile Frontend:** Flutter (Dart)

This stack balances developer productivity, scalability, and security, making it a solid foundation for this project.



{EPITECH}