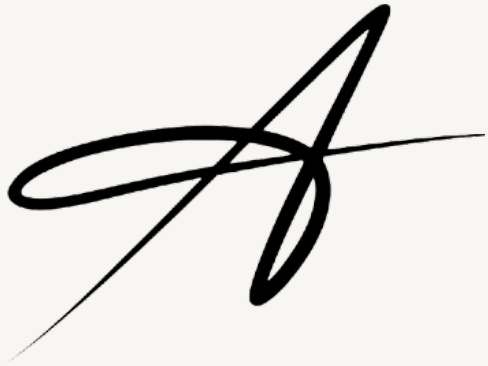

DEV-500 – Area

Stack Benchmark

{EPITECH}

Table of content

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

	NestJS (Node.js/TS)	**FastAPI** (Python)	**Go** (Golang)
Performance	**Good** (compiled language)	**Average** (Python interpreter)	**Excellent** (compiled, close to C performance)
Ease of Development	**Medium to high** (MVC structure, decorators)	**Very high** (clear syntax, little boilerplate)	**Medium** (simple syntax but lower level)
Ecosystem	**Large** (npm, frontend/backend, modern libs)	**Rich** (web, ORM)	**Decent** (solid stdlib, but smaller ecosystem)
Learning Curve	**Medium** (TypeScript + Nest concepts)	**Low**	**Medium** (simple syntax but different paradigm)
Scalability	**Good** (microservices-oriented, WS)	**Limited**, good for I/O APIs	**Very good** (native concurrency support)
Community	**Large** (backed by the JS ecosystem)	**Large**	**Large** (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	**Everyone** (Python)	- Esteban 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

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

	React/TypeScript/Vite	Angular	Svelte
Architecture	UI library, highly flexible	Full framework (opinionated, MVC-like)	Compiled framework, component-driven
Performance	**Good**	**Good**, but heavy bundles	**Excellent** (compiled to optimized JS)
Learning curve	**Medium** (some ecosystem learning)	**High** (complex concepts, RxJS, modules)	**Low-medium** (syntax close to native HTML)
Ecosystem	**Very large** (npm, many third-party tools)	**Large**, structured	**Smaller**, but rapidly growing
Community	**Very large and dominant** (Meta)	**Large and active** (Google)	**Smaller** but very active
Scalability	**Good**, requires strong architecture discipline	**Very good** for large enterprise projects	Less suited for very large projects
Typical Use Case	Versatile web apps, flexible frontends	Enterprise apps, complex projects	Lightweight apps, fast prototypes
Experience	- **Esteban B.** (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)

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

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