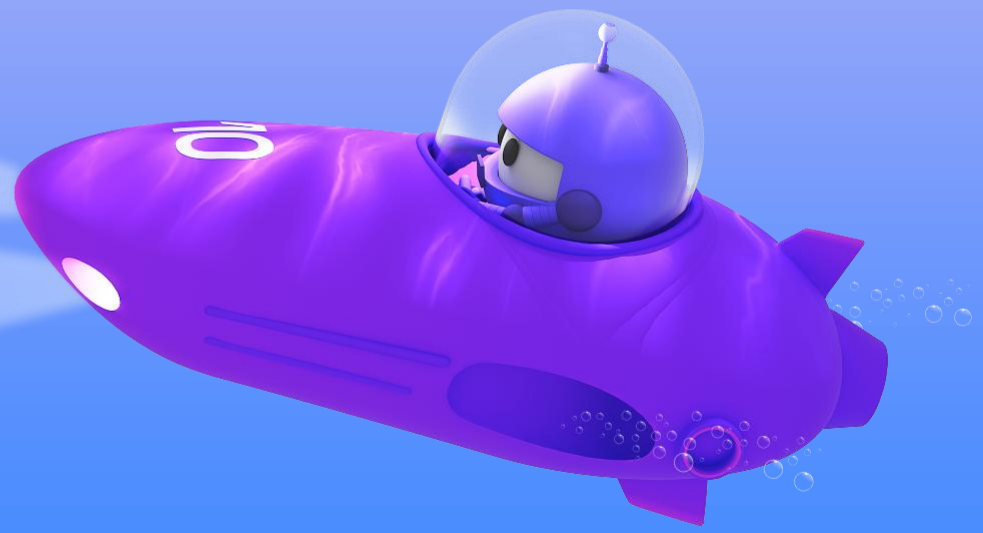
Decision Records

Understanding Why Those Decisions Were Made!

Sarah “sadukie” Dutkiewicz
Senior Trainer
NimblePros

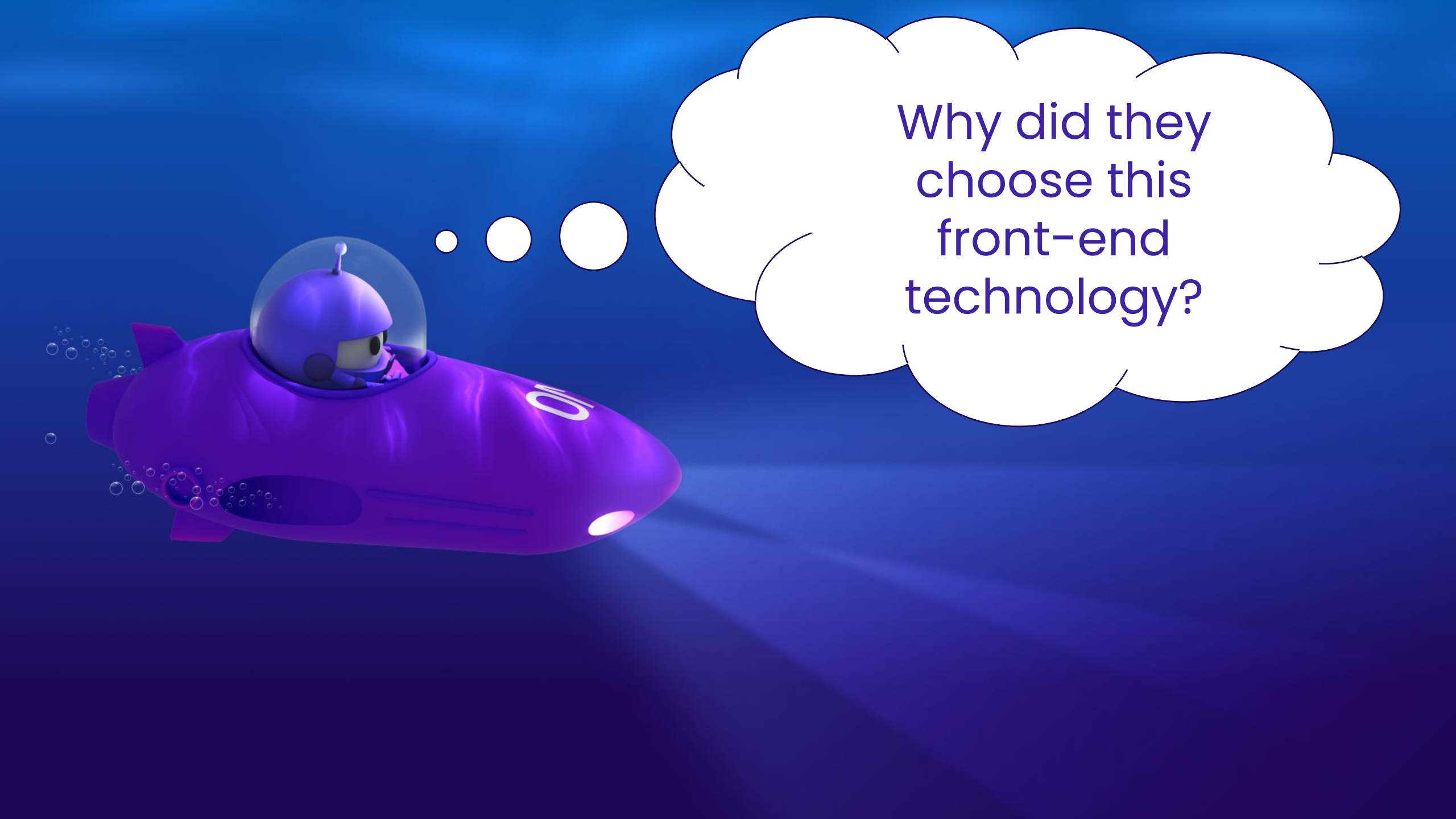



We’ve just got assigned a project.
10

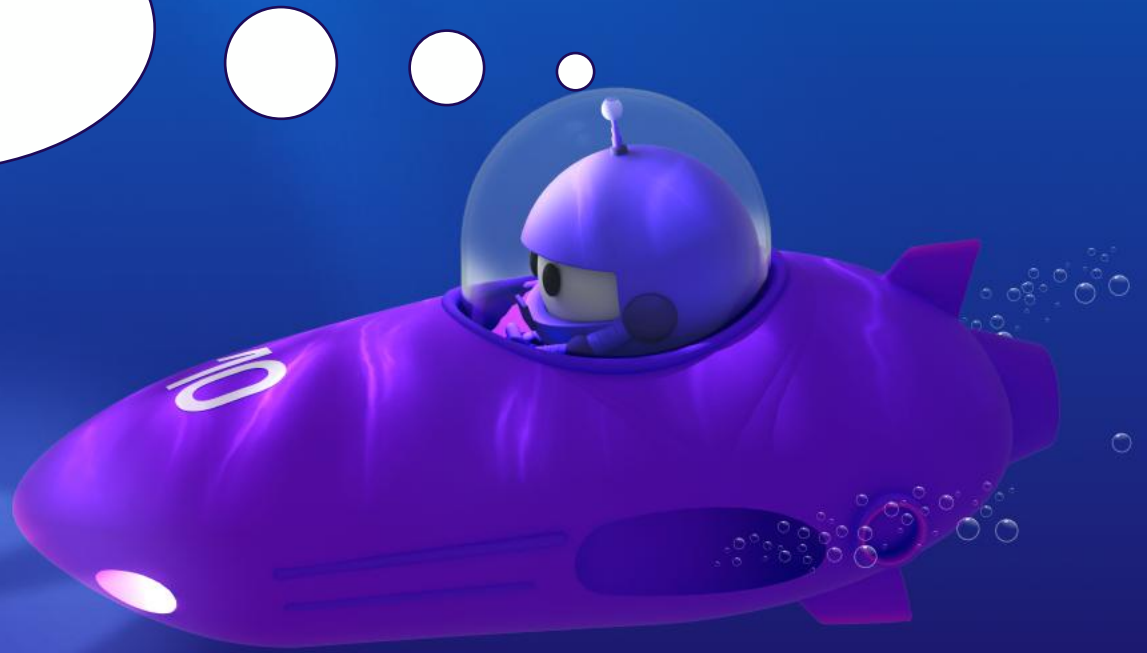

It’s a legacy codebase that needs some updates.
10


Let’s dive into this codebase!
10

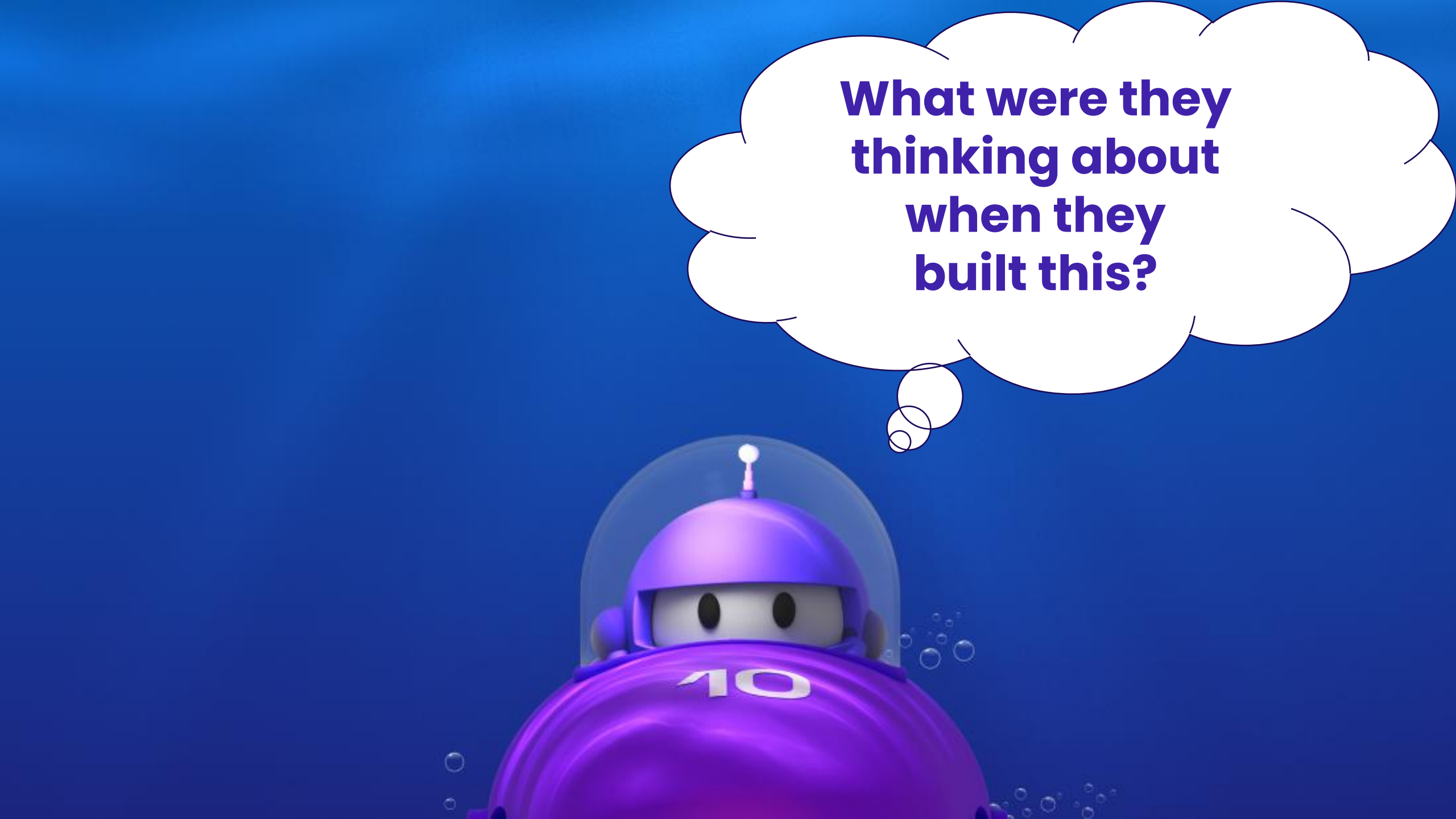

Dive! Dive! Dive!
!!!
10


Why did they choose this front-end technology?

Did they attempt a ***distributed monolith***?

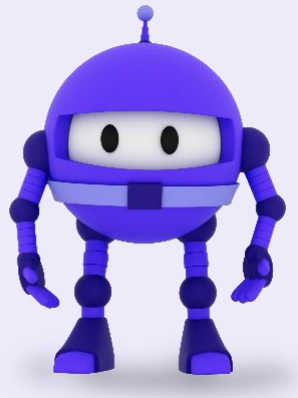


Why does this code seem overly complex?


What were they thinking about when they built this?
10


This could've been better if they had decision records!
10

What are decision records?



Contents of a decision record

Capture what matters – Nygard's Template

Title

Identifier + short description

Status

- Draft
- Proposed
- Accepted
- Superseded
- Obsoleted

Context

Why do we need to make this decision?

Decision

- What was decided?
- What's the justification?

Consequences

What are the trade-offs?

Additional Notes

From compliance guidance to additional supporting information

"

These headings are a **general recommendation**. Evolve your template for **your organization's needs**.

Sadukie

Contents of a Decision Record

Another Example

ADR Format from *Fundamentals of Software Architecture*

TITLE

Identifier + short description

STATUS

Draft, Proposed, Accepted, Superseded, Obsoleted

CONTEXT

What is causing us to make this decision?

DECISION

What is the decision and its justification

CONSEQUENCES

What is the impact? What are the **trade-offs**?

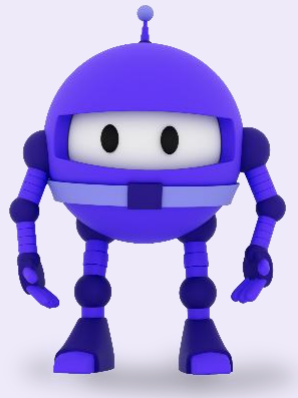
COMPLIANCE

How do we ensure people comply with this decision?

NOTES

Additional supporting information

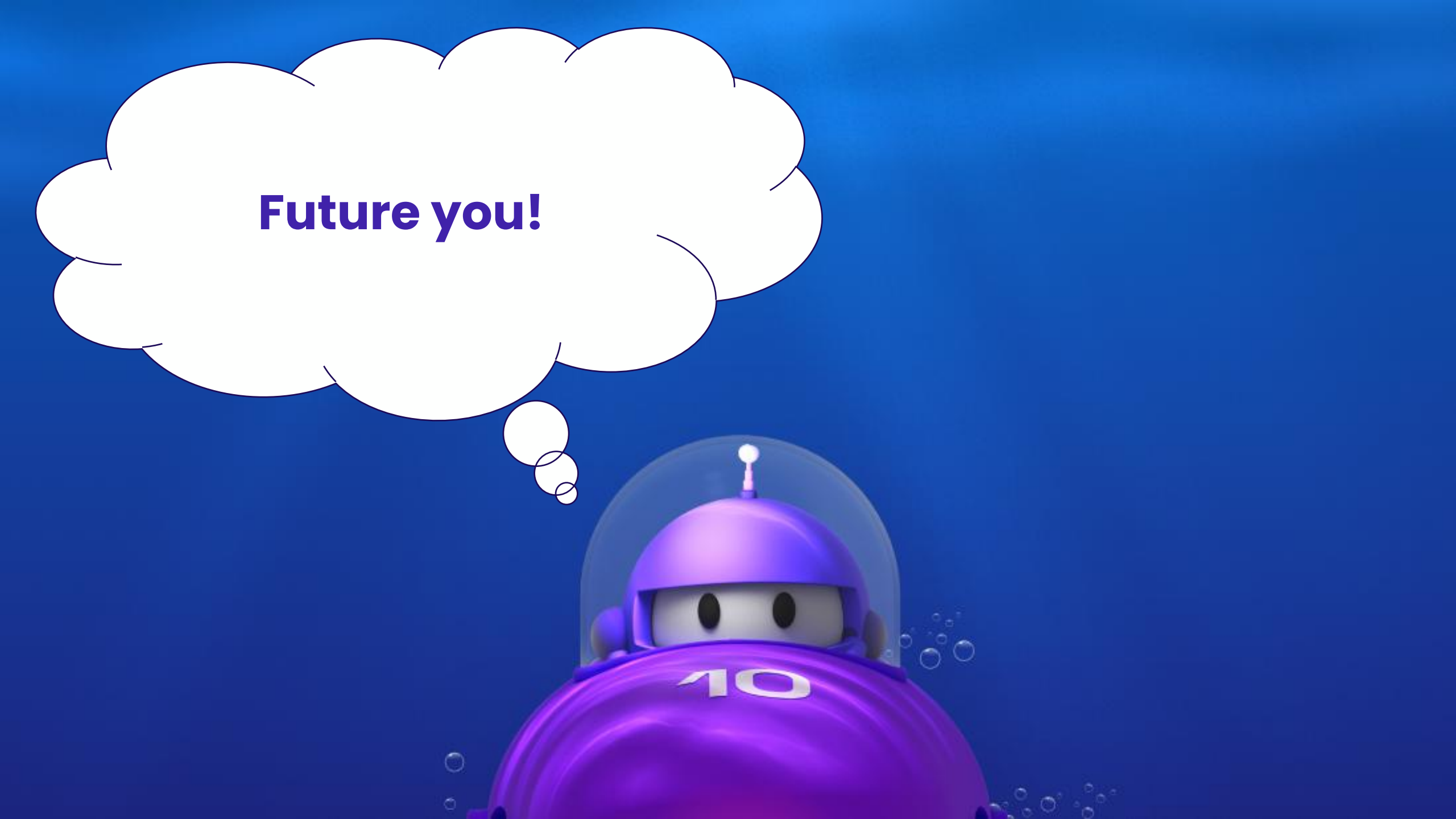
Who are decision records for?



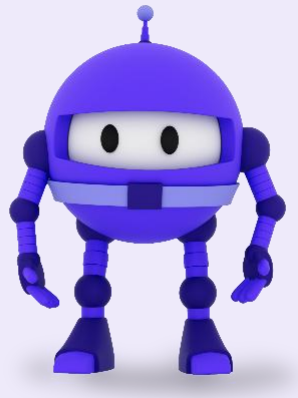

Anyone who is onboarding into a codebase
10

Stakeholders

Decision makers

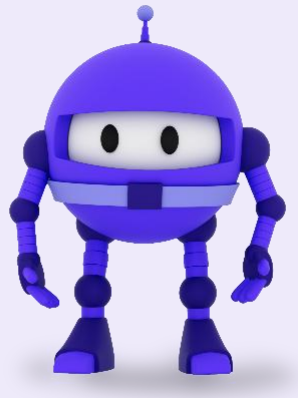

Future you!
10

Which decisions
should be captured?




All of them!
But that feels like it’s too much!
10

Architectural significance of a decision



Factors of architectural significance
Decision records

- Structure
- Architectural characteristics
- Dependencies
- Interfaces
- Construction

Architectural significance

Structure

Impact patterns or styles of architecture.

Examples:

- Modular Monolith, Clean Architecture, Vertical Slices?
- Modules in a modular monolith
- Bounded contexts in microservices

Architectural significance
Architectural characteristics

Impacts any architectural characteristics / “-ilities”

Examples:

- Security
- Performance
- Maintainability
- Scalability
- Auditability
- Usability

Architectural significance
Dependencies

Impacts coupling between components

Examples:

- Direct Database Access vs ORM
- Shared Libraries
- API – Async vs Sync

Architectural significance
Interfaces

Impacts how services and components are accessed

Examples:

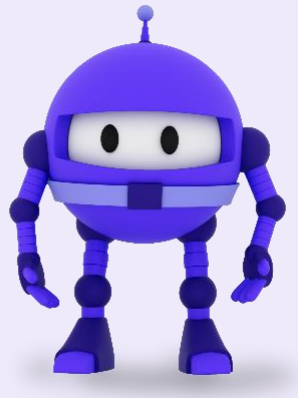
- REST vs GraphQL
- Protocols – gRPC vs HTTP
- Event Streaming

Architectural significance
Construction

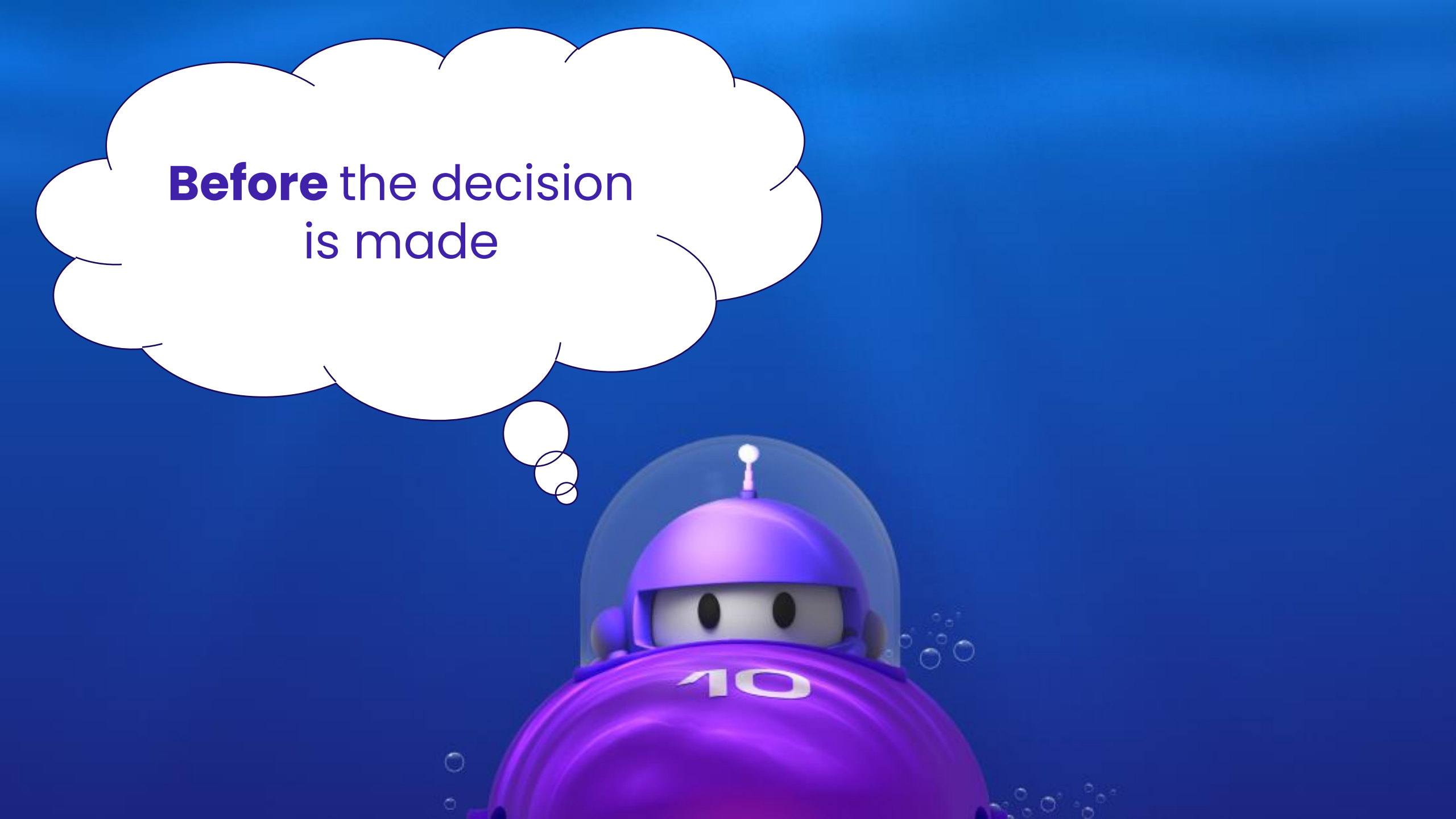
Anything important to the construction of the solution

Examples:
- Platforms
- Frameworks
- Tools
- Processes

When should these decisions be captured?



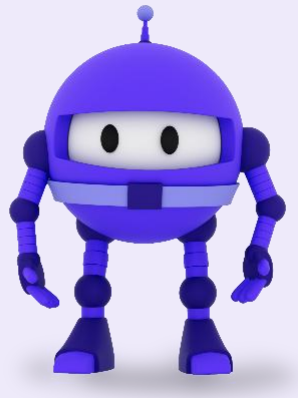

When you have the context!
10


Before the decision is made
10


While the decision
is being discussed
10

After the decision is made

Who should capture these decisions?




Software Architect
10

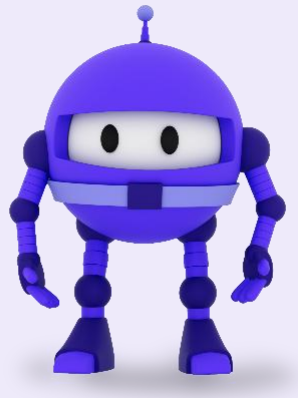

Software Developer
10


Project Manager
10


Someone who is good at documentation
10


Someone who is good with AI!
10

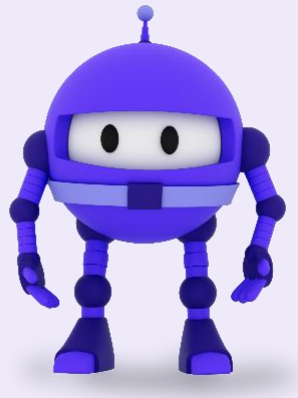
Where should these decisions be stored?



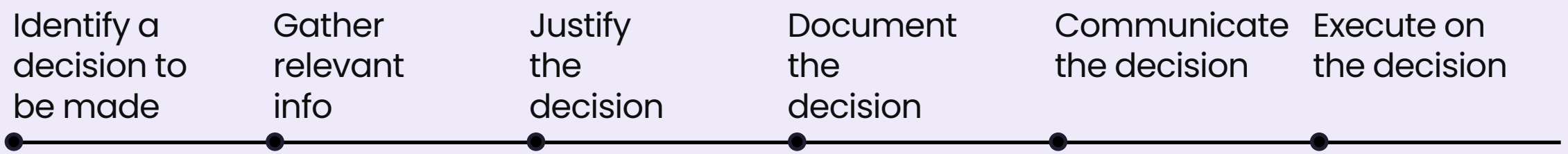
Where to store
Decision records

- In a document store
- In a formal document control system
- In a folder on an intranet
- In a wiki
- **In a place where all who are impacted can review the decision records**

How should these decisions be captured?

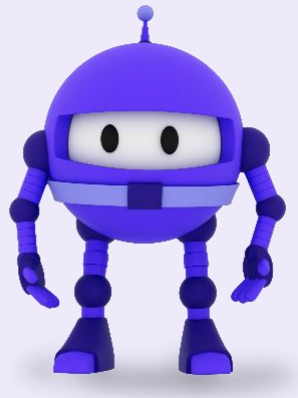


Decision records
Decision process

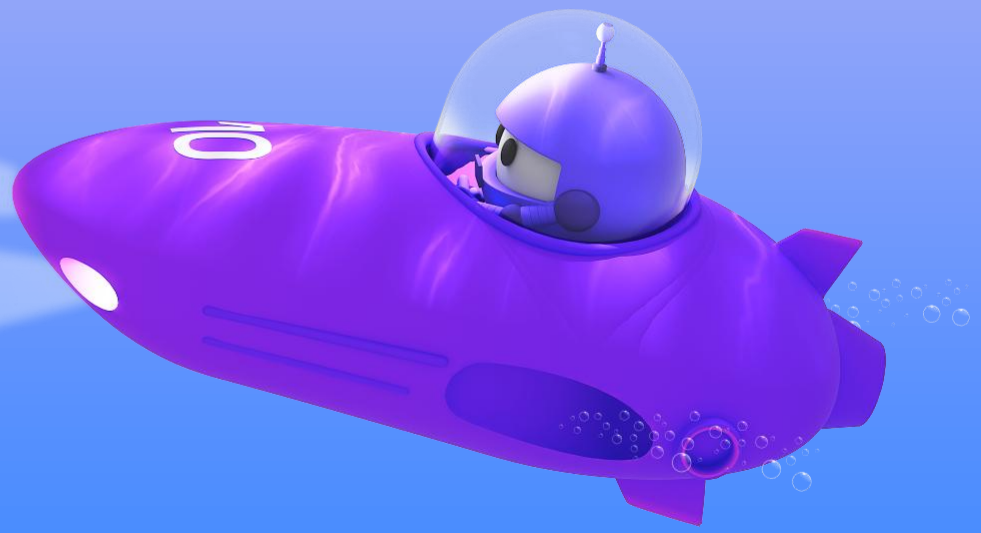


Can be more formalized with Architecture Decision Committees, commonly seen in enterprise architecture organizations

Examples of Decision Records



Example: Security with MFA Options



ADR 003 - Security Decision: Multi-Factor Authentication (MFA)

Architectural Significance: Architectural Characteristic - Security

Date - 2025-11-13

Status

Active

Context

We are establishing a strategy for multi-factor authentication (MFA) to enhance the security of our application and protect against unauthorized access.

Decision

We will implement Multi-Factor Authentication (MFA) requiring at least two factors for all user accounts.

Options Considered

- **Option 1: 2FA via Email & SMS:** Utilizing email and SMS messages as the primary second factor for authentication.
- **Option 2: 2FA via Email & SMS with Time-Based One-Time Passwords (TOTP):** Employing TOTP generated via authenticator apps.
- **Option 3: Full MFA with Biometrics, Authenticator Apps, and Passkeys:** Implementing a comprehensive MFA strategy utilizing biometrics, authenticator apps, and passkeys.

Rationale

- **Security Risks of Email & SMS:** Email and SMS-based 2FA are vulnerable to phishing attacks, SIM swapping, and other interception methods.
- **Stronger Authentication:** We prioritize a more secure authentication approach by leveraging factors resistant to interception, such as biometrics, authenticator apps (TOTP), and passkeys.
- **Authenticator Apps & Passkeys:** These offer a significantly higher level of security compared to SMS and email.

Consequences

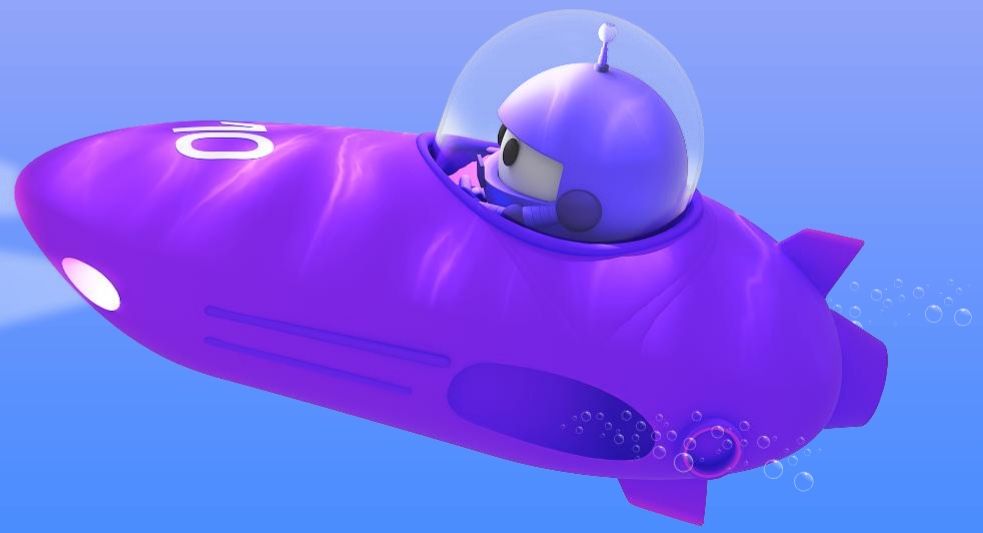
- **Positive:** Significantly reduced risk of unauthorized access, enhanced security posture, and improved compliance.
- **Negative:** Increased complexity for users, potential need for user education and support.

Open Issues

- **User Education:** Developing and providing clear instructions and support for users transitioning to MFA.
- **Passkey Adoption:** Strategizing the rollout and adoption of passkeys as a more secure and user-friendly alternative to traditional authentication methods.

Example: Modular Monolith vs. Clean Architecture

.NET



Architectural Significance: Structure

ADR 001 - Clean Architecture vs. Modular Monolith

Status: Accepted

Context

We were exploring architectural approaches for a new project, specifically considering Clean Architecture versus a Modular Monolith. Early in the process, we lacked a robust understanding of the underlying business domain. Our primary goal was to establish a clear and organized structure for the application.

Decision

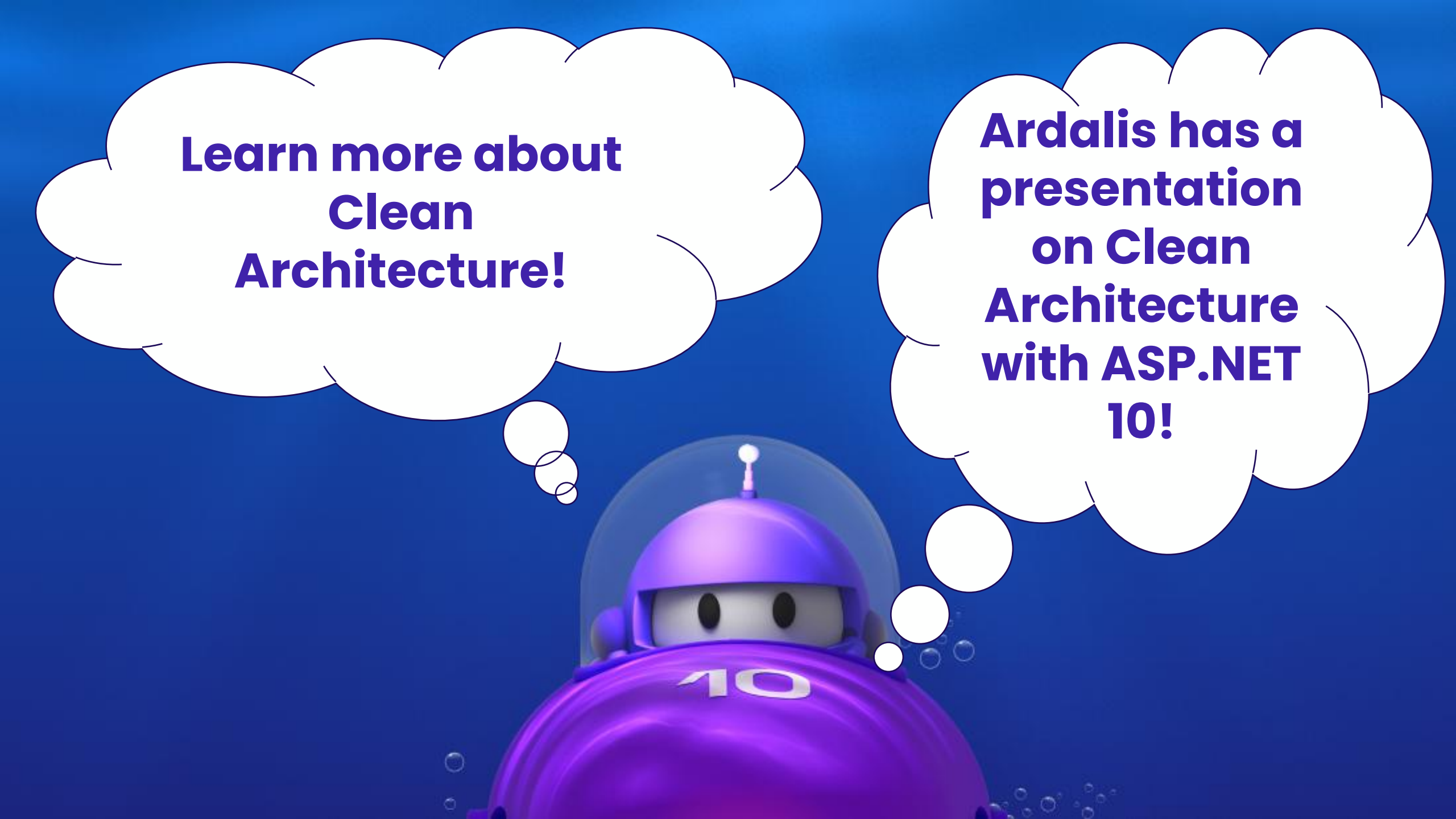
We chose to implement Clean Architecture.

Consequences

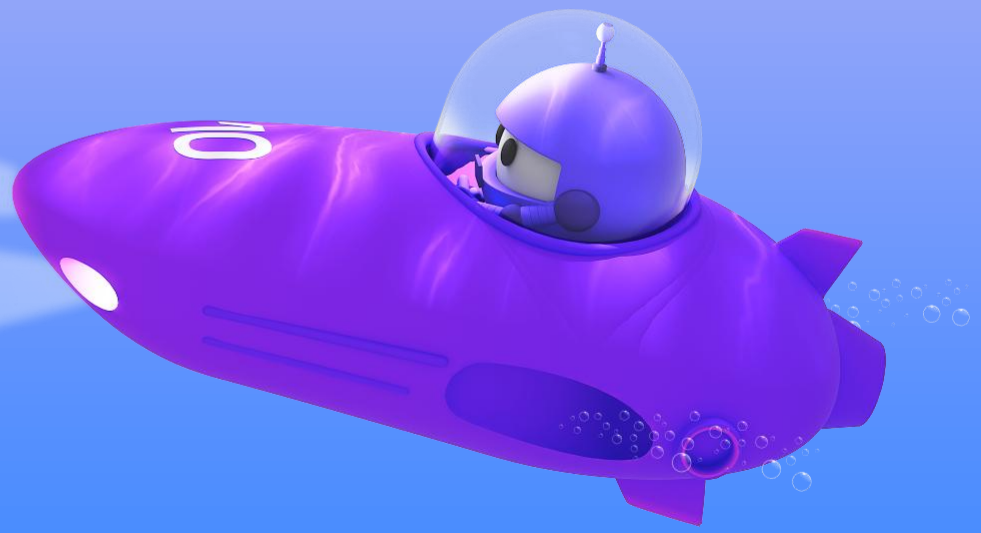
This decision resulted in a loosely coupled, testable application with well-defined boundaries, enabling independent development and easier maintenance over the long term. It also prioritized alignment with evolving business requirements due to its separation of concerns.

Additional Notes

This initial decision was driven by a desire for future flexibility and maintainability, acknowledging the potential risks of a monolithic structure given our limited initial domain knowledge.


Learn more about Clean Architecture!
Ardalis has a presentation on Clean Architecture with ASP.NET 10!
10

Example: Containerization Strategies



ADR 002 - Containerization Strategy

Architectural Significance: Structure

Date: 2025-11-13

Status

Active

Context

We are exploring containerization strategies for our application, considering our use of C# and Blazor. We considered options such as Docker, containerd, and Podman.

Decision

We have chosen to implement our containerization strategy using .NET Aspire.

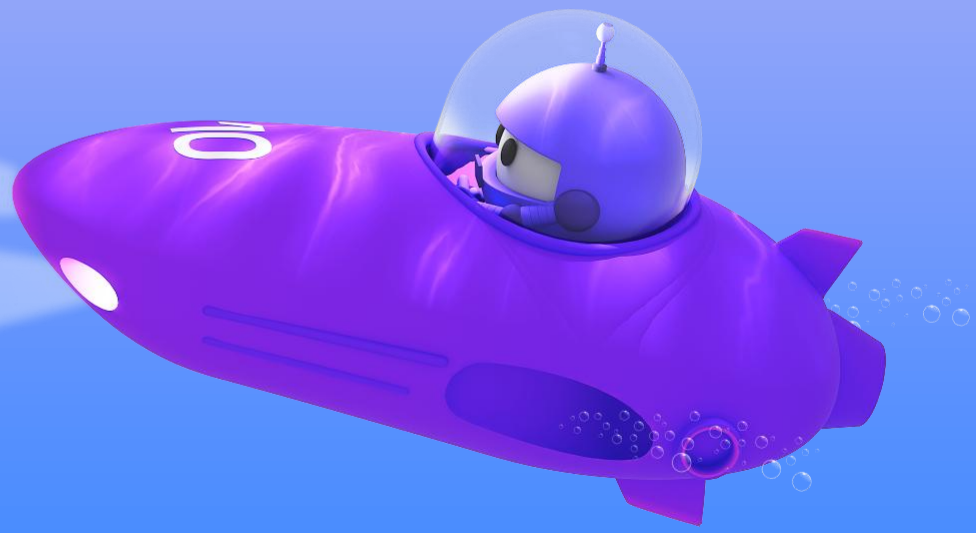
Consequences

- **Positive:** Reduced operational complexity, streamlined development workflow, and improved consistency across environments.
- **Negative:** Initial setup time will be slightly increased, but the long-term benefits outweigh this.

Additional Notes

This decision leverages our existing C# and Blazor expertise while providing a robust framework for managing our containerized application. Crucially, Aspire's seamless integration with the .NET ecosystem – particularly with C# and Blazor – simplifies deployment and management considerably. While we evaluated Docker, containerd, and Podman, Aspire offered a more streamlined experience for our specific .NET application.

Example:
.NET Version



ADR 001 - .NET Version Selection

Architectural Significance: Construction

- **Date:** 2025-07-03
- **Status:** Active
- **Priority:** High

Context

We are facing a decision regarding the .NET version to use for our upcoming project. The release of .NET 10 is imminent, alongside continued support for .NET 8 and .NET 9. A key consideration is the long-term support implications and the risk of rework.

Problem Statement

We need to determine the most appropriate .NET version to adopt, balancing the benefits of newer versions with the stability requirements of our project, particularly regarding support lifecycle and potential upgrade paths to .NET 10.

Decision

We will initially adopt .NET 9 and plan to upgrade to .NET 10 as soon as it's mature and ready for production use.

Options Considered

- **Option 1: .NET 8:** This offers continued support until November 10, 2026. However, it's nearing end-of-life, and starting with a nearly-obsolete version is undesirable.
- **Option 2: .NET 9:** Offers a reasonable support window (May 12, 2026) and a solid foundation for our application.
- **Option 3: .NET 10 (Initial):** While .NET 10 will likely launch in November 2025 with 3 years support, the prematurity of the release and lack of thorough evaluation necessitates a cautious approach.

Rationale

- **Minimize Rework:** Starting with .NET 9 avoids committing to a version that will quickly become obsolete.
- **Long-Term Stability:** .NET 9 provides a sufficient support window for our project's foreseeable needs.
- **Flexibility:** Allows us to readily upgrade to .NET 10 when it's stabilized and proven viable, aligning with Microsoft's support patterns.
- **Steve's Insight:** Steve's decision to upgrade to .NET 10 as soon as possible is a key driver.

Consequences

Positive

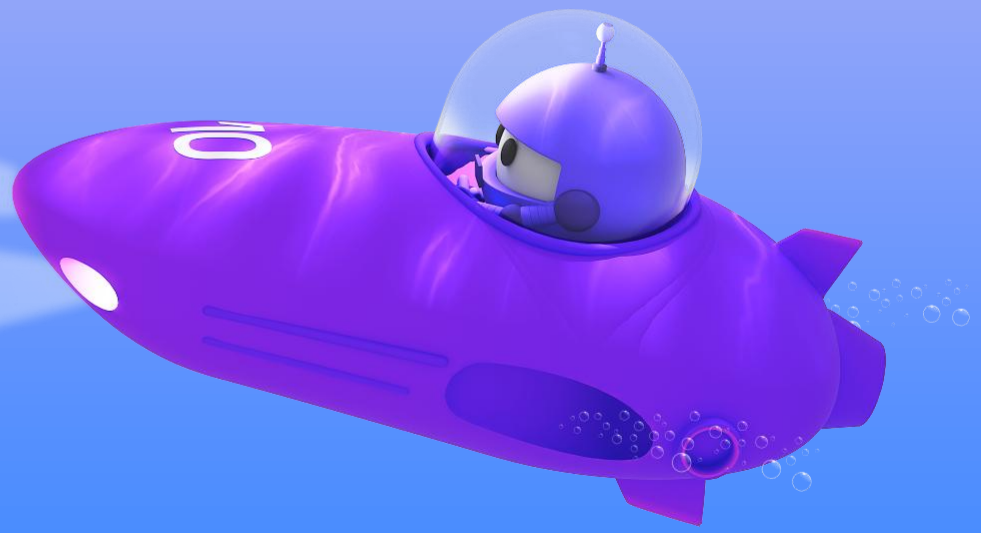
```
- Reduced risk of needing to rework the application due to .NET version obsolescence.
- Greater flexibility to adopt .NET 10 when it's mature.
```

Negative

```
- Potential for some functionality to be dependent on .NET 9 specific features. (Requires thorough assessment of dependencies).
- Potential for rework if there are breaking changes in .NET 10.
```

Example: Handling Domain Events in C#

.NET



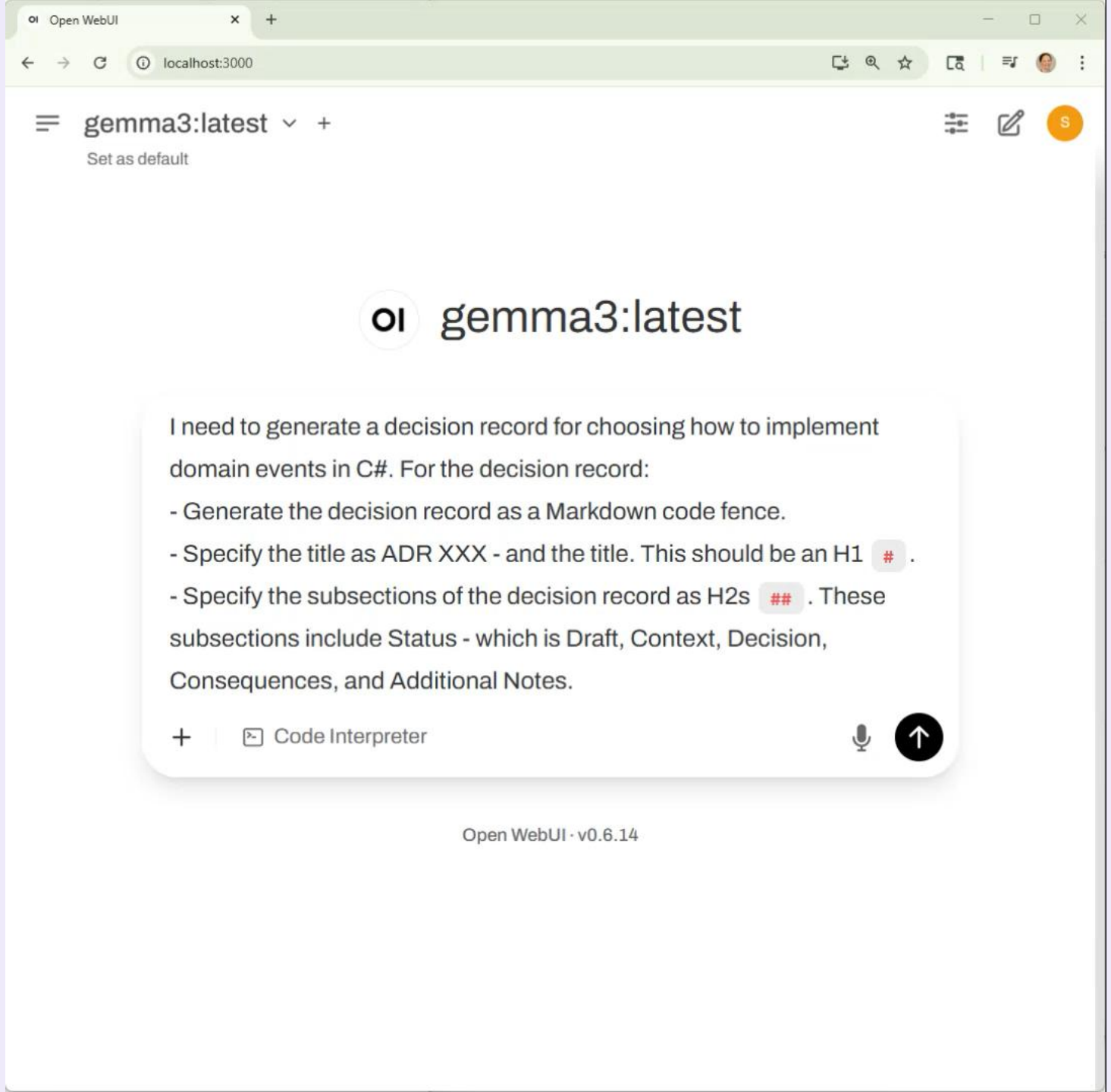
Using AI to Generate Decision Records
Decision Records

Be explicit

- Specify the headings in a particular order
- Specify the format – so something specific like **Markdown code fence** or **rich-text format**

Walk it through your decision

- Use a model that has **summarize** capabilities for better success
- Talk through your decision and let it organize into the desired format.


Open WebUI
localhost:3000
gemma3:latest
Set as default
gemma3:latest
I need to generate a decision record for choosing how to implement domain events in C#. For the decision record:
- Generate the decision record as a Markdown code fence.
- Specify the title as ADR XXX - and the title. This should be an H1 #.
- Specify the subsections of the decision record as H2s ##. These subsections include Status - which is Draft, Context, Decision, Consequences, and Additional Notes.
Code Interpreter
Open WebUI · v0.6.14

ADR 001 - Domain Event Implementation

Status - Draft

Context

We're investigating the best approach for generating and handling domain events in our C# project. Several options were considered, ranging from full-fledged event bus frameworks to simpler, more lightweight solutions. The primary goal is to decouple our domain logic from presentation layers and enable asynchronous communication between components. We need a solution that is manageable, well-suited for a project of this scale, and doesn't introduce unnecessary licensing costs.

Decision

We have decided to implement domain events using **FastEndpoints**.

Consequences

- **Reduced Complexity:** FastEndpoints simplifies the event generation and handling process compared to heavier frameworks like NServiceBus or MediatR.
- **Cost-Effective:** FastEndpoints is open-source and free to use, eliminating licensing concerns that would have been associated with NServiceBus and MediatR.
- **Integration:** FastEndpoints is already being used within this project for managing API endpoints, creating a cohesive development experience. This minimizes the learning curve and integration effort.
- **Suitable Scale:** FastEndpoints is designed for smaller projects like this, offering a sensible balance between features and complexity.
- **Performance:** FastEndpoints' design focuses on performance and efficiency.

Additional Notes

- We evaluated NServiceBus and MediatR, which provide robust eventing capabilities but carry licensing costs that would likely be prohibitive for this project.
- The decision was made with consideration for the long-term maintainability and operational costs of the eventing infrastructure. A heavier framework would require more overhead for configuration, monitoring, and potential upgrades.
- We will explore FastEndpoints' features (e.g., automatic serializing/deserializing of event data) to optimize our implementation.

Additional References
Get Started with Decision Records today!

⬆️ Gist with Resource Links

adr-tools command line tool

Joel Parker Henderson's GitHub repo on ADRs

Fundamentals of Software Architecture

Architecture decision record - Microsoft Azure Well-Architected Framework

Cognitect – Documenting Architecture Decisions (Nygard)

Architecture Decision Records - AWS Prescriptive Guidance

Document your decisions today!

Thanks for tuning in!

Sarah “sadukie” Dutkiewicz
Senior Trainer
NimblePros
📨 sarah.dutkiewicz@nimblepros.com

🌐 https://linkedin.com/in/sadukie