# Architecture Decision Record: Rust programming language

Decision Number: AR-001

Decision Title: Adoption of Rust programming language

Date: December 1, 2021

Status: Accepted

## Problem Statement

As we continue to develop software applications, we have observed that it is increasingly challenging to mitigate potential security vulnerabilities and prevent runtime errors. With the existing programming languages, such as C and C++, we continue to experience issues such as buffer overflows, memory leaks, and undefined behavior leading to applications' crashes. We require a programming language that provides memory safety guarantees and is efficient enough to support performance-critical applications.

## Considerations

Several programming languages are designed to address the existing problems. Among them, Rust programming language has gained significant attention from the developers' community due to its unique design features. Considerations include;

1. Memory safety and security
2. Performance and efficiency
3. Community support and adoption
4. Learning curve
5. Tools and ecosystem
6. Compatibility with existing software systems.

## Constraints

Adopting a new programming language requires retraining developers, which takes time and resources. Integrating the language into the existing development workflow may be a challenge. We must ensure compatibility with the existing systems and avoid breaking changes to maintain continuity.

## Implementation

1. Our development team will undergo training to learn and familiarize themselves with the Rust programming language.
2. We will create a new project using Rust on a trial basis to evaluate its compatibility and suitability for our development purposes.

3. We will gradually migrate existing systems written in C and C++ to Rust.
4. We will collaborate with the Rust community to explore the available tools and libraries that can enhance our development workflow.
5. We will monitor the performance of Rust and compare it to the performance of the existing programming languages regularly.
6. We will adopt a long-term approach that balances the costs of training, integration, and potential benefits of using Rust.

## Rationale

We have adopted Rust due to its unique features designed to provide memory safety and security guarantees while maintaining performance and efficiency. Rust's robust type system, borrow checker, and memory safety concepts make it highly suitable for developing performance-critical and safety-critical applications. Moreover, Rust has a significant community of developers, enabling us to access a wide range of tools, libraries, and ecosystem that support our development workflow. Although Rust comes with a learning curve, we believe the benefits of adopting Rust outweigh the costs and provide an excellent opportunity for continued growth and innovation.

## Consequences

1. The adoption of Rust will require a significant investment in time and resources to train developers and integrate the language into the existing development workflow.
2. Adopting Rust may cause some degree of compatibility issues with existing systems, requiring refactoring, and modifications.
3. Rust's adoption may increase the number of developers who can contribute to our project by attracting Rust developers who want to work on exciting projects.
4. The adoption could lead to improved performance, efficiency, and safety as compared to the existing languages.
5. Finally, adopting Rust comes with the potential benefit of reducing security vulnerabilities in our applications.

**Credit: this page is generated by ChatGPT, then edited for clarity and format.**