Project Idea: MERN to Rust Backend Migration for Streaming App

Paramvir Singh C0912101, Lovepreet Kaur C0902873, Akshveer Kaur C0908937,

Gagandeep Kaur C0910427, Navjot Kaur C0904082

Project Group 3

CSD 3423 - Introduction to Project Management 02

Lambton College

Dalia Haggag

February 10, 25

Project Idea: MERN to Rust Backend Migration for Streaming App

Project Description: In this project we are assuming that we are running a streaming website which is initially made on MERN (Mongo DB as database, express JS for database connection and React JS as front-end to create user interface and lastly node JS). Now as we are getting more and more users on our streaming platform we found that our Backend which is JavaScript based of the application is not fast enough, reliable and efficient. The project involves migrating the backend of an existing streaming application from JavaScript (Node.js) to Rust to improve performance, scalability, and efficiency. The goal is to leverage Rust's memory safety and performance advantages to handle increased user traffic and provide better resource management, especially for CPU and memory-heavy operations. This migration will involve rewriting the backend APIs, integrating Rust with the current MERN stack (MongoDB, Express, React, Node.js), and ensuring minimal disruption to the existing frontend and database.

Objectives:

<u>Backend Migration</u>: Transferring the backend from Node JS to Rust, ensuring the system's logic and functionality remains same.

<u>Improve Performance</u>: Enhance performance, particularly in terms of handling more simultaneous streaming requests, processing data faster, and reducing server load.

Maintain API Integrity: Ensure that the Rust backend mimics the same API structure as the Node.js version to avoid major frontend changes.

Optimize Resource Management: Use Rust's low-level control over system resources to reduce latency and memory consumption.

Seamless Transition: Achieve a seamless migration process with minimal downtime for the existing users.

Testing and Debugging: Perform comprehensive testing to identify and resolve any bugs or compatibility issues introduced during the migration.

Expected Budget Needed:

Total Estimated Budget: $30,000

Development costs (Rust development team, training): $12,000

Infrastructure & Deployment costs (server resources for testing, CI/CD setup): $8,000

Testing (QA tools, automated testing setup): $4,000

Consulting fees (Rust experts for code review and optimization): $3,000

Miscellaneous expenses (licenses, tools for integration): $3,000

Time Frame:

Project Start Date: March 03, 2025

Project End Date: March 17, 2026

Total Duration: 12 Months and 14 days

Resources Needed:

Human Resources:

Project Manager: To oversee the migration process and manage timelines.

Rust Developers: A team of developers familiar with Rust and capable of writing efficient backend code.

Frontend Developers: To ensure the frontend works seamlessly with the new Rust-based APIs.

DevOps/Infrastructure Engineers: For setting up the servers, deployment pipelines, and managing infrastructure.

QA/Test Engineers: To test the system extensively after migration for bugs, performance issues, and compatibility.

Rust Consultants (Optional): Experts for code reviews, performance tuning, and advice on best practices.

Material Resources:

Servers/Cloud Services: AWS, Digital Ocean, or similar for development and staging environments.

Development Tools: IDEs and text editors (e.g., VSCode, JetBrains), Rust toolchain (cargo, rustup).

Database Integration Tools: For MongoDB and any required integrations with Rust.

Technology Resources:

Rust Development Frameworks: Actix or Rocket for building the backend APIs in Rust.

Testing Tools: Rust-specific testing tools (e.g., Cargo test), as well as integration tests and load testing tools.

CI/CD Tools: GitHub Actions, Jenkins, or GitLab CI to automate testing and deployment.

Monitoring Tools: Use monitoring tools like Prometheus or Grafana to track application performance and issues post-migration.

Profit Expected (Value Created):

Direct Profit: Enhanced system performance and scalability, potentially reducing costs for server infrastructure due to better resource management. Also, improved user experience can lead to increased retention and subscriptions.

Long-term Benefits: The Rust-based backend will be easier to maintain in the long run with fewer bugs and performance bottlenecks, which means lower operational costs and fewer disruptions for end-users.

Increased Brand Reputation: As Rust is known for its performance and reliability, being one of the first to leverage it for a streaming app might differentiate the project and gain attention in the tech industry.

Revenue Growth: Enhanced performance could support higher user capacity and new features, which may attract more subscribers or advertisers, leading to increased revenue.