# **ADR-001 – Stack de Tecnologias para o Back-end**

* **Status:** Aceito
* **Data:** 2025-05-19
* **Autor:** Carlos H. P. Jacinto

## **Contexto**

A aplicação bancária em desenvolvimento requer uma API back-end **robusta, segura e bem estruturada**, capaz de realizar operações como:

* Cadastro de usuários
* Autenticação
* Criação e movimentação de contas bancárias
* Verificação do histórico de movimentações

Além disso, o projeto precisa garantir:

* **Organização de código seguindo boas práticas**
* **Facilidade para testes automatizados**
* **Escalabilidade a longo prazo**
* **Ambiente de desenvolvimento acessível e produtivo**

Diante disso, foi necessário definir uma stack de tecnologias que promovesse **legibilidade, modularidade e produtividade**.

## **Decisão**

A stack de back-end adotada foi composta pelas seguintes tecnologias e abordagens:

### **Node.js + NestJS + Express**

Plataforma leve e performática com o **framework NestJS**, que oferece estrutura modular, injeção de dependência, validação integrada e excelente suporte a TypeScript.  
 Internamente, o NestJS utiliza o **Express** como camada HTTP.

### **TypeScript**

Linguagem com tipagem estática que torna o código mais **seguro, legível e previsível**.

### **MySQL + TypeORM**

Banco de dados relacional robusto, com suporte amplo e boa documentação.  
 O **TypeORM** foi adotado como ORM por sua **integração nativa com NestJS** e uso de decorators, facilitando a modelagem orientada a objetos.

### **Programação Orientada a Objetos (POO)**

Estruturação do sistema com **classes e entidades (ex: User, BankAccount, BankAccountHistory)**, garantindo modularidade, reusabilidade e organização.

### **Princípios de Clean Code**

Adoção de boas práticas que favorecem a **manutenção e clareza do código**.

### **Testes com Jest**

Utilizado para testes unitários das regras de negócio, garantindo **confiabilidade e cobertura de código**.

### **Git**

Ferramenta de controle de versão distribuído para **colaboração, rastreamento e versionamento de código**.

## **Consequências**

### **Vantagens**

* Arquitetura **modular, escalável e testável**
* Código limpo, bem tipado e com **responsabilidades bem definidas**
* Integração fluida com **banco de dados relacional via ORM**
* Ferramentas com **comunidade ativa e excelente documentação**
* Ambiente de desenvolvimento produtivo para **novos integrantes da equipe**

### **Desvantagens**

* **TypeORM** pode apresentar complexidade em casos de relacionamentos avançados.
* O **setup do MySQL** pode demandar configurações extras em ambientes de produção.
* O **NestJS**, embora poderoso, possui **curva de aprendizado inicial** para quem vem do Express puro.

## **Alternativas Consideradas**

### **Sequelize**

Alternativa ao TypeORM com suporte a múltiplos bancos, porém com **menor integração com TypeScript** e abordagem menos orientada a objetos.

### **Prisma ORM**

ORM moderno e eficiente, considerado pela sua **tipagem forte e excelente DX**. Foi preterido inicialmente em favor do TypeORM por **integração já consolidada com o NestJS**.

### **PostgreSQL**

Banco de dados robusto e altamente escalável, porém o **MySQL foi escolhido pela familiaridade** e facilidade de configuração local.

### **NestJS vs. Express puro**

Apesar do NestJS utilizar o Express internamente, sua **estrutura modular, organização e suporte nativo a boas práticas** motivaram sua escolha em detrimento do uso do Express isoladamente.

## **📚 Referências**

* [NestJS](https://nestjs.com/)
* [TypeORM](https://typeorm.io/)
* [MySQL](https://dev.mysql.com/)
* [Jest](https://jestjs.io/)
* [Prisma ORM (futuro considerado)](https://www.prisma.io/)