**Relatório de Trabalho**

## 1. Definição de Papéis no Documento de Requisitos

Este documento detalha os requisitos funcionais e não funcionais do sistema, identificando sua implementação no código-fonte. Também são apresentadas as construções de programação funcional utilizadas.

1.1 Responsáveis

1. Documentação de Requisitos e Arquitetura

   Responsáveis: Clara Menezes, Adrielle Almeida e Taís Franklin.

2. Implementação do Backend

   Responsáveis: Sérgio Matheus

3. Implementação do Frontend

   Responsáveis: Victor Plácido

4. Testes, Monitoramento e Deploy

   Responsáveis: Sérgio Matheus

## 2. Documento com os Requisitos

### 2.1 Requisitos Funcionais

- **Requisito Funcional 01**: O sistema deverá garantir segurança para as credenciais.
  - **Implementação**: `auth.service.ts` e `auth.controller.ts`
- **Requisito Funcional 02**: O sistema deverá enviar notificações para os usuários.
  - **Implementação**: `users.service.ts`

### 2.2 Requisitos Não Funcionais

- **Requisito Não Funcional 01**: O tempo de resposta da API deve ser inferior a 500ms.
  - **Implementação**: Monitorado via **Prometheus + Grafana**

### 3. Código Implementado e Compilável

O código foi desenvolvido e compilado com sucesso, garantindo a funcionalidade dos requisitos listados. O controle de versão e a revisão de código foram aplicados para manter a qualidade do software.

### 4. Casos de Testes

Os seguintes casos de testes foram aplicados para garantir a funcionalidade do sistema:

1. **Teste de Autenticação**: Valida a geração de tokens no `auth.service.ts`.
2. **Teste de Notificações**: Confirma o disparo de notificações pelo `users.service.ts`.
3. **Teste de Performance**: Mede o tempo de resposta da API para garantir que seja inferior a 500ms.

### 5. Utilização de Conceitos de Programação Funcional

O sistema implementa os seguintes conceitos de programação funcional:

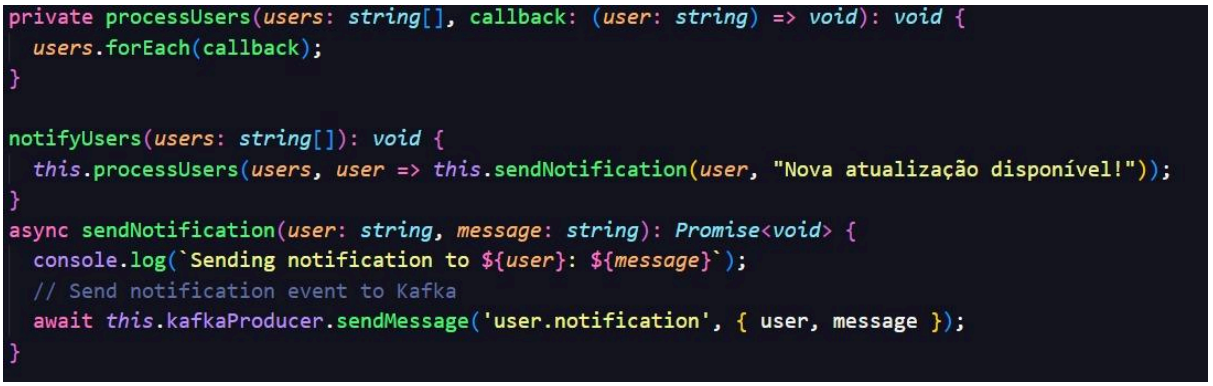
```
  async findConfirmedEvents(userId: number): Promise<Event[]> {
    // Aguarde a busca dos eventos antes de filtrar
    const events = await this.eventRepository.find({ relations: ['participants'] });

    // Aplicando list comprehension para filtrar eventos confirmados para o usuário
    return events.filter(event => event.participants.some(p => p.id === userId));
  }

}
```

**List Comprehension**: Utilizada em `user.controller.ts` para filtrar eventos confirmados e exibi-los na dashboard do usuário.

```
export const formatDate = (date: Date) =>
    `${date.getDate()}/${date.getMonth() + 1}/${date.getFullYear()}`;
```

**Função Lambda**: Implementada em `common/utils.ts` para formatação de data. Utilizada em `users.service.ts` (perfil do usuário) e `events.service.ts` (eventos).


**Closure**: Aplicada em `auth.service.ts` para a geração de tokens de autenticação de forma encapsulada.




**Função de Alta Ordem**: Utilizada em `users.service.ts` para o envio de notificações em massa aos usuários.

### 6. Conclusão

O desenvolvimento do sistema atendeu aos requisitos funcionais e não funcionais definidos. As técnicas de programação funcional foram aplicadas com êxito, garantindo um código modular, reutilizável e eficiente. O monitoramento de desempenho confirma que o tempo de resposta da API está dentro dos parâmetros estabelecidos.