

# RELATÓRIO DO TRABALHO DE ENGENHARIA DE SOFTWARE SOBRE ARQUITETURAS(MVC, MVP, MVVM, REST e Reativo)

**Alunos: Luis Fernando Soares Sales, Victor Bruno Freire, Marcos Vinicius  
Bandeira e Raimundo Edson Miranda do Santo**

**No decorrer do documento serão respondidas as seguintes perguntas:**

1. Onde ficou a maior parte da lógica em cada arquitetura?
2. Qual arquitetura foi mais simples de integrar com o backend reativo?
3. O que mudou no frontend ao trocar REST por reativo?
4. Qual arquitetura você escolheria para um sistema maior? Por quê?

## **1 - Onde ficou a maior parte da lógica em cada arquitetura?**

### **Frontend:**

- **MVC:**
  - TaskController.ts
  - TaskModel.ts
- **MVP:**
  - TaskModel.ts
  - TaskPresenter.ts
- **MVVM:**
  - TaskModel.ts
  - useTaskViewModel.ts

### **Backend:**

- **REST & Reativo:**
  - TaskService.java
  - TaskEventPublisher.java

## **2 - Qual arquitetura foi mais simples de integrar com o backend reativo?**

A arquitetura mais simples para integrar com o backend reativo foi a **MVVM**, pois sua organização em pastas e arquivos já apresenta uma boa compatibilidade com a reatividade do servidor. Nessa arquitetura, a **View** é responsável apenas por reagir às mudanças de estado expostas pelo **ViewModel**, sem conter lógica de integração com o backend.

Além disso, a natureza reativa do **useTaskViewModel** se integra diretamente a fluxos de eventos, como Server-Sent Events (SSE), permitindo que a interface seja atualizada automaticamente sempre que novos eventos são recebidos, sem a necessidade de recarregar a página.

### 3 - O que mudou no frontend ao trocar REST por reativo?

No frontend, a mudança de **REST** para reativo substituiu um modelo baseado em requisições — um modelo que depende completamente do usuário para as interações com o backend — por um modelo orientado a eventos. Isso reduziu a necessidade de refetch de dados, simplificou a View e permitiu atualizações automáticas da interface em tempo real.

#### Mudanças de código:

- REST tradicional:

```
async getAll(): Promise<Task[]> {
  const res = await api.get('/task')
  return res.data
}

async create(task: Omit<Task, 'id' | 'createdAt>): Promise<Task> {
  const res = await api.post('/task', task)
  return res.data
}
```

- Reativo com SSE:

```
async init() {
  this.tasks = await this.model.getAll()
  this.view.renderTasks(this.tasks)

  this.unsubscribe = this.model.subscribeToEvents(event => {
    if (event.type === 'CREATED' && event.task) {
      const exists = this.tasks.some(t => t.id === event.task.id)

      if (!exists) {
        this.tasks = [...this.tasks, event.task]
        this.view.renderTasks(this.tasks)
      }
    }
  })
}
```

### 4 - Qual arquitetura você escolheria para um sistema maior? Por quê?

Escolheríamos a arquitetura **MVVM**, pois ela apresenta maior escalabilidade em projetos de grande porte a longo prazo, além de possuir melhor compatibilidade com um backend reativo. A **MVVM** também oferece maior modularização, o que facilita a manutenção, a evolução do sistema e a reutilização de código, tornando a aplicação mais organizada e sustentável conforme sua complexidade aumenta.