Guia de Conteúdo para a Primeira Semana: Introdução a Sistemas Corporativos e Microserviços

Introdução: Bem-vindos ao Mundo dos Sistemas Corporativos

Esta semana marca o início de uma jornada fundamental no campo da engenharia de software, focando nos sistemas corporativos e na ascensão das arquiteturas de microsserviços. O objetivo principal é fornecer uma base sólida para a compreensão de como as organizações modernas constroem e gerenciam suas infraestruturas digitais. Ao final desta semana, os participantes terão uma visão clara do contexto dos sistemas corporativos, entenderão a motivação por trás da arquitetura de microsserviços como uma solução contemporânea e serão capazes de identificar as distinções cruciais entre sistemas monolíticos e distribuídos. Além disso, a semana servirá para introduzir o projeto prático semestral – uma plataforma de eventos corporativos – e para orientar a configuração do ambiente de desenvolvimento, garantindo que todos estejam preparados para as atividades práticas que se seguirão.

A relevância dos sistemas corporativos na indústria moderna não pode ser subestimada. Eles constituem a espinha dorsal de qualquer organização, independentemente de seu porte ou setor de atuação. Essas ferramentas digitais são essenciais para orientar e apoiar o trabalho, viabilizando a gestão de tarefas, rotinas e processos de negócios de maneira eficiente e confiável.[1] Por meio da informatização e da atualização de procedimentos internos, esses sistemas têm a capacidade de otimizar operações, digitalizando processos para eliminar o uso de papel e a movimentação física de documentos, além de permitir a criação de fluxos de trabalho personalizados.[1] Essa modernização pode levar a uma economia considerável e a um controle abrangente sobre todos os departamentos, resultando em uma melhoria significativa da produtividade geral da empresa.[2]

1. Fundamentos dos Sistemas Corporativos

Para compreender a evolução das arquiteturas de software, é imperativo primeiro definir o que são sistemas corporativos e quais características são essenciais para seu sucesso.

1.1. Definição e Propósito dos Sistemas Corporativos

Sistemas corporativos são ferramentas digitais abrangentes, meticulosamente projetadas para viabilizar a gestão de tarefas e rotinas em uma empresa, assegurando que objetivos específicos sejam alcançados de forma eficiente e confiável.[1] Eles são, em essência, um conjunto interligado de ferramentas, processos e estratégias que operam em conjunto para produzir um resultado ou serviço específico.[1]

Tipicamente, um sistema corporativo pode ser visualizado em camadas distintas, cada uma com um propósito bem definido [2]:

- **Camada de Aplicação:** Esta é a interface "visível" aos usuários, onde as funcionalidades, processos, cadastros e os dados operacionais do dia a dia da empresa são manipulados. É a parte do sistema com a qual os usuários interagem diretamente.
- **Camada de Armazenamento de Informações (Dados):** Aqui, as informações geradas na camada de aplicação são guardadas de forma persistente. O armazenamento pode ser nativo, residindo na infraestrutura da própria empresa, ou remoto, como em modelos de computação em nuvem, oferecendo flexibilidade e escalabilidade.
- **Camada de Framework:** Esta camada contém as configurações e parametrizações da solução. É o local onde o sistema pode ser customizado e personalizado de acordo com as necessidades específicas da operação da empresa, permitindo a construção de novo código-fonte ou a modificação do existente para atender a requisitos únicos.

1.2. Características Essenciais: Escala, Modularidade e Manutenção

A eficácia de um sistema corporativo é intrinsecamente ligada a três características cruciais: escala, modularidade e manutenção.

- **Escala:** Refere-se à capacidade de um sistema de crescer e se adaptar a um aumento significativo de usuários, volume de dados ou funcionalidades sem que haja degradação de desempenho. Em ambientes corporativos, a demanda por escalabilidade é uma constante, impulsionada pelo crescimento contínuo dos negócios e pela necessidade de suportar cargas de trabalho cada vez maiores.
- **Modularidade:** Esta propriedade descreve um sistema composto por componentes independentes e bem definidos que podem ser desenvolvidos, testados, implantados e mantidos separadamente. Um sistema modular oferece uma flexibilidade notável e alta capacidade de personalização, permitindo que componentes sejam adicionados ou removidos sem comprometer o funcionamento geral.[3] Isso não apenas garante maior eficiência e controle, pois cada empresa pode configurar o sistema de acordo com suas demandas específicas, mas também facilita a manutenção e a expansão da estrutura de TI, eliminando a necessidade de grandes mudanças no sistema como um todo.[3] O conceito de modularidade se estende à infraestrutura, como em servidores modulares (servidores blade), que permitem a substituição ou atualização de hardware de forma "trocável a quente" (hot-swappable). Essa capacidade é vital para datacenters, onde o tempo de inatividade é inaceitável.[4]
- **Manutenção:** Diz respeito à facilidade com que um sistema pode ser modificado, corrigido, atualizado ou aprimorado ao longo do tempo. A modularidade é um pilar fundamental para a manutenibilidade, pois permite isolar problemas em módulos específicos e aplicar atualizações pontuais sem afetar a totalidade do sistema.[3]

A modularidade emerge como um pilar estratégico para a resiliência e agilidade empresarial. Ao observar como a modularidade se manifesta tanto no software (capacidade de adaptar, adicionar ou remover componentes sem impactar o todo [3]) quanto no hardware (servidores blade que permitem substituições sem interrupção [4]), percebe-se que esta característica transcende uma mera escolha técnica de design. Ela é, na verdade, um imperativo estratégico para qualquer sistema de grande escala que almeja viabilidade a longo prazo, adaptabilidade e alta disponibilidade. A capacidade de modificar ou substituir partes de um sistema sem afetar sua operação contínua contribui diretamente para sua resiliência e agilidade. Isso significa que a modularidade não apenas facilita a manutenção e a expansão eficientes, mas também impacta diretamente a continuidade operacional de um negócio e sua capacidade de inovar. Ao introduzir a modularidade desde o início do curso, os alunos são expostos a uma filosofia arquitetural que será fundamental para a compreensão

dos microsserviços, incentivando-os a pensar nos sistemas de forma holística, onde as decisões em uma camada (software) têm paralelos e impactos em outras (hardware/infraestrutura).

2. Arquiteturas de Software: Monolito vs. Microsserviços

A escolha da arquitetura de software é uma decisão crítica que impacta diretamente a escalabilidade, a manutenibilidade e a agilidade de um sistema corporativo. As duas abordagens predominantes são a arquitetura monolítica e a arquitetura de microsserviços.

2.1. O Monolito: Conceito, Vantagens e Desafios

- **Conceito:** Uma arquitetura monolítica é como um grande edifício onde todos os componentes da aplicação — interface de usuário, lógica de negócio e acesso a dados — residem juntos, compartilhando recursos e dependências em uma única base de código e um único artefato de implantação.[5] A interface de usuário se conecta diretamente a essa aplicação singular, que por sua vez acessa um banco de dados centralizado.[5]
- **Vantagens:** No início de um projeto, a simplicidade é um grande atrativo. Há menos configuração e o gerenciamento é mais fácil.[5] O desenvolvimento inicial é mais direto com uma única base de código.[6] A consistência de dados é mais fácil de gerenciar, especialmente com transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), pois tudo está em um único banco de dados.[5] A depuração e os testes também são simplificados, já que todo o código está em um só lugar, permitindo que testes de ponta a ponta sejam realizados mais rapidamente.[5]
- **Desafios/Desvantagens:** À medida que o aplicativo cresce, a velocidade de desenvolvimento tende a diminuir, tornando-se mais complexo e demorado.[6] A escalabilidade é limitada, pois não é possível dimensionar componentes individuais de forma independente; se um recurso específico precisa escalar, todo o monólito escala junto, aumentando custos e complexidade desnecessariamente.[5] A confiabilidade e a resiliência são baixas, pois um erro em qualquer módulo pode afetar a disponibilidade de todo o aplicativo, criando um

ponto único de falha.[5] Há uma barreira significativa para a adoção de novas tecnologias, já que qualquer alteração na estrutura ou na linguagem afeta todo o aplicativo, tornando as mudanças caras e demoradas.[5] O alto acoplamento significa que pequenas mudanças em uma parte podem impactar outras, dificultando atualizações frequentes e aumentando o risco de introduzir bugs.[5] Finalmente, monólitos podem se tornar ineficazes e difíceis de gerenciar quando crescem muito.[6]

- **Quando Usar Monólito:** Esta arquitetura é geralmente mais adequada para projetos pequenos ou médios, com escopo bem definido e equipes reduzidas, especialmente quando a velocidade inicial de desenvolvimento e implantação é a principal prioridade.[5]

2.2. Microsserviços: Conceito, Vantagens e Desafios

- **Conceito:** Microsserviços representam uma abordagem de arquitetura na qual as aplicações são distribuídas em serviços independentes que se comunicam por meio de APIs (Application Programming Interfaces).[5] Cada serviço possui uma responsabilidade específica e isolada, podendo ter seu próprio banco de dados e ser implantado de forma independente.[5] A interface de usuário, nesse modelo, conecta-se a múltiplos microsserviços independentes.[5]
- **Vantagens:** A escalabilidade é granular e flexível, permitindo dimensionar apenas os serviços que realmente precisam, o que economiza recursos e permite uma resposta rápida a picos de demanda.[5] A resiliência e a alta confiabilidade são notáveis; se um serviço falha, os outros podem continuar funcionando normalmente devido à sua independência, evitando um ponto único de falha.[5] A agilidade e a implementação contínua são promovidas por equipes pequenas e autônomas, permitindo ciclos de lançamento mais rápidos e frequentes, pois cada serviço pode ser desenvolvido, atualizado e implantado de forma independente.[6] Há uma grande flexibilidade tecnológica, pois as equipes têm a liberdade de selecionar as ferramentas (linguagem, framework, banco de dados) mais adequadas para cada serviço.[5] Os microsserviços são altamente sustentáveis e testáveis, facilitando a experimentação de novas funções e a reversão se algo não funcionar, além de ser mais fácil isolar e corrigir falhas e bugs em serviços individuais.[6] A autonomia das equipes e a integração com DevOps são pontos fortes, pois as equipes podem criar e implementar sozinhas, alinhando-se perfeitamente com as práticas de entrega contínua.[6] Por fim, microsserviços não reduzem a complexidade inerente de um sistema grande, mas

a tornam visível e mais gerenciável, separando tarefas em processos menores e independentes.[6]

- **Desafios/Desvantagens:** A complexidade operacional aumenta significativamente ao gerenciar e monitorar múltiplos serviços distribuídos.[5] Os custos de infraestrutura podem ser exponenciais, já que cada novo microsserviço pode ter seu próprio custo para testes, implantação, hospedagem e ferramentas de monitoramento.[6] Há uma sobrecarga organizacional adicional, exigindo um nível extra de comunicação e colaboração entre as equipes para coordenar atualizações e interfaces.[6] A depuração se torna mais desafiadora, com múltiplos logs e processos distribuídos.[6] A falta de padronização e propriedade clara pode ocorrer se não houver governança adequada, levando à proliferação de linguagens e padrões.[6] A comunicação entre serviços distribuídos introduz novos pontos de falha e desafios de latência.[5] Por fim, manter a consistência de dados entre serviços distribuídos, cada um potencialmente com seu próprio banco de dados, é um desafio significativo que exige estratégias específicas.[5]

A transição para microsserviços não é uma panaceia, e a complexidade não é eliminada, mas sim transformada. Enquanto um monólito apresenta sua complexidade em uma massa única e em suas interdependências internas, a complexidade dos microsserviços reside em sua distribuição, comunicação e orquestração. Essa mudança requer uma alteração fundamental na forma como as equipes e a organização operam. A adoção de microsserviços, portanto, não é meramente uma decisão técnica; é uma escolha estratégica que exige alta maturidade organizacional, investimento em ferramentas e uma cultura de colaboração e autonomia, alinhada aos princípios de DevOps. Sem esses pré-requisitos, as desvantagens dos microsserviços podem facilmente superar as vantagens, levando a um aumento da complexidade não intencional.[6] A compreensão de que microsserviços "não são uma solução milagrosa" [6] é crucial, pois a arquitetura "ideal" é aquela que melhor se adapta ao contexto do negócio e às capacidades da equipe, e não apenas a mais avançada tecnologicamente.

2.3. Diferenças Cruciais entre Sistemas Monolíticos e Distribuídos

Para solidificar o entendimento das diferenças e dos *trade-offs* entre as duas arquiteturas, a tabela a seguir oferece uma comparação direta e estruturada, consolidando as informações discutidas.

Critério	Arquitetura Monolítica	Arquitetura de Microsserviços
Conceito	Aplicação única e coesa, com todos os componentes acoplados em uma base de código.[5]	Coleção de serviços independentes, pequenos e autônomos, comunicando-se via APIs.[5]
Escalabilidade	Limitada; escala todo o aplicativo, mesmo que apenas um componente precise.[5]	Granular; escala apenas os serviços necessários, otimizando recursos.[5]
Resiliência/Confiabilidade	Baixa tolerância a falhas; um erro pode derrubar todo o sistema.[5]	Alta tolerância a falhas; falha em um serviço não afeta os outros.[5]
Complexidade Inicial	Baixa; mais fácil de configurar e gerenciar no começo.[5]	Alta; maior complexidade de configuração e gestão de múltiplos serviços.[5]
Complexidade Geral	Alta a longo prazo; torna-se difícil de gerenciar e atualizar à medida que cresce.[5]	Alta e distribuída; exige ferramentas e práticas sofisticadas para monitoramento e orquestração.[5]
Consistência de Dados	Alta; mais fácil gerenciar transações ACID em um único banco de dados.[5]	Baixa (eventual); manter consistência entre bancos de dados distribuídos é um desafio.[5]
Flexibilidade Tecnológica	Baixa; limitada pelas tecnologias já usadas no monólito.[5]	Alta; cada serviço pode usar a tecnologia mais adequada.[5]
Velocidade de Desenvolvimento	Mais lenta em grandes aplicações devido à complexidade e acoplamento.[6]	Mais ágil e permite ciclos de lançamento mais rápidos e frequentes.[6]
Facilidade de Debug	Mais fácil, pois todo o código está em um só lugar.[6]	Mais desafiador, com múltiplos logs e processos distribuídos.[6]

Custos de Infraestrutura	Potencialmente menores inicialmente, mas podem aumentar com escala desnecessária.[5]	Potencialmente exponenciais, cada serviço pode ter seu próprio custo de infraestrutura.[6]
Modelo de Equipe/Organização	Equipes maiores, menos autônomas; conflitos de código mais frequentes.[6]	Equipes pequenas e autônomas; maior agilidade e integração com DevOps.[6]
Deployment	Uma pequena alteração requer a reimplementação de todo o monólito.[6]	Independente; permite implantação rápida e fácil de funções individuais.[6]

3. Estudos de Caso: A Jornada da Migração para Microsserviços

A transição de arquiteturas monolíticas para microsserviços é uma jornada complexa, mas frequentemente necessária, impulsionada por desafios que surgem com o crescimento e a demanda de negócios.

3.1. Motivações e Benefícios da Migração

A migração de um monólito para microsserviços é, em muitos casos, uma resposta estratégica a problemas críticos de escalabilidade, manutenção, agilidade e resiliência que se tornam evidentes à medida que um negócio cresce e sua complexidade aumenta.[5] Sistemas monolíticos, por serem um ponto único de falha, podem causar grandes impactos nos serviços em caso de erros ou quedas, resultando em perdas financeiras e de clientes a cada minuto de inatividade.[7]

Os benefícios tangíveis alcançados após uma migração bem-sucedida incluem uma notável facilidade de manutenção, escalabilidade granular dos serviços, baixo acoplamento e independência entre os componentes. Isso significa que um problema em um serviço específico não afeta o restante da aplicação, garantindo maior resiliência e continuidade operacional.[8]

3.2. Exemplos de Sucesso na Indústria: Netflix e Amazon

Empresas líderes de mercado como Netflix e Amazon são exemplos paradigmáticos da migração bem-sucedida de monólitos para microsserviços, demonstrando as motivações e os desafios inerentes a essa transformação.

- **Netflix:** O caso da Netflix é frequentemente citado como um marco. Em 2008, a empresa sofreu uma queda de seus serviços por quatro dias, causada por corrupção de dados em seu banco de dados monolítico principal. Esse evento crítico foi o catalisador para uma migração massiva para a nuvem da AWS e para uma arquitetura de microsserviços, um processo que só foi considerado finalizado em 2016.[7] Essa experiência ilustra como falhas catastróficas em monólitos podem forçar uma mudança arquitetural estratégica e de longo prazo.
- **Amazon:** A Amazon é reconhecida como pioneira na quebra de seu monólito. Seus arquitetos de software perceberam que a arquitetura monolítica estava impedindo a expansão da empresa. Eles começaram a trabalhar na decomposição do sistema em partes, muito antes de o termo "microsserviços" se tornar amplamente conhecido.[7] Este exemplo destaca uma visão proativa em resposta a gargalos de crescimento, buscando agilidade e capacidade de inovação.

A migração de um monólito para microsserviços é um processo complexo e desafiador. Envolve a quebra do sistema em componentes menores, a garantia de que esses componentes se comuniquem de forma eficiente e segura, e a definição de uma estratégia robusta para gerenciar dados e assegurar a consistência do sistema como um todo.[7] É uma "mudança estrutural na empresa" [7], que vai além da tecnologia, impactando processos e cultura organizacional.

Para auxiliar nesse processo complexo, existem padrões arquiteturais específicos. Exemplos incluem:

- **Gateway:** Um componente responsável por rotear requisições para os serviços corretos e garantir a comunicação segura e eficiente.
- **Circuit Breaker:** Monitora a comunicação entre serviços e a interrompe em caso de falha, evitando que uma falha em um serviço afete todo o sistema.
- **Service Registry:** Armazena informações sobre os serviços disponíveis, permitindo que outros serviços os descubram e se comuniquem com eles.
- **Load Balancer:** Distribui requisições entre os serviços disponíveis, garantindo uma carga equilibrada.

Um padrão particularmente útil para migrações graduais é o *Strangler Application* (Aplicação Estranguladora). Ele permite que um monólito seja migrado para microsserviços de forma segura e controlada, sem comprometer a integridade do sistema durante a transição.[9]

A migração para microsserviços é um imperativo de negócio e uma transformação holística. Os casos da Netflix e da Amazon demonstram que a decisão de migrar foi desencadeada por problemas críticos de negócio, como tempo de inatividade e impedimentos à expansão, que eram diretamente atribuíveis às limitações de suas arquiteturas monolíticas.[7] Isso revela que a mudança para microsserviços não é apenas uma escolha de "modernização" técnica, mas uma resposta direta a necessidades empresariais urgentes. As vantagens técnicas de escalabilidade e resiliência se traduzem diretamente em benefícios empresariais tangíveis, como continuidade do serviço e capacidade de crescimento. A migração é um projeto de alta complexidade e risco, que exige não apenas expertise técnica (como o conhecimento de padrões como Strangler Application), mas também uma profunda compreensão do impacto organizacional e financeiro. É um investimento significativo justificado pelo valor estratégico que desbloqueia, permitindo que as empresas mantenham sua competitividade e capacidade de inovação. Essa perspectiva prepara os futuros arquitetos para pensar além do aspecto técnico, considerando o impacto total de suas decisões no contexto do negócio.

4. Introdução ao Projeto Prático: Plataforma de Eventos Corporativos

O projeto prático semestral será o desenvolvimento de uma "plataforma de eventos corporativos". Este projeto servirá como um estudo de caso prático para aplicar os conceitos de sistemas distribuídos e microsserviços, permitindo que os participantes transformem a teoria em experiência prática.

4.1. Visão Geral do Projeto Semestral: Domínio e Requisitos Gerais

O domínio de "eventos corporativos" é vasto e pode englobar desde feiras de cosméticos que reúnem fabricantes para expor novidades até workshops internos para capacitação de colaboradores.[10] A plataforma a ser desenvolvida deverá

gerenciar o ciclo de vida completo de um evento, desde sua criação e divulgação até a gestão de participantes, pagamentos e coleta de feedback pós-evento.

4.2. Brainstorming de Funcionalidades Essenciais

Na fase inicial do projeto, é crucial que as equipes se concentrem nas funcionalidades nucleares que definem o *core* da plataforma de gestão de eventos. É importante distinguir entre essas funcionalidades essenciais e as tecnologias de aprimoramento que, embora valiosas, podem ser integradas em iterações futuras.

- **Criação de Eventos (para Organizadores):**
  - Definição de detalhes do evento: nome, descrição, data, horário, local, capacidade máxima de participantes, categorias, agenda de palestras/atividades, informações de palestrantes.
  - Configuração de ingressos: tipos (gratuito/pago), preços, lotes.
- **Inscrições de Participantes:**
  - Para participantes: funcionalidade para navegar por eventos disponíveis, visualizar detalhes completos e realizar a inscrição.
  - Para organizadores: gerenciar inscrições, aprovar/rejeitar, gerar listas de presença, funcionalidades de check-in (com potencial para integrar tecnologias como *beacons* no futuro [11]).
- **Pagamento:**
  - Integração com *gateways* de pagamento para processar transações de eventos pagos.
  - Gerenciamento do status de pagamento (pendente, aprovado, cancelado).
  - Emissão de recibos e comprovantes.
- **Feedback:**
  - Mecanismos para coleta de feedback dos participantes após o evento (formulários de avaliação, pesquisas de satisfação).
  - Funcionalidades para organizadores visualizarem e analisarem os resultados do feedback.
- **Outras Funcionalidades (para discussão e futuras iterações – *não são o foco da Semana 1, mas importantes para o panorama geral*):**
  - **Divulgação:** Criação de *hotsites* simples e objetivos para divulgação [11], integração com redes sociais para engajamento [11], envio de notificações *push* para participantes.[11]
  - **Interatividade no Evento:** Totens interativos, uso de Realidade Virtual (RV) e

Realidade Aumentada (RA), drones para filmagem, hologramas para palestrantes remotos.[11]
    - Gestão de usuários (perfis, autenticação e autorização).
    - Geração de relatórios e *dashboards* para análise de eventos.

A prioridade do *core business* sobre o "brilho" tecnológico é um conceito fundamental aqui. Embora o material de pesquisa apresente uma vasta gama de tecnologias avançadas para eventos corporativos (como *beacons*, realidade virtual, drones [11]), é crucial que o foco inicial do projeto esteja nas funcionalidades que constituem a essência de uma

plataforma de gestão de eventos (criação, inscrição, pagamento, feedback). Existe o risco de os alunos se desviarem para as tecnologias mais chamativas e negligenciarem as funcionalidades fundamentais que definem a própria utilidade da plataforma. O objetivo é construir uma *plataforma de gestão*, não apenas uma experiência de evento enriquecida. Essa distinção ensina a importância da priorização de requisitos e da definição de um Produto Mínimo Viável (MVP) em projetos de software. Os alunos aprendem a focar na resolução do problema central do negócio antes de adicionar complexidade com funcionalidades avançadas, uma habilidade crítica para o sucesso de qualquer projeto de engenharia de software.

A tabela a seguir servirá como um guia para o brainstorming inicial das equipes, distinguindo claramente entre as funcionalidades essenciais (core) e as funcionalidades avançadas ou de aprimoramento.

Categoria da Funcionalidade	Funcionalidades Essenciais (Foco Inicial)	Funcionalidades Avançadas / Tecnologias de Aprimoramento (Futuras Iterações)
Gestão de Eventos	Criação e edição de eventos (nome, data, local, descrição, capacidade, agenda, palestrantes)	Geração de relatórios e *dashboards* analíticos

Participantes	Inscrição de participantes, visualização de detalhes do evento, gerenciamento de inscrições, lista de presença, *check-in*	*Beacons* para *check-in* automatizado e *networking* [11], Pulseiras NFC para rastreamento de tráfego [11]		
Financeiro	Processamento de pagamentos (integração com *gateway*), gestão de status de pagamento, emissão de recibos	-		
Comunicação/ Feedback	Coleta de feedback pós-evento (formulários, pesquisas), visualização de resultados	*Hotsites* responsivos para divulgação [11], Notificações	*push* [11], Integração com redes sociais [11]	
Interatividade no Local	-	Totens interativos [11], Robôs	*display* [11], Realidade Virtual/Aumentada [11], Drones [11], Hologramas [11], Decoração	*hi-tech* [11]

4.3. Formação de Equipes e Discussão Inicial do Projeto

Os participantes serão organizados em equipes (sugere-se 3-5 membros, dependendo do tamanho da turma). A primeira discussão em equipe deve focar em:

- Confirmar o tema "plataforma de eventos corporativos" como o projeto a ser

desenvolvido.
- Iniciar o brainstorming e o levantamento de exemplos de funcionalidades, priorizando as essenciais (criar eventos, inscrever-se, pagamento, feedback), utilizando a tabela de funcionalidades como guia.
- Começar a discutir a "visão geral do problema que o sistema irá resolver", que será o foco do entregável da semana.

5. Configuração do Ambiente de Desenvolvimento

Um ambiente de desenvolvimento bem configurado é a base para o trabalho prático eficiente e consistente ao longo do semestre. A padronização via Docker será um elemento chave para minimizar problemas de compatibilidade entre diferentes máquinas.

5.1. Ferramentas Essenciais

As seguintes ferramentas serão utilizadas e são cruciais para o desenvolvimento de sistemas distribuídos e microsserviços:

- **JDK (Java Development Kit) / Spring Boot (Java):** O JDK é o kit de desenvolvimento essencial para aplicações Java. O Spring Boot é um *framework* amplamente utilizado para a construção rápida e eficiente de aplicações *stand-alone* e baseadas em microsserviços, facilitando a configuração e o empacotamento.[12]
- **Python / FastAPI (Python):** Python é uma linguagem versátil, e FastAPI é um *framework* moderno e de alta performance para a construção de APIs web, ideal para microsserviços devido à sua velocidade e facilidade de uso.[13]
- **Node.js / npm (JavaScript):** Node.js permite a execução de JavaScript no lado do servidor, sendo útil para microsserviços baseados em JavaScript ou para ferramentas de *frontend* e automação de *build*. npm (Node Package Manager) é o gerenciador de pacotes padrão.[16]
- **Docker:** Uma ferramenta fundamental para criar ambientes de desenvolvimento consistentes e isolados. Permite empacotar aplicações e suas dependências em "contêineres", garantindo que o software funcione de forma idêntica em qualquer

ambiente. Essencial para microsserviços para padronização e facilidade de implantação.[12]

- **IDEs (Ambientes de Desenvolvimento Integrado):**
  - **IntelliJ IDEA:** Uma IDE robusta e altamente produtiva, especialmente popular e recomendada para desenvolvimento Java.[20]
  - **VS Code (Visual Studio Code):** Um editor de código leve, extensível e versátil, com vasta gama de extensões que o tornam adequado para desenvolvimento em Java, Python, Node.js, e para trabalhar com Docker.[22]

5.2. Guias de Instalação e Configuração (Passo a Passo Simplificado com Referências)

Um guia conciso para a instalação de cada ferramenta é fornecido, com ênfase na utilização de Docker para padronização do ambiente.

- **JDK (Java Development Kit):**
  - **Passo 1:** Acessar o site oficial da Oracle ou OpenJDK (e.g., Adoptium para OpenJDK) e baixar a versão mais recente ou a versão recomendada para o curso (e.g., JDK 17 LTS ou JDK 21). Selecionar o link de download com base no sistema operacional (Windows, macOS, Linux).[26]
  - **Passo 2:** Executar o instalador e seguir as instruções na tela.
  - **Passo 3:** Configurar a variável de ambiente JAVA_HOME para apontar para o diretório de instalação do JDK.[27]
- **Spring Boot:**
  - Spring Boot não é uma "instalação" de software no sentido tradicional, mas sim um conjunto de bibliotecas e um *framework*.
  - **Recomendação:** Utilizar o Spring Initializr (start.spring.io) para gerar projetos base. Selecionar as dependências necessárias, como "Spring Web", e baixar o projeto ZIP. Isso automatiza grande parte da configuração inicial.[12]
- **Python:**
  - **Passo 1:** Acessar o site oficial do Python (www.python.org) e baixar o instalador da versão mais recente ou a versão recomendada.[28]
  - **Passo 2:** Executar o instalador. **Importante:** Durante a instalação, marcar a opção "Add Python to PATH" (ou equivalente) para que o Python e o pip (gerenciador de pacotes) sejam acessíveis via linha de comando.[28]
- **FastAPI:**
  - Após a instalação do Python, usar o pip para instalar o FastAPI e o Uvicorn

(um servidor ASGI necessário para rodar aplicações FastAPI) [15]:

pip install fastapi "uvicorn[standard]"

- Um exemplo básico de código FastAPI para teste pode ser fornecido.[14]

- **Node.js e npm:**
  - **Recomendação:** Para Windows, recomenda-se instalar nvm-windows (Node Version Manager for Windows) para gerenciar múltiplas versões do Node.js de forma fácil. Para Linux/macOS, usar nvm (Node Version Manager).[16]
  - **Passo 1:** Remover quaisquer instalações existentes de Node.js/npm para evitar conflitos.[16]
  - **Passo 2:** Baixar e instalar o nvm para o seu sistema operacional.
  - **Passo 3:** Usar nvm para instalar a versão recomendada do Node.js (que já inclui o npm).[17]
- **Docker Desktop:**
  - **Windows:**
    - **Passo 1:** Garantir que o Windows esteja atualizado (Windows 10/11) e que a virtualização de CPU esteja habilitada na BIOS.[18]
    - **Passo 2:** Habilitar o WSL2 (Windows Subsystem for Linux) e instalar o *kernel* Linux.[18]
    - **Passo 3:** Baixar e instalar o Docker Desktop do site oficial (docs.docker.com/desktop/install/windows-install/).[18]
  - **macOS:** Baixar e instalar o Docker Desktop do site oficial.
  - **Linux:** Baixar e instalar o Docker Desktop ou o Docker Engine CE para a sua distribuição.[19]
  - **Importância do Docker:** O Docker será a ferramenta central para padronizar os ambientes de desenvolvimento e implantação, minimizando problemas de compatibilidade ("funciona na minha máquina"). Exemplos de como aplicações Spring Boot e FastAPI podem ser conteinerizadas usando *Dockerfiles* devem ser demonstrados.[12]

A inclusão do Docker como base para DevOps e para a criação de um ambiente consistente em microsserviços é um aspecto crucial. A ênfase no Docker para diferentes *frameworks* de *backend* (Java e Python) não é acidental; ela aponta para o papel central do Docker na criação de ambientes de desenvolvimento e produção que são *consistentes e isolados*. Essa consistência é fundamental para gerenciar a complexidade inerente aos microsserviços.[6] Ao introduzir e configurar o Docker na Semana 1, o curso está implicitamente familiarizando os participantes com um conceito chave das práticas de DevOps e arquiteturas

cloud-native: a conteinerização. Isso não apenas resolve problemas práticos de compatibilidade de ambiente, mas também prepara os participantes para entenderem como os microsserviços são empacotados, implantados e gerenciados em ambientes distribuídos. Essa abordagem pedagógica transforma a configuração do ambiente de uma tarefa meramente técnica em uma lição sobre a importância da padronização e da infraestrutura como código para o sucesso de projetos de microsserviços, pavimentando o caminho para a compreensão de conceitos mais avançados de orquestração e entrega contínua que serão abordados nas semanas seguintes.

6. Atividade Avaliativa da Semana 1: Alinhando o Entendimento

A primeira atividade avaliativa do semestre, embora aparentemente simples, possui um objetivo pedagógico de grande importância: garantir o alinhamento e a compreensão inicial do projeto.

Descrição do Entregável

Ao final da Semana 1, cada equipe deverá entregar um documento conciso (1-2 páginas) contendo:

- Uma breve descrição do projeto escolhido, confirmando o tema "plataforma de eventos corporativos" e a delimitação inicial do escopo.
- Um parágrafo de visão geral do problema que o sistema proposto irá resolver. Este parágrafo deve articular claramente a necessidade ou o desafio que a plataforma visa endereçar.

Objetivo do Entregável para o Alinhamento com o Professor

O principal objetivo deste entregável é assegurar que todas as equipes tenham compreendido o escopo do projeto prático e estejam alinhadas com as expectativas do curso e do professor antes de avançar para fases de desenvolvimento mais complexas. Isso permite que o professor forneça feedback inicial construtivo sobre a

interpretação do problema e a proposta da solução, corrigindo quaisquer mal-entendidos ou direcionando as equipes para um caminho mais produtivo. Além disso, a atividade estimula a comunicação, a colaboração e o pensamento crítico sobre o problema a ser resolvido desde o início do semestre, uma habilidade fundamental em engenharia de software. Este documento servirá como a base para as próximas etapas do projeto, como o levantamento de requisitos detalhados e o design da arquitetura.

A essencialidade da definição do problema na engenharia de software é destacada por esta atividade. Embora o entregável seja uma "breve descrição do projeto" e um "parágrafo de visão geral do problema", sua simplicidade esconde um pilar de qualquer projeto de engenharia de software bem-sucedido. Ela força os participantes a irem além da ideia superficial e a articularem *qual problema* eles estão realmente tentando resolver e *para quem*. Isso se conecta diretamente com a discussão sobre as características dos sistemas corporativos e o brainstorming de funcionalidades. Ao tornar a definição do problema o primeiro entregável avaliativo, o curso enfatiza que a clareza na compreensão do desafio de negócio é tão, ou mais, importante quanto a proficiência técnica. Muitos projetos falham não por falta de habilidade técnica, mas por uma compreensão inadequada do problema ou desalinhamento com as expectativas dos *stakeholders*. Esta atividade incute nos participantes uma mentalidade de "solucionador de problemas" em vez de apenas "codificador", ensinando-os que a fase de descoberta e alinhamento é crítica para evitar retrabalho dispendioso e garantir que o produto final realmente atenda às necessidades, estabelecendo um padrão profissional para a abordagem de projetos ao longo de suas carreiras.

Conclusão: Próximos Passos e Expectativas para o Semestre

A primeira semana do curso estabeleceu uma base robusta, tanto teórica quanto prática, para a compreensão dos sistemas corporativos e das arquiteturas de software modernas. A familiaridade com as características essenciais de escala, modularidade e manutenção, a distinção clara entre monólitos e microsserviços, e a análise de casos de migração bem-sucedida, como os da Netflix e Amazon, são conhecimentos fundamentais. A introdução ao projeto prático da plataforma de eventos corporativos e a configuração do ambiente de desenvolvimento com ferramentas como JDK/Spring Boot, Python/FastAPI, Node.js/npm, Docker e IDEs são

passos cruciais para o sucesso no restante do curso.

Encoraja-se vivamente que os participantes continuem explorando as ferramentas e conceitos apresentados, praticando a configuração do ambiente e aprofundando a discussão sobre o projeto prático em suas equipes. Os próximos módulos aprofundarão os conceitos de microsserviços, padrões de design, comunicação entre serviços e a construção iterativa da plataforma de eventos corporativos. O semestre promete desafios estimulantes e oportunidades de aprendizado significativas, promovendo uma mentalidade de crescimento e colaboração contínuos.

Referências citadas

1. Sistema para empresas: saiba como ele pode melhorar a sua gestão de processos, acessado em julho 31, 2025, https://blog.1doc.com.br/sistema-para-empresas/
2. Um pouco mais sobre sistemas corporativos - Expert System, acessado em julho 31, 2025, https://blog.expertsystem.com.br/um-pouco-mais-sobre-sistemas-corporativos/
3. Sistema modular: o que é e quais os seus tipos - Digi Office, acessado em julho 31, 2025, https://www.digioffice.com.br/blog/sistema-modular/
4. O que é um servidor modular? - Pure Storage, acessado em julho 31, 2025, https://www.purestorage.com/br/knowledge/what-is-a-modular-server.html
5. Monolito vs. Microsserviços:: como escolher sua arquitetura, acessado em julho 31, 2025, https://www.rocketseat.com.br/blog/artigos/post/monolito-vs-microservices-como-escolher-arquitetura
6. Microsserviços versus arquitetura monolítica | Atlassian, acessado em julho 31, 2025, https://www.atlassian.com/br/microservices/microservices-architecture/microservices-vs-monolith
7. Migrando para microsserviços - Amazon e Netflix | by Matheus de Andrade Lima - Medium, acessado em julho 31, 2025, https://mathlimma.medium.com/microsservi%C3%A7os-um-estudo-de-caso-amazon-e-netflix-3582648540a0
8. MICROSERVIÇOS: um estudo de caso apontando suas potencialidades - ResearchGate, acessado em julho 31, 2025, https://www.researchgate.net/publication/350433415_MICROSERVICOS_um_estudo_de_caso_apontando_suas_potencialidades
9. Migração de Monolito para Microservices com Strangler Application Pattern - YouTube, acessado em julho 31, 2025, https://www.youtube.com/watch?v=tYVKIlshKPk
10. Eventos Corporativos: Conheça os principais tipos e como criar, acessado em julho 31, 2025, https://blog.sympla.com.br/blog-do-produtor/tipos-de-eventos-corporativos-par

a-produzir/
11. Inovação em eventos: 13 tecnologias em alta para ter sucesso, acessado em julho 31, 2025, https://www.copastur.com.br/blog/tecnologia-em-eventos-corporativos-7-tendencias-que-estao-em-alta/
12. Getting Started | Spring Boot with Docker, acessado em julho 31, 2025, https://spring.io/guides/gs/spring-boot-docker/
13. FastAPI in Containers - Docker, acessado em julho 31, 2025, https://fastapi.tiangolo.com/deployment/docker/
14. Primeiros Passos - FastAPI, acessado em julho 31, 2025, https://fastapi.tiangolo.com/pt/tutorial/first-steps/
15. How to Use FastAPI [Detailed Python Guide] - Uptrace, acessado em julho 31, 2025, https://uptrace.dev/blog/python-fastapi
16. Set up Node.js on native Windows - Microsoft Learn, acessado em julho 31, 2025, https://learn.microsoft.com/en-us/windows/dev-environment/javascript/nodejs-on-windows
17. NodeJS e NPM - Instalando e Configurando #01 Hcode Setup - YouTube, acessado em julho 31, 2025, https://m.youtube.com/watch?v=7iSylg2UvU0&t=0s
18. Como instalar o Docker Desktop no Windows 10? Passo a passo. 2023 - YouTube, acessado em julho 31, 2025, https://www.youtube.com/watch?v=kh1gkqCrNx4
19. Instalando o Docker Desktop no Ubuntu. - DEV Community, acessado em julho 31, 2025, https://dev.to/angelobms/instalando-o-docker-desktop-no-ubuntu-241f
20. Como Instalar Intellij Idea 2024 no Windows? Simples e Rápido - YouTube, acessado em julho 31, 2025, https://www.youtube.com/watch?v=7YiYGbSY1pI
21. Como Instalar e Configurar o IntelliJ IDEA | Guia Completo para Java e Kotlin - YouTube, acessado em julho 31, 2025, https://www.youtube.com/watch?v=YKHM_DUOV0k&pp=0gcJCfwAo7VqN5tD
22. VS Code Remote Development, acessado em julho 31, 2025, https://code.visualstudio.com/docs/remote/remote-overview
23. Visual Studio Code for the Web, acessado em julho 31, 2025, https://code.visualstudio.com/docs/setup/vscode-web
24. Documentation for Visual Studio Code, acessado em julho 31, 2025, https://code.visualstudio.com/docs
25. Setting up Visual Studio Code, acessado em julho 31, 2025, https://code.visualstudio.com/docs/setup/setup-overview
26. Como Instalar o Kit de Desenvolvimento de Software Java (JDK) - wikiHow, acessado em julho 31, 2025, https://pt.wikihow.com/Instalar-o-Kit-de-Desenvolvimento-de-Software-Java-(JDK)
27. Instalar o JDK (Java Development Kit - Kit de Desenvolvimento Java), acessado em julho 31, 2025, https://ftpdocs.broadcom.com/cadocs/0/CA%20IT%20Asset%20Manager%2012%209-PTB/Bookshelf_Files/HTML/Implementation/2156423.html
28. Como Instalar Python: Guia Passo a Passo para Iniciantes - Awari, acessado em julho 31, 2025,

https://awari.com.br/como-instalar-python-guia-passo-a-passo-para-iniciantes/
29. Instalação do Python e nosso primeiro Olá Mundo | Blog da TreinaWeb, acessado em julho 31, 2025, https://www.treinaweb.com.br/blog/instalacao-do-python-e-nosso-primeiro-ola-mundo