GUIDE CAPSTONE - DS 3

Table of Contents

Projeto Capstone: Introdução

Este guia servirá como um recurso central para entender os requisitos, a arquitetura e os objetivos de cada fase do projeto.

Convenções do Projeto

Para garantir a consistência e a clareza em todo o projeto, as seguintes convenções serão rigorosamente seguidas:

- **Idioma:** Todo o projeto, incluindo documentação (wikis, especificações, etc.), código-fonte, comentários, mensagens de commit será desenvolvido e mantido **inteiramente em inglês**.
- **Estilo de Código:** O estilo de código seguirá as melhores práticas e padrões estabelecidos para cada linguagem e framework utilizados. Detalhes específicos serão definidos e comunicados pelas equipes.
- **Documentação:** A documentação será clara, concisa e voltada para desenvolvedores juniores, focando no *porquê* e *como* das implementações.

Definição do Produto

O Projeto Capstone é baseado no desenvolvimento de uma Solução de Ideação Colaborativa (CIS), também conhecida como "Crowdsourced Ideation Solution". Esta plataforma de criatividade coletiva permite que os utilizadores proponham e votem em ideias.

As principais características incluem:

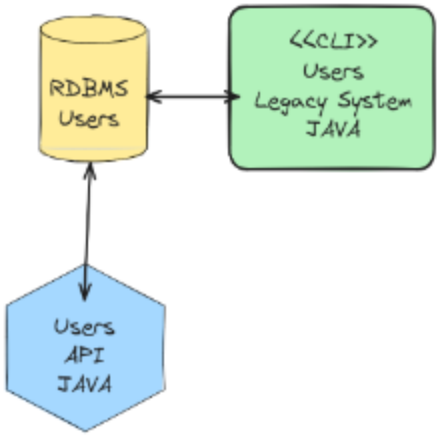
- **Organização por Tópico:** As ideias são organizadas por tópicos, que podem ser arbitrários (ex: "Hábitos de estudo recomendados", "Reduzir o aquecimento global").
- **Proposição e Votação de Ideias:** Os usuários podem propor ideias inovadoras para esses tópicos e votar nas melhores propostas.

Fases do Projeto Capstone

O Projeto Capstone será desenvolvido em três fases principais, cada uma com seus próprios objetivos e especificações.

Fase 1: Suporte a Sistemas Legados

Esta fase foca na integração com sistemas existentes e na criação de uma API de usuário moderna.



fase 1

Objetivos:

- Garantir compatibilidade e integração com um sistema "Legacy" (CLI baseado em Java) e um banco de dados MySQL existente que armazena dados do usuário (identificador, login, nome e senha).
- Utilizar os dados do usuário do sistema "Legacy" e garantir a consistência dos dados (exclusão, criação, atualizações).
- Desenvolver uma solução que coexista com o sistema "Legacy", garantindo a operabilidade de ambos.

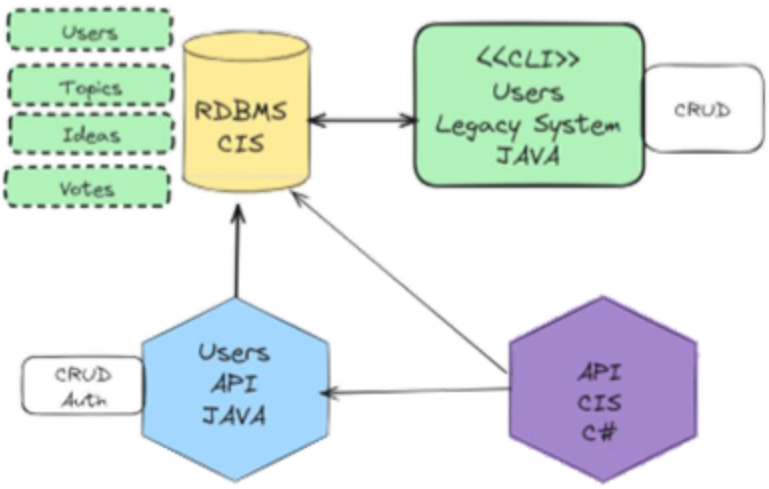
Especificações:

- **API de Usuário Moderna:** Capacidade de realizar operações CRUD (Create, Read, Update, Delete) nos dados do usuário.
- **Consistência de Dados:** A API do usuário deve ser consistente com o sistema legado em termos de dados.

- **Tecnologia:** A API deve ser implementada em Java.
- **API Cliente:** Uma API cliente simples para operar e validar a API de usuário, simulando o CRUD de "n" usuários.
- **Segurança:** A API requer um nível de segurança (ex: acesso autorizado com login/senha).
- **Definição Prévia:** A API deve ser definida antes do início da implementação.

Fase 2: Implementação da API CIS

Esta fase visa implementar a funcionalidade do CIS e integrá-la com a API de usuário desenvolvida na Fase 1.



Fase 2

Objetivos:

- Implementar a funcionalidade do CIS e integrá-la ao sistema legado através da API de usuário da Fase 1.
- Desenvolver uma solução baseada nos artefatos da Fase 1, seguindo um SDLC ágil e escolhendo a tecnologia de software adequada.

Especificações:

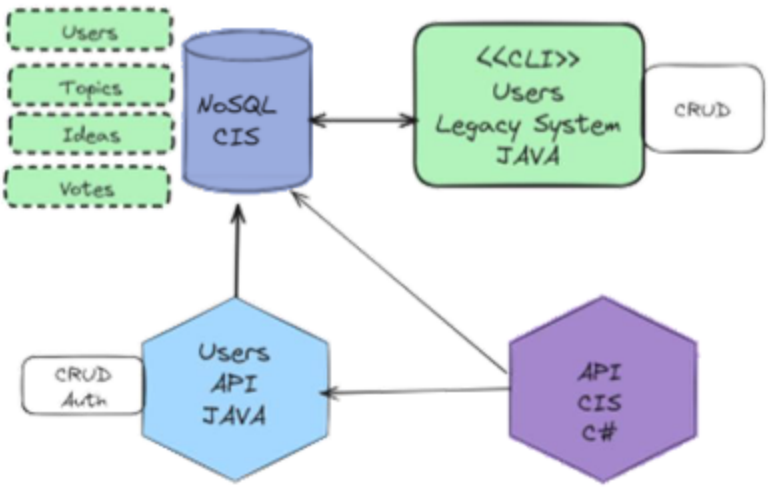
- **Gerenciamento de Tópicos e Ideias:** Estender o banco de dados existente para incluir

tabelas para tópicos e ideias.

- **API CIS Moderna:** Lidar com operações CRUD para criação de tópicos, ideias e votação/cancelamento de votação de ideias.
- **Autenticação:** A autenticação do usuário para usar esta API deve ser feita através da API de Usuário.
- **API Cliente CIS:** Uma API cliente simples para operar e validar a API CIS, simulando aleatoriamente "n" usuários com tópicos, ideias, votação/cancelamento de votação.
- **Definição Prévia:** A API deve ser definida antes do início da implementação.

Fase 3: Mudanças na Infraestrutura

Esta fase aborda a migração do banco de dados e a adaptação das APIs para a nova infraestrutura.



Fase 3

Objetivos:

- Substituir o RDBMS existente por um banco de dados NoSQL (MongoDB).
- Desenvolver uma solução baseada nos artefatos da Fase 2, seguindo um SDLC ágil e escolhendo as tecnologias de software adequadas.

Especificações:

- **Migração de Banco de Dados:** Substituir o banco de dados existente pelo MongoDB.
- **Compatibilidade de APIs:** Adaptar as APIs para serem compatíveis com o novo banco de dados.
- **Funcionalidade Contínua:** O sistema deve continuar a funcionar como antes após a alteração do banco de dados.
- **Atualização da CLI:** O aplicativo de linha de comando (CLI) será atualizado externamente.
- **Migração de Dados:** Migrar os dados existentes para o novo banco de dados com precisão.
- **Manutenção de Desempenho:** Manter o desempenho adequado do sistema após a movimentação.

Organização do Projeto

A implementação das fases do projeto será realizada por alunos organizados em grupos, com o tutor responsável pela organização das equipes.

Abordagem SDLC:

- Cada grupo aplicará um SDLC moderno (com cerimônias SCRUM) para implementar cada fase como uma equipe.

Papéis do Tutor:

O tutor desempenhará um dos seguintes papéis para cada grupo:

- **Product Owner:** Orientação sobre os objetivos de cada componente do sistema.
- **Arquiteto:** Responsável pelo projeto, aspectos técnicos e estrutura geral da solução.
- **Líder Técnico:** Auxiliar nas dúvidas técnicas e orientar as equipes.

Recomendações para o Scrum Master:

Cada equipe deve ter um Scrum Master designado para auxiliar nas cerimônias, seguindo estas recomendações:

- O projeto final é a atividade principal do curso e tem o maior impacto na nota.
- Semanalmente, o professor explicará a parte conceitual/teórica do curso.
- Os alunos devem reforçar os conceitos com atividades práticas relacionadas ao projeto final.

Avaliação:

O desenvolvimento de cada componente da solução será avaliado em duas dimensões:

- **Nível da Equipe:** Conquistas e progresso da equipe (entregas, esgotamento do backlog, documentação, conclusão do planejamento).
- **Nível Individual:** Conquistas e progresso de cada membro (conclusão de tarefas, qualidade do código).

Cerimônias e Acompanhamento:

- **Sessão de Planejamento:** Semanalmente, uma sessão de planejamento é realizada por cada equipe.
- **Reunião Ágil Diária:** Toda semana, para cada laboratório, cada equipe realiza sua reunião ágil diária após o sprint.
- **Orientações do Tutor:** O tutor pode ajudar com orientações sobre a forma correta de realizar as cerimônias.
- **Cerimônias Curtas:** Devido ao grande número de grupos, o tutor agendará cerimônias curtas para comparecer sequencialmente.
- **Verificação do Tutor:** Semanalmente, o tutor verificará a execução das cerimônias e os resultados (artefatos SDLC, código, wikis, etc.). Se houver problemas de qualidade ou compreensão, o tutor poderá fornecer exercícios laboratoriais adicionais.
- **Avaliação na Plataforma Canvas:** Semanalmente, tanto a equipe quanto os indivíduos serão avaliados na plataforma Canvas.

Definição do Produto

O projeto Capstone é baseado no desenvolvimento de uma **Solução de Ideação Colaborativa (CIS)**, também conhecida como *"Crowdsourced Ideation Solution"*.

Esta solução é uma plataforma de criatividade coletiva, onde os utilizadores podem propor ideias e também votar nas melhores.

Funcionamento

1. **Tópicos**: As ideias são organizadas por tópico. Os usuários podem propor tópicos arbitrários (por exemplo, "Hábitos de estudo recomendados", "Reduzir o aquecimento global", etc.).
2. **Ideias e Votos**: Os usuários podem então propor ideias inovadoras para esses tópicos e também votar na melhor ideia proposta. O sistema permite votos positivos e negativos para classificar as ideias.

O objetivo é criar um ambiente colaborativo para gerar e refinar ideias de forma coletiva.

Conceitos-Chave

Esta seção aborda os principais conceitos teóricos e práticos que fundamentam o Projeto Capstone.

SDLC (Ciclo de Vida de Desenvolvimento de Software)

O SDLC é um processo estruturado que define as fases necessárias para construir um sistema de software, desde a concepção até a manutenção.

- **Fases Comuns**: Planejamento, Análise de Requisitos, Design, Desenvolvimento, Testes, Implantação, Manutenção.
- **Modelos**: Waterfall, Agile, Spiral.

Metodologia Ágil

Uma abordagem iterativa e incremental para o desenvolvimento de software, focada na colaboração, feedback contínuo e entrega de valor em ciclos curtos (sprints).

- **Princípios**: Indivíduos e interações mais que processos e ferramentas; software em funcionamento mais que documentação abrangente.

Scrum

Um framework ágil para gerenciar o desenvolvimento de produtos complexos.

- **Artefatos**: Product Backlog, Sprint Backlog.
- **Cerimônias**: Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective.

Padrões Arquitetônicos

Soluções reutilizáveis para problemas comuns de design de software.

- **Exemplos**: MVC (Model-View-Controller), MVP (Model-View-Presenter), MVVM

(Model-View-ViewModel), Arquitetura Hexagonal, Onion Architecture.

TDD (Desenvolvimento Guiado por Testes)

Uma prática de desenvolvimento onde os testes são escritos antes do código da funcionalidade.

- **Ciclo**: Red (escrever um teste que falha) -> Green (escrever o código mínimo para o teste passar) -> Refactor (melhorar o código sem alterar o comportamento).

CI/CD (Integração e Entrega Contínua)

Práticas para automatizar a integração, teste e entrega de código.

- **CI (Integração Contínua)**: Mesclar alterações de código em um repositório central frequentemente.
- **CD (Entrega/Implantação Contínua)**: Automatizar a liberação do software para um ambiente de produção ou pré-produção.

API REST

Uma API (Interface de Programação de Aplicações) REST (Representational State Transfer) é um estilo arquitetural para sistemas distribuídos que define um conjunto de restrições para a criação de serviços web.

- **Princípios Chave**: Stateless (sem estado), Client-Server, Cacheable, Layered System, Uniform Interface.
- **Recursos**: Utiliza métodos HTTP (GET, POST, PUT, DELETE) para manipular recursos identificados por URIs.

APIs em Java e C#

Ambas as linguagens, Java e C#, são amplamente utilizadas para o desenvolvimento de APIs robustas e escaláveis.

- **Java**: Com frameworks como Spring Boot, permite a criação de APIs RESTful de alto desempenho, com forte tipagem e um vasto ecossistema de bibliotecas.
- **C#**: Com o .NET (especialmente ASP.NET Core), oferece um ambiente produtivo para construir APIs web, com recursos como LINQ, async/await e integração nativa com serviços Microsoft Azure.

Docker

Docker é uma plataforma de código aberto que permite automatizar a implantação, escala e gerenciamento de aplicações usando contêineres.

- **Contêineres**: Pacotes leves, portáteis e autossuficientes que incluem tudo o que uma aplicação precisa para rodar (código, runtime, bibliotecas, configurações).
- **Benefícios**: Consistência entre ambientes, isolamento, escalabilidade e portabilidade.

Links Úteis

- O que é SDLC? (https://www.atlassian.com/continuous-delivery/software-development-life-cycle)
- Manifesto Ágil (https://agilemanifesto.org/iso/ptbr/manifesto.html)
- Guia do Scrum (https://scrumguides.org/scrum-guide.html)
- Padrões de Arquitetura (Microsoft) (https://learn.microsoft.com/pt-br/azure/architecture/patterns/)
- O que é REST? (https://restfulapi.net/)
- Introdução ao Spring Boot (https://spring.io/guides/gs/spring-boot/)
- Introdução ao ASP.NET Core (https://learn.microsoft.com/en-us/aspnet/core/get-started?view=aspnetcore-9.0)
- O que é Docker? (https://www.docker.com/what-docker)

Apresentação de Projetos

A apresentação de projetos é um momento crucial para demonstrar o valor e a funcionalidade do trabalho desenvolvido. Ao longo do ciclo de vida do projeto, haverá diferentes oportunidades para apresentar o progresso e os resultados, culminando em uma apresentação final abrangente.

Artefatos do SDLC que Podem Ser Incluídos nas Demonstrações

Para enriquecer a demonstração e fornecer contexto adicional, diversos artefatos do Ciclo de Vida do Desenvolvimento de Software (SDLC) podem ser apresentados:

- **Roadmap:** Uma representação visual do plano estratégico do produto, mostrando a visão geral, metas e etapas futuras.
- **Situação das Histórias de Usuário:** Apresenta o progresso e o status das histórias de usuário, mostrando o que foi concluído e o que está em andamento.
- **Casos de Testes e Resultados:** Demonstra a qualidade do software através dos resultados dos testes, evidenciando que as funcionalidades estão operando conforme o esperado.
- **Diagramas Técnicos:** Diagramas de arquitetura, fluxo de dados ou outros diagramas técnicos que ajudam a explicar a estrutura interna e o funcionamento do sistema.
- **Detalhes do Processo de Desenvolvimento:** Informações sobre a metodologia utilizada, como Scrum ou Kanban, e como o processo contribuiu para o resultado final.

Apresentações por Fase

As apresentações do projeto serão realizadas em diferentes etapas para os **tutores**, com focos e requisitos de idioma distintos:

Apresentações de Fases Intermediárias

- **Foco:** Demonstração do progresso, desafios superados e entregas específicas de cada fase (e.g., Fase 1: Suporte ao Legado, Fase 2: API CIS).

- **Idioma:** Estas apresentações serão realizadas em **Português**.
- **Objetivo:** Validar o trabalho realizado até o momento, obter feedback e alinhar as expectativas para as próximas etapas.

Apresentação da Fase Final

- **Foco:** Uma demonstração abrangente da solução completa, integrando todas as fases do projeto. Deve apresentar o valor total da solução, os resultados alcançados e as perspectivas futuras.
- **Idioma:** Esta apresentação **deve ser realizada em Inglês**.

Visão Geral da Arquitetura

Este documento fornece uma visão geral de alto nível da arquitetura do sistema, seus principais componentes e como eles interagem. A arquitetura foi projetada para ser modular, escalável e evolutiva, acompanhando as fases de desenvolvimento da **Solução de Ideação Crowdsourced (CIS)**.

Estilo Arquitetônico

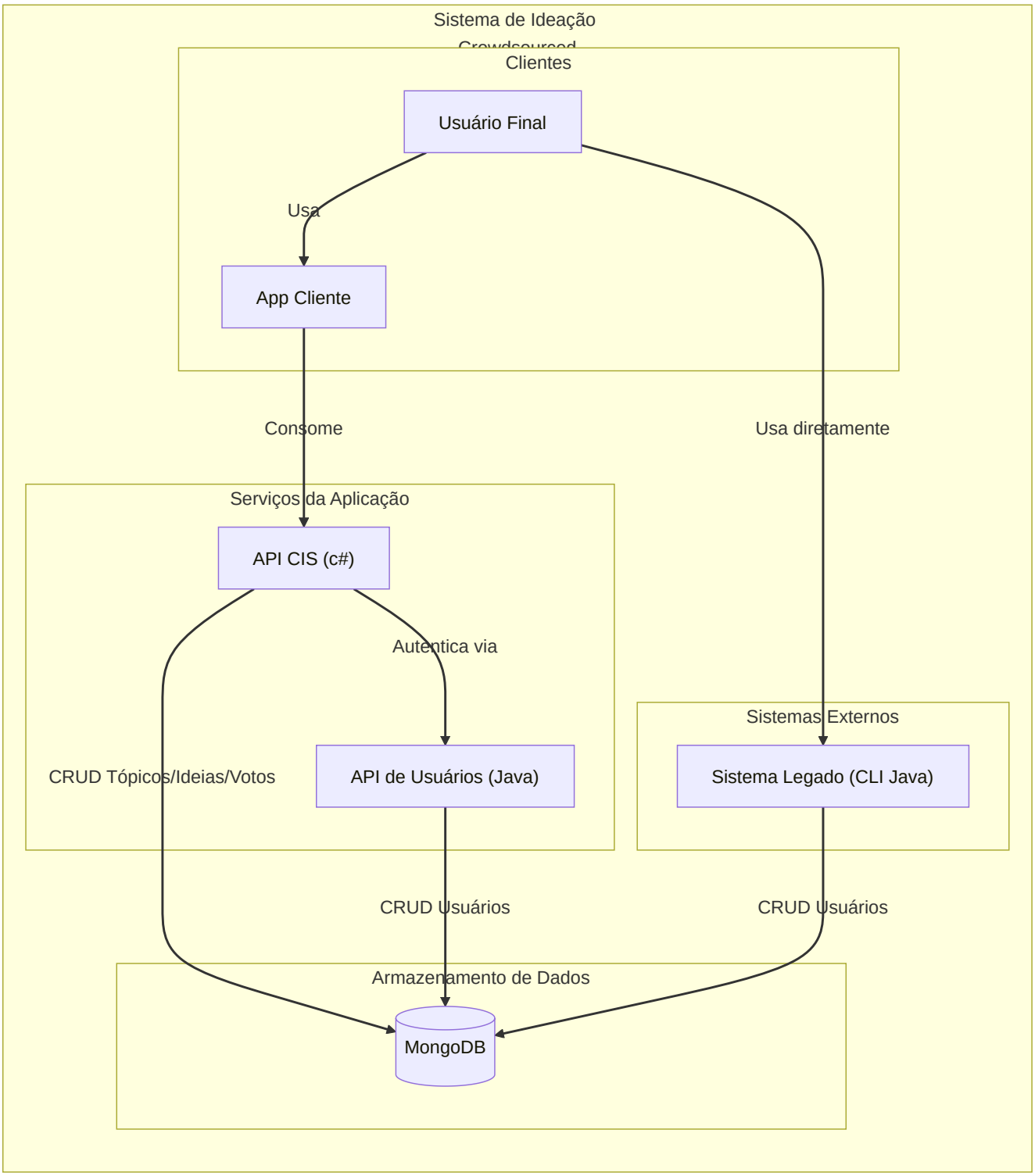
O sistema adota uma **Arquitetura de Microsserviços (MSA)**. Essa abordagem foi escolhida para permitir o desenvolvimento, implantação e escalabilidade independentes de diferentes partes do sistema. Cada microsserviço é responsável por uma capacidade de negócio específica, promovendo uma clara **Separação de Preocupações (SoC)**.

Os principais serviços são:

- **Serviço de Usuários**: Responsável pelo gerenciamento de usuários, incluindo operações CRUD e autenticação.
- **Serviço de Ideação (CIS)**: Responsável pelos recursos centrais da plataforma, como o gerenciamento de tópicos, ideias e votos.

Diagrama da Arquitetura

O diagrama abaixo ilustra a arquitetura final planejada para o sistema, com a utilização de um banco de dados NoSQL.


Sistema de Ideação
Clientes
Usuário Final
Usa
App Cliente
Consome
Usa diretamente
Serviços da Aplicação
API CIS (c#)
Autentica via
CRUD Tópicos/Ideias/Votos
API de Usuários (Java)
Sistemas Externos
Sistema Legado (CLI Java)
CRUD Usuários
CRUD Usuários
Armazenamento de Dados
MongoDB

Descrição dos Componentes

- **Sistema Legado (CLI)**: Uma aplicação de linha de comando (CLI) existente, baseada em Java, que permite o gerenciamento de usuários. O novo sistema deve coexistir e manter a consistência dos dados com este componente.
- **API de Usuários (Java)**: Um microsserviço RESTful desenvolvido em Java. Ele expõe operações CRUD para os dados dos usuários e serve como o serviço de autenticação para outros componentes do sistema.
- **API CIS (c#)**: Um microsserviço RESTful desenvolvido em c#. É o coração da plataforma de ideação, gerenciando tópicos, ideias e votos. Ele depende da API de Usuários para autenticar as requisições.
- **Banco de Dados**: O sistema passa por uma evolução em sua camada de persistência.
  - **Fases Iniciais**: Utiliza um banco de dados relacional **MySQL** para armazenar os dados dos usuários, tópicos e ideias.
  - **Fase Final**: Migra para um banco de dados **NoSQL (MongoDB)** para maior flexibilidade e escalabilidade, substituindo completamente o MySQL.
- **App Cliente**: Uma aplicação cliente simples (simulador) que interage com as APIs para validar a funcionalidade do sistema de ponta a ponta.

Evolução da Arquitetura

A arquitetura é implementada de forma incremental em três fases:

- **Fase 1: Suporte ao Sistema Legado**: O foco é criar a **API de Usuários** em Java, que opera sobre o banco de dados **MySQL** existente, garantindo a consistência com o **Sistema Legado (CLI)**.

- **Fase 2: Implementação da API CIS**: A **API CIS** é desenvolvida em c#, introduzindo as funcionalidades de ideação (tópicos, ideias, votos). Esta API utiliza a API de Usuários para autenticação e também persiste seus dados no **MySQL**.
- **Fase 3: Mudanças na Infraestrutura**: O sistema é migrado do MySQL para o **MongoDB**. Ambas as APIs (`Usuários` e `CIS`) são adaptadas para operar com o novo banco de dados NoSQL, e os dados existentes são migrados.

Padrões e Modelos de Visualização

- **Padrões Arquitetônicos**: Para promover a modularidade e a testabilidade, os microsserviços são projetados seguindo princípios de arquiteturas limpas, como a **Arquitetura Hexagonal (Portas e Adaptadores)**. Isso isola a lógica de negócio central das dependências externas (frameworks de UI, bancos de dados, etc.), o que é especialmente útil para a migração de banco de dados planejada.
- **Modelo de Visualização**: A documentação da arquitetura utiliza o **Modelo C4** (Contexto, Contêineres, Componentes e Código) para descrever o sistema em diferentes níveis de abstração. Isso fornece uma compreensão clara para diferentes públicos, desde uma visão geral de alto nível até os detalhes de implementação para os desenvolvedores.

Processos de Desenvolvimento

Os 4 P's no Planejamento de Projetos

Para planejar um projeto de software, pensamos em quatro coisas principais, os "4 P's": Produto, Processo, Projeto e Pessoas. Entender cada um ajuda muito no sucesso do projeto.

Produto (O Quê?)

O "Produto" é o software que estamos criando. Precisamos saber:

- O que ele precisa fazer (requisitos e características).
- Quais são os objetivos (funcionalidade, desempenho, segurança, facilidade de uso).
- O que o cliente e os usuários realmente precisam.

Processo (Como?)

O "Processo" é como vamos construir o software. Isso inclui:

- Escolher a melhor forma de trabalhar (como Scrum ou Kanban).
- Planejar as etapas, dividir tarefas, lidar com mudanças e garantir a qualidade.
- Fazer o trabalho de forma organizada e eficiente.

Projeto (Quando?)

O "Projeto" é sobre como vamos gerenciar o trabalho para entregar o software. Isso envolve:

- Definir prazos, objetivos e quanto dinheiro e recursos vamos usar.
- Gerenciar riscos e conversar bem com todos os envolvidos.
- Garantir que o projeto avance e atinja seus objetivos.

Pessoas (Quem?)

As "Pessoas" são todos que participam do projeto: a equipe de desenvolvimento, os interessados e os usuários. É importante:

- Saber quem faz o quê na equipe.
- Entender as habilidades e expectativas de todos.
- Distribuir as tarefas e responsabilidades de forma justa.

Ciclo de Vida do Desenvolvimento de Software (SDLC)

O SDLC (Software Development Life Cycle) é como um roteiro que mostra todas as fases de um projeto de software, desde a ideia inicial até a manutenção.

Fases Comuns do SDLC:

Um projeto de software geralmente passa por estas fases:

- **Planejamento:** Definimos o que o projeto vai ser, seus objetivos e os recursos necessários. É a base de tudo.
- **Análise:** Entendemos a fundo o que o sistema precisa fazer, priorizando as funcionalidades e as necessidades dos usuários.
- **Design (Projeto):** Criamos a estrutura e o desenho do software, detalhando como ele vai funcionar e quais serão suas principais partes.
- **Desenvolvimento:** É a hora de escrever o código e construir o software, seguindo o que foi planejado.
- **Testes:** Verificamos se o software funciona como esperado, procurando por erros e garantindo que ele atenda a todos os requisitos.
- **Implantação (Deployment):** O software é entregue aos usuários para que eles possam começar a usá-lo.
- **Manutenção:** Continuamos dando suporte ao software, corrigindo problemas e fazendo melhorias ao longo do tempo.

Modelos de SDLC:

Existem diferentes formas de seguir o SDLC. Para este projeto, vamos usar **Metodologias Ágeis**, que são flexíveis e focam em entregar valor rapidamente. Os principais exemplos que usaremos são Scrum e Kanban.

Metodologias Ágeis

Metodologias ágeis são muito usadas hoje em dia porque se adaptam bem a mudanças e entregam resultados rápidos. Elas tornam o gerenciamento de projetos mais flexível.

- **Scrum:** É uma forma de trabalhar em ciclos curtos, chamados "Sprints". O Scrum ajuda a equipe a colaborar e entregar valor constantemente. Ele tem papéis definidos (Product Owner, Scrum Master, Time de Desenvolvimento) e reuniões específicas (Planejamento da Sprint, Daily Scrum, Revisão da Sprint, Retrospectiva da Sprint).
- **Kanban:** É um método visual para gerenciar o fluxo de trabalho. Usamos um quadro (Kanban Board) para ver as etapas do trabalho e limitar o que está sendo feito ao mesmo tempo. Isso ajuda a equipe a ser mais eficiente e a entregar as coisas de forma contínua.

Como Lidamos com o Backlog

O backlog é uma lista organizada e priorizada de tudo o que precisa ser feito no projeto. É a nossa lista oficial de tarefas.

- **Backlog do Produto:** Contém todas as ideias, funcionalidades, melhorias e correções que o produto pode precisar. O Product Owner decide a prioridade de cada item.
- **Backlog da Sprint:** É uma parte do Backlog do Produto que a equipe se compromete a entregar em uma Sprint específica. No planejamento da Sprint, a equipe escolhe os itens que consegue fazer e os divide em tarefas menores.

Estimativas

Estimativa é prever quanto esforço será necessário para completar uma tarefa. É importante para planejar os recursos.

- **Estimativas de Alto Nível:** Dão uma ideia geral do esforço total do projeto. Podemos usar "tamanhos de camisetas" (XS, S, M, L, XL) para ter uma noção rápida da complexidade. Essas estimativas são um guia inicial e são ajustadas conforme entendemos melhor as tarefas.
- **Estimativas Detalhadas (Story Points):** Atribuímos valores aos itens do backlog (geralmente User Stories) para representar o esforço, a complexidade e o risco. Usamos a técnica de **Scrum Poker** para que a equipe chegue a um consenso. Cada membro escolhe uma carta que representa sua percepção de esforço, e as estimativas são reveladas e discutidas para chegar a um acordo.

Acompanhando o Progresso

É muito importante acompanhar o progresso para garantir que o projeto esteja no caminho certo e para identificar problemas cedo.

- **Roadmap:** É um plano visual e estratégico do produto. Ele mostra a visão geral, os objetivos e as principais etapas de desenvolvimento. Ajuda a ver as dependências e a priorizar o que é mais importante.
- **Burn Down Chart:** É um gráfico que mostra o trabalho que ainda falta ser feito em uma Sprint ou projeto. Ele ajuda a equipe a ver se está no ritmo certo para terminar o trabalho planejado.
- **Burn Up Chart:** É um gráfico que mostra o trabalho total que já foi concluído. Ele ajuda a ver o progopo acumulado e o escopo total do projeto.

Engenharia de Requisitos

A Engenharia de Requisitos é o processo de definir, documentar e manter os requisitos de um sistema de software. É uma fase crucial para garantir que o produto final atenda às necessidades dos usuários e stakeholders.

Requisitos do Projeto Capstone

O projeto Capstone é dividido em fases, cada uma com requisitos específicos que guiam o desenvolvimento da solução.

Fase 1: Suporte de Sistemas "Legacy"

Esta fase foca na integração com um sistema existente e um banco de dados de usuários.

- **Requisitos Funcionais:**
  - O CIS (Sistema de Ideação Colaborativa) deve se integrar e ser compatível com um sistema "Legacy" (CLI baseado em Java) e um banco de dados MySQL existente que armazena dados do usuário (identificador, login único, nome e senha).
  - A solução CIS deve usar os dados de usuários do sistema "Legacy" e garantir a consistência desses dados ao realizar operações de exclusão, criação e atualização.
  - Deve ser possível realizar operações CRUD (Criar, Ler, Atualizar, Deletar) nos dados do usuário através de uma API de usuário moderna.
  - A API do usuário deve ser consistente com o sistema legado em termos de dados.
  - Deve haver uma API cliente simples para operar e validar a API de usuário, simulando operações CRUD para "n" usuários.
- **Requisitos Não-Funcionais:**
  - O sistema "Legacy" e o CIS devem coexistir e ser operáveis simultaneamente.
  - A API do usuário deve ser implementada em JAVA por motivos técnicos.
  - A API requer um nível de segurança (ex: acesso autorizado com login/senha).

- A API deve ser definida antes de iniciar a implementação.
- O desenvolvimento deve seguir um SDLC ágil, com uma estrutura de software adequada e detalhada.

Fase 2: Implementação do CIS API

Esta fase expande a funcionalidade do CIS e sua integração.

- **Requisitos Funcionais:**
  - O sistema deve implementar a funcionalidade do CIS e se integrar ao sistema legado usando a API de usuário desenvolvida na Fase 1.
  - O banco de dados existente deve ser estendido para incluir tabelas para tópicos e ideias dos usuários.
  - Deve haver uma API CIS moderna capaz de lidar com operações CRUD para tópicos, ideias, e votação/cancelamento de votação de ideias.
  - Deve haver uma API cliente simples para operar e validar a API CIS, simulando "n" usuários com tópicos, ideias e votações.
- **Requisitos Não-Funcionais:**
  - Para usar a API CIS, é necessário autenticar o usuário através da API de usuário.
  - A API CIS deve ser definida antes de iniciar a implementação.
  - O desenvolvimento deve seguir um SDLC ágil, com uma tecnologia de software adequada e detalhada.

Fase 3: Mudanças na Infraestrutura

Esta fase envolve uma mudança significativa na tecnologia do banco de dados.

- **Requisitos Funcionais:**
  - Os dados existentes devem ser migrados com precisão para o novo banco de dados NoSQL.

- O sistema deve continuar funcionando como antes após a mudança do banco de dados.

- **Requisitos Não-Funcionais:**
  - As APIs existentes devem ser adaptadas para serem compatíveis com o novo banco de dados.
  - O aplicativo de linha de comando (CLI) será atualizado externamente.
  - O desempenho adequado do sistema deve ser mantido após a migração.
  - O desenvolvimento deve seguir um SDLC ágil, com uma tecnologia de software adequada e detalhada.

Épicas

Épicas do Projeto Capstone

As fases do projeto Capstone podem ser consideradas as épicas principais, pois representam grandes blocos de trabalho que serão detalhados em funcionalidades e histórias de usuário ao longo do desenvolvimento:

- **Fase 1: Suporte de Sistemas "Legacy"**
  - **Objetivo:** Integrar o novo sistema com o sistema legado existente e seu banco de dados de usuários, garantindo a compatibilidade e a consistência dos dados.
  - **Valor:** Permite que o novo sistema utilize e gerencie os dados de usuários já existentes, evitando a duplicação e facilitando a transição.
- **Fase 2: Implementação do CIS API**
  - **Objetivo:** Desenvolver a API central do Sistema de Ideação Colaborativa (CIS), incluindo o gerenciamento de tópicos, ideias e votações, e sua integração com a API de usuários da Fase 1.
  - **Valor:** Habilita as funcionalidades principais da plataforma de ideação, permitindo que os usuários criem, organizem e interajam com ideias.
- **Fase 3: Mudanças na Infraestrutura**
  - **Objetivo:** Migrar o banco de dados de usuários de RDBMS para NoSQL (MongoDB) e adaptar todas as APIs e componentes para funcionar com a nova infraestrutura.
  - **Valor:** Moderniza a infraestrutura do sistema, potencialmente melhorando o desempenho, a escalabilidade e a flexibilidade para futuras expansões.

Épicas são essenciais para o planejamento estratégico e para a organização de grandes iniciativas em projetos ágeis, permitindo que as equipes visualizem o panorama geral enquanto trabalham em partes menores e entregáveis.

User Stories

Uma História de Usuário (User Story) é uma descrição concisa de uma funcionalidade do software do ponto de vista do usuário final. Ela descreve o que o usuário quer alcançar e por que, focando no valor que a funcionalidade trará. As User Stories são a base para o planejamento e desenvolvimento em metodologias ágeis, promovendo a colaboração e o entendimento compartilhado entre a equipe e os stakeholders.

Histórias de Usuário do Projeto Capstone

As histórias de usuário a seguir detalham as funcionalidades necessárias para alcançar os objetivos de cada Épica do projeto Capstone, organizadas por fase:

Fase 1: Gerenciamento de Usuários (Sistema Legacy)

Esta seção detalha as histórias de usuário que compõem a Épica "Suporte de Sistemas 'Legacy'", focando na integração e gerenciamento de usuários do sistema existente.

- Como um **administrador do sistema**, eu quero **criar um novo usuário no sistema**, para que **possa gerenciar o acesso de novas pessoas**.
- Como um **administrador do sistema**, eu quero **visualizar os detalhes de um usuário existente**, para que **possa verificar suas informações**.
- Como um **administrador do sistema**, eu quero **atualizar as informações de um usuário existente**, para que **possa corrigir ou modificar seus dados**.
- Como um **administrador do sistema**, eu quero **excluir um usuário do sistema**, para que **possa remover acessos indevidos ou desativados**.
- Como um **usuário da API**, eu quero **autenticar-me no sistema**, para que **possa acessar as funcionalidades protegidas**.
- Como um **usuário da API cliente**, eu quero **simular operações CRUD de usuários**, para que **possa testar a API de gerenciamento de usuários**.

Fase 2: Gerenciamento de Tópicos e Ideias (CIS)

Esta seção apresenta as histórias de usuário que contribuem para a Épica "Implementação do CIS API", abrangendo as funcionalidades centrais da plataforma de

ideação.

- Como um **usuário autenticado**, eu quero **criar um novo tópico**, para que **possa organizar as ideias por assunto**.
- Como um **usuário autenticado**, eu quero **visualizar a lista de tópicos existentes**, para que **possa escolher um assunto de interesse**.
- Como um **usuário autenticado**, eu quero **criar uma nova ideia dentro de um tópico**, para que **possa contribuir com minhas sugestões**.
- Como um **usuário autenticado**, eu quero **visualizar as ideias de um tópico específico**, para que **possa ver as sugestões de outros usuários**.
- Como um **usuário autenticado**, eu quero **votar em uma ideia**, para que **possa demonstrar meu apoio a ela**.
- Como um **usuário autenticado**, eu quero **cancelar meu voto em uma ideia**, para que **possa mudar minha opinião**.
- Como um **usuário da API cliente**, eu quero **simular operações CRUD de tópicos e ideias, incluindo votação**, para que **possa testar a API do CIS**.

Fase 3: Migração e Adaptação (MongoDB)

Esta seção descreve as histórias de usuário relacionadas à Épica "Mudanças na Infraestrutura", focando na transição para o novo banco de dados e adaptação dos componentes.

- Como um **administrador do sistema**, eu quero **migrar os dados de usuários existentes para o novo banco de dados (MongoDB)**, para que **o sistema possa usar a nova infraestrutura**.
- Como um **desenvolvedor**, eu quero **adaptar as APIs existentes para funcionar com o MongoDB**, para que **o sistema continue operando normalmente**.
- Como um **usuário do CLI**, eu quero **que o aplicativo CLI funcione com o novo banco de dados**, para que **possa continuar usando as funcionalidades via linha de comando**.

Histórias de Usuário são frequentemente derivadas de Épicas e Features maiores,

representando as menores unidades de trabalho que ainda entregam valor ao usuário.

Organização do Projeto

A implementação das fases do projeto será desenvolvida pelos alunos organizados em grupos, aplicando um modelo moderno de SDLC.

Estrutura e Metodologia

- **Grupos**: A implementação será desenvolvida em grupos (o tamanho será definido pelo tutor). O praticante organizará os alunos em equipes.
- **SDLC**: Cada grupo aplicará um modo SDLC moderno (após as cerimônias SCRUM) para implementar cada fase como uma equipe.
- **Scrum Master**: Cada equipe deve ter um Scrum Master designado para ajudar nas cerimônias.

Papéis do Tutor

O tutor ficará a cargo de um destes papéis para orientar as equipes:

- **Product Owner**: Orientação sobre os objetivos de cada componente do sistema.
- **Arquiteto**: Será responsável pelo projeto, pelos aspectos técnicos e pela estrutura dos módulos e pela solução geral.
- **Líder Técnico**: Será responsável por auxiliar nas dúvidas técnicas e orientar as equipes.

Cerimônias e Avaliação

- **Cerimônias SCRUM**: Semanalmente, cerimônias baseadas em SCRUM devem ser articuladas em cada equipe (sessão de planejamento, reunião ágil diária, etc.).
- **Avaliação**: O desenvolvimento será avaliado em duas dimensões:
    1. **Nível da Equipe**: Conquistas e progresso do grupo como um todo (entregas, documentação, planejamento).

2. **Nível Individual**: Conquistas e progresso de cada membro (tarefas atribuídas, qualidade do código, etc.).

Sistema Legado - Users CLI

Este tópico descreve o sistema legado "Users CLI", uma aplicação de linha de comando (CLI) desenvolvida em Java, responsável pelas operações de CRUD (Create, Read, Update, Delete) de usuários. Este sistema é fundamental para a Fase 1 do projeto Capstone, pois a nova API de usuários precisará interagir e manter a consistência com ele.

Tecnologias Utilizadas

O Users CLI é construído com as seguintes tecnologias:

- **Java (JDK 19):** A linguagem de programação principal.
- **Maven (https://maven.apache.org/):** Ferramenta de automação de build e gerenciamento de dependências.
- **MySQL (https://www.mysql.com/):** Banco de dados relacional utilizado para armazenar as informações dos usuários.
- **Docker (https://www.docker.com/) e Docker Compose (https://docs.docker.com/compose/):** Utilizados para orquestrar o ambiente do banco de dados MySQL.
- **MyBatis (https://mybatis.org/mybatis-3/):** Framework de persistência de dados que mapeia objetos Java para o banco de dados.
- **Picocli (https://picocli.info/):** Biblioteca para criar aplicações de linha de comando robustas e fáceis de usar.

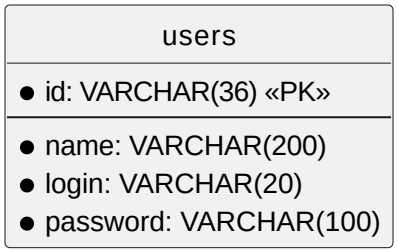
Estrutura do Banco de Dados

O sistema Users CLI interage com uma única tabela no banco de dados MySQL, chamada `users`. Esta tabela armazena as informações básicas de cada usuário.

Tabela users

Coluna	Tipo	Descrição
`id`	`VARCHAR(36)`	Identificador único do usuário (PK)
`name`	`VARCHAR(200)`	Nome completo do usuário
`login`	`VARCHAR(20)`	Nome de usuário para login (único)
`password`	`VARCHAR(100)`	Senha do usuário

Diagrama de Entidade-Relacionamento (ERD)



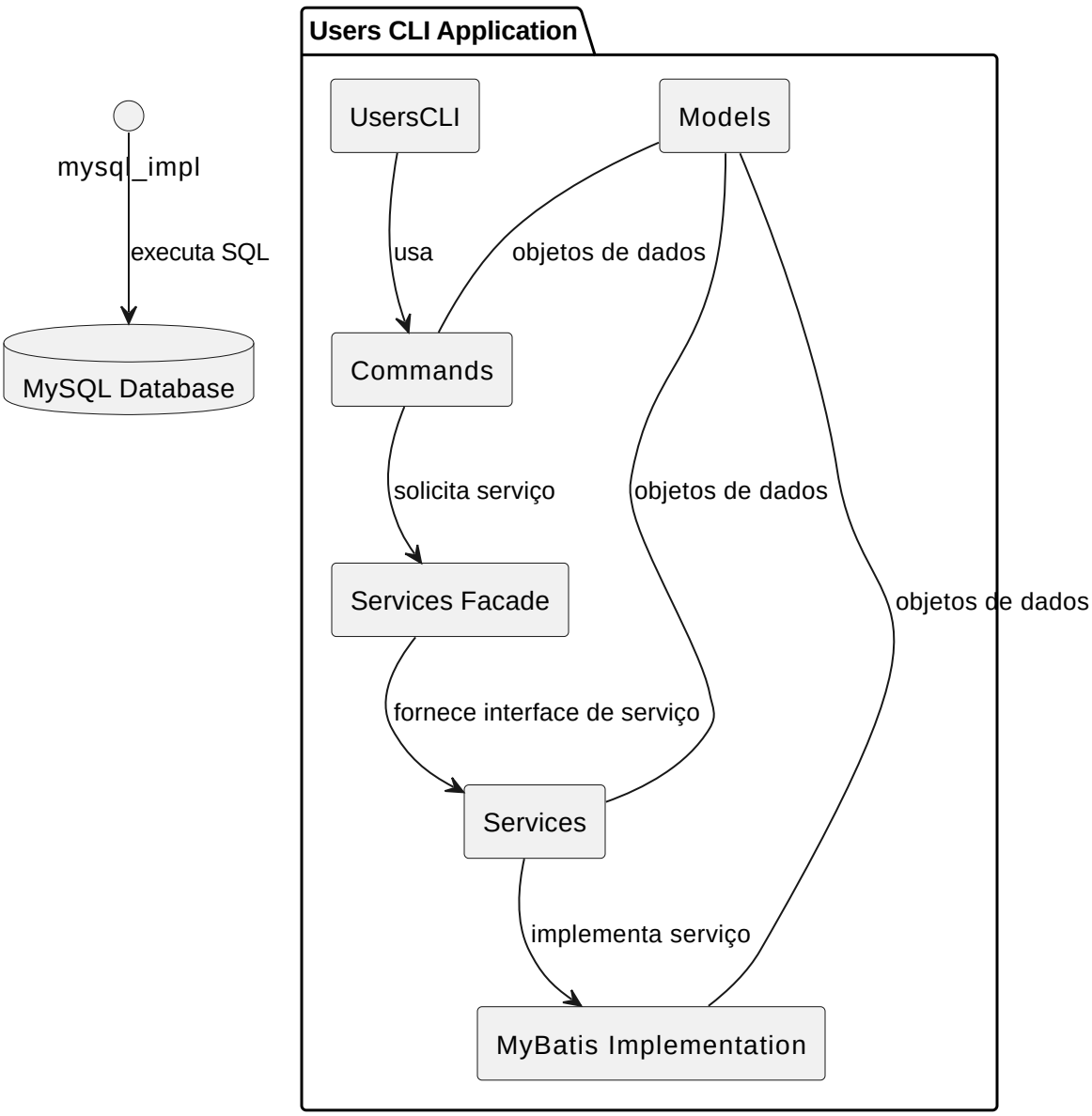
Arquitetura e Fluxo de Operações

Para entender como o Users CLI funciona internamente, podemos visualizá-lo como um sistema dividido em camadas, onde cada parte tem uma responsabilidade específica. Isso ajuda a organizar o código e a facilitar a manutenção.

Visão Geral da Arquitetura

O Users CLI segue uma arquitetura em camadas, composta principalmente por:

- **Interface de Usuário (CLI):** Onde o usuário interage com o sistema através de comandos de texto.
- **Camada de Serviços:** Contém a lógica de negócio principal e coordena as operações.
- **Camada de Persistência:** Responsável por interagir diretamente com o banco de dados.
- **Banco de Dados:** Onde os dados são armazenados.



Como uma Operação Acontece (Exemplo: Criação de Usuário)

Vamos detalhar o fluxo quando um usuário decide criar um novo registro:

1. **Entrada do Usuário:** O usuário digita um comando no terminal, como `java -jar UsersCLI.jar -create -n Maria -l maria.l -p senha123`.
2. **Processamento do Comando (Picocli):** A biblioteca `Picocli` (utilizada pela classe `UsersCLI`) interpreta os argumentos da linha de comando. Ela identifica que o comando `-create` foi acionado e direciona a execução para a classe Java correspondente, que é a `CreateUserCommand`.

3. **Lógica de Negócio (Camada de Serviços):** Dentro da `CreateUserCommand`, a lógica de negócio é acionada. Ela obtém uma instância do serviço de usuários (`UsersService`) através do `ServicesFacade` e chama o método `createUser()`, passando um objeto `User` com os dados fornecidos pelo usuário.

4. **Interação com o Banco de Dados (MyBatis):** A implementação concreta do `UsersService` (que é `MySqlDBService`) é responsável por se comunicar com o banco de dados. Ela utiliza o `MyBatis` para mapear o objeto `User` para uma instrução SQL `INSERT`. O `MyBatis` então executa essa instrução no banco de dados MySQL.

5. **Persistência:** O banco de dados MySQL recebe a instrução SQL e armazena os dados do novo usuário de forma persistente.

6. **Retorno:** O resultado da operação (sucesso ou falha) é retornado através das camadas de volta para a interface de linha de comando, que exibe uma mensagem ao usuário.

Estrutura do Código

O código-fonte do Users CLI está organizado em pacotes, refletindo as camadas da arquitetura e suas responsabilidades:

- `jalau.cis.commands`: **Interface com o Usuário.** Contém as classes que representam os comandos da linha de comando (e.g., `CreateUserCommand`, `ReadUsersCommand`). Elas são o ponto de entrada para as operações do usuário.

- `jalau.cis.models`: **Representação dos Dados.** Define as classes que estruturam os dados do sistema, como a classe `User`. Esses objetos são usados para transferir informações entre as camadas.

- `jalau.cis.services`: **Lógica de Negócio.** Este pacote define as interfaces de serviço (e.g., `UsersService`) que especificam as operações que o sistema pode realizar (criar, ler, atualizar, deletar usuários) de forma abstrata. O `ServicesFacade` atua como um ponto de entrada unificado para acessar esses serviços.

- `jalau.cis.services.mybatis`: **Implementação da Persistência.** Contém a lógica concreta de como as operações de serviço interagem com o banco de dados usando o framework `MyBatis`. É aqui que a tradução de objetos Java para SQL acontece.

- `jalau.cis.services.mybatis.UserMapper`: **Mapeamento SQL.** Uma interface crucial que o `MyBatis` implementa automaticamente. Ela contém métodos anotados com instruções SQL (e.g., `@Insert`, `@Select`), que o `MyBatis` executa no banco de dados quando os métodos são chamados.
- `jalau.cis.utils`: **Utilitários.** Contém classes com funções auxiliares e genéricas que podem ser usadas em diversas partes do sistema, como `StringUtils`.

Fase 1: Suporte a Sistemas Legados

O objetivo principal desta fase é dar o primeiro passo na modernização de um sistema existente, garantindo a coexistência entre o novo e o antigo.

Cenário Detalhado

O CIS (Sistema de Informação Centralizado) deve ser compatível e integrado com um sistema "Legacy" (legado) já em operação. Este sistema legado é composto por:

1. **Aplicação CLI (Interface de Linha de Comando) em Java**: Um programa de console existente, desenvolvido em Java, que interage diretamente com os dados do usuário.
2. **Banco de Dados Relacional (RDBMS MySQL)**: Um banco de dados MySQL que serve como a fonte primária para o armazenamento de dados de usuário, incluindo identificador, login, nome e senha.

A nova solução CIS tem como premissa fundamental a utilização e integração com os dados de usuário já existentes neste sistema legado. Um desafio crucial e um requisito não-funcional de alta prioridade é a **consistência dos dados**. Isso implica que qualquer alteração nos dados do usuário (como exclusão, criação ou atualizações) deve ser propagada e mantida de forma síncrona ou assíncrona (a ser definida) entre a nova solução CIS e o sistema legado.

Uma das principais regras arquitetônicas para esta fase é que o sistema "Legacy" e o CIS devem **coexistir**, garantindo que ambos possam ser operáveis simultaneamente sem interrupções ou inconsistências de dados.

Contexto e Boas Práticas para a Fase 1

Para o desenvolvimento desta fase, aplicaremos conceitos e boas práticas abordados no curso de Desenvolvimento de Software 3, visando a construção de uma solução robusta e de alta qualidade:

- **Metodologia Ágil (SDLC)**: Adotaremos um Ciclo de Vida de Desenvolvimento de Software (SDLC) ágil, como o Scrum, para gerenciar o projeto. Isso permitirá entregas incrementais, feedback contínuo e adaptação a mudanças.
- **Análise de Requisitos**: A compreensão aprofundada dos requisitos funcionais (CRUD de usuários) e não-funcionais (consistência de dados, segurança, desempenho) será fundamental. Utilizaremos técnicas de análise de requisitos para garantir que a API atenda às necessidades do negócio e do sistema legado.
- **Arquitetura de Software**: A API de usuário será projetada com base em princípios de arquitetura de software, considerando padrões e estilos que promovam modularidade, manutenibilidade e escalabilidade.
- **Boas Práticas de Codificação**: Seguiremos princípios como YAGNI (You Ain't Gonna Need It), KISS (Keep It Simple, Stupid), DRY (Don't Repeat Yourself) e SOC (Separation of Concerns) para escrever um código limpo, eficiente e fácil de manter.
- **Testes de Desenvolvimento**: A qualidade será garantida através de uma estratégia de testes abrangente, incluindo testes unitários, de integração e, se aplicável, testes end-to-end. A prática de TDD (Desenvolvimento Guiado por Testes) será incentivada para garantir a cobertura e a robustez do código.
- **Documentação**: A documentação será uma parte integrante do processo, incluindo a especificação da API (OpenAPI/Swagger) e comentários claros no código-fonte.

Tópicos Essenciais e Referências

Para aprofundar o conhecimento sobre os conceitos abordados nesta fase, consulte os seguintes recursos:

- **RESTful**: Estilo arquitetural para sistemas distribuídos.
  - RESTful API Tutorial (https://restfulapi.net/)
- **Spring Boot**: Framework para desenvolvimento de aplicações Java.
  - Site Oficial do Spring Boot (https://spring.io/projects/spring-boot)
- **OpenAPI/Swagger**: Especificação para descrever APIs RESTful.

    - Swagger na Prática (Klaus Laube) (https://klauslaube.com.br/2018/03/15/swagger-na-pratica.html)

- **API First**: Abordagem de desenvolvimento onde a API é projetada e definida antes da implementação.
    - API First: Processo e Ferramentas (Klaus Laube) (https://klauslaube.com.br/2020/03/02/api-first-processo-e-ferramentas.html)
    - Adoção de API First: O que você precisa saber (API Brasil) (https://apibrasil.blog/adocao-de-api-first-o-que-voce-precisa-saber/)
    - API FIRST - O PASSO QUE FALTAVA NO SEU PROJETO (YouTube) (https://www.youtube.com/watch?v=ULfIBpeIGFk)

Fase 1: Especificações Detalhadas

Nesta primeira fase do Projeto Capstone, as seguintes funcionalidades e requisitos técnicos são mandatórios para o sucesso da integração e modernização do sistema:

- **API de Usuário Moderna (CRUD)**:
  - **Requisito Funcional Principal**: A API deve permitir a execução completa das operações **CRUD (Create, Read, Update, Delete)** sobre os dados de usuário.
  - **Modernidade**: A API deve seguir padrões de design modernos para APIs, preferencialmente RESTful, garantindo facilidade de uso, escalabilidade e manutenibilidade.
- **Consistência de Dados**:
  - **Requisito Não-Funcional Crítico**: A API de usuário deve garantir a **consistência total dos dados** com o sistema legado. Isso significa que qualquer alteração realizada através da nova API deve ser refletida de forma precisa e imediata (ou conforme estratégia definida) no banco de dados MySQL existente do sistema legado, e vice-versa.
- **Tecnologia: Java**:
  - **Restrição Técnica**: Por motivos de alinhamento com a infraestrutura e o ecossistema tecnológico existente, a implementação desta API deve ser exclusivamente na linguagem **Java**. Frameworks como Spring Boot são fortemente recomendados para agilizar o desenvolvimento e aproveitar as melhores práticas do ecossistema Java para APIs.
- **Cliente de Teste**:
  - **Ferramenta de Validação**: Será necessário desenvolver um cliente de API simples. Este cliente terá como objetivo principal operar e validar a API de usuário, simulando operações CRUD para um número "n" de usuários. Isso permitirá verificar a funcionalidade, o desempenho e a robustez da API principal sob diferentes cenários de uso.
- **Segurança**:

    - **Requisito Não-Funcional Essencial**: A API de usuário deve incorporar um nível adequado de segurança. O acesso a qualquer endpoint da API que manipule dados de usuário (CRUD) deve ser **autorizado** através de um mecanismo de autenticação, como login e senha. A implementação de Spring Security, por exemplo, é uma abordagem recomendada.
- **Design da API (OpenAPI/Swagger)**:
    - **Prática de Engenharia de Software**: Antes de qualquer implementação de código, a API deve ser formalmente definida. Utilizaremos a especificação **OpenAPI (anteriormente conhecida como Swagger)** para descrever todos os endpoints, modelos de dados (schemas), tipos de requisição/resposta, parâmetros e mecanismos de segurança. Esta prática garante clareza, facilita a comunicação entre equipes e permite a geração automática de documentação interativa.

Tópicos Essenciais e Referências

Para aprofundar o conhecimento sobre os conceitos abordados nesta fase, consulte os seguintes recursos:

- **Postman**: Ferramenta para testar e documentar APIs.
    - Site Oficial do Postman (https://www.postman.com/)
- **OpenAPI/Swagger**: Especificação para descrever APIs RESTful.
    - Swagger na Prática (Klaus Laube) (https://klauslaube.com.br/2018/03/15/swagger-na-pratica.html)
- **API First**: Abordagem de desenvolvimento onde a API é projetada e definida antes da implementação.
    - API First: Processo e Ferramentas (Klaus Laube) (https://klauslaube.com.br/2020/03/02/api-first-processo-e-ferramentas.html)
    - Adoção de API First: O que você precisa saber (API Brasil) (https://apibrasil.blog/adocao-de-api-first-o-que-voce-precisa-saber/)
    - API FIRST - O PASSO QUE FALTAVA NO SEU PROJETO (YouTube) (https://www.youtube.com/watch?v=ULfIBpeIGFk)

- **JSON Web Tokens (JWT)**: Padrão para criação de tokens de acesso seguros.
  - Introdução ao JWT (https://jwt.io/introduction)
- **Autenticação Baseada em Sessão**: Mecanismo de autenticação comum em aplicações web.
  - Autenticação de Sessão (MDN Web Docs) (https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Authentication)
- **RESTful**: Estilo arquitetural para sistemas distribuídos.
  - RESTful API Tutorial (https://restfulapi.net/)
- **Spring Boot**: Framework para desenvolvimento de aplicações Java.
  - Site Oficial do Spring Boot (https://spring.io/projects/spring-boot)

Fase 2: Implementação da API CIS

O principal objetivo desta fase é desenvolver a funcionalidade central da Solução de Ideação Colaborativa (CIS - Collaborative Ideation Solution). Esta fase marca a construção do coração do novo sistema, permitindo a gestão de tópicos, ideias e votos.

Visão Geral Detalhada

Nesta fase, será desenvolvido um novo sistema que implementará as funcionalidades específicas do CIS. Este sistema deverá se integrar de forma transparente com o sistema legado, utilizando a **API de Usuário** desenvolvida e consolidada na Fase 1 para todas as operações relacionadas a usuários, especialmente autenticação e autorização.

O processo de desenvolvimento seguirá as diretrizes de um **Ciclo de Vida de Desenvolvimento de Software (SDLC) ágil**, aproveitando os artefatos e as lições aprendidas na Fase 1. A escolha da tecnologia de software será crucial para garantir a qualidade, o detalhamento e a manutenibilidade da solução.

A arquitetura proposta para esta fase é centrada na nova **API CIS**. Esta API terá duas responsabilidades principais:

1. **Comunicação com a API de Usuário (Fase 1)**: Para autenticar e autorizar as requisições dos usuários, a API CIS fará chamadas à API de Usuário existente. Isso garante a reutilização da lógica de segurança e a consistência dos dados de usuário.
2. **Interação com o Banco de Dados Legado**: A API CIS será responsável por armazenar e gerenciar os novos dados específicos do CIS (tópicos, ideias e votos) diretamente no banco de dados legado. Isso implica em estender o esquema do banco de dados existente para acomodar essas novas entidades.

Contexto e Boas Práticas para a Fase 2

Para o desenvolvimento desta fase, aplicaremos conceitos e boas práticas abordados no curso de Desenvolvimento de Software 3, com foco nos desafios específicos da Fase 2:

- **Modelagem de Dados**: Será essencial projetar cuidadosamente o esquema das novas tabelas (tópicos, ideias, votos) e suas relações com a tabela de usuários existente no

banco de dados legado. A integridade referencial e a otimização para consultas serão considerações importantes.

- **Design de API (RESTful)**: A API CIS deve seguir os princípios RESTful para garantir uma interface clara, consistente e escalável. A definição dos recursos (tópicos, ideias, votos) e das operações HTTP (GET, POST, PUT, DELETE) será fundamental.
- **Autenticação e Autorização**: A integração com a API de Usuário da Fase 1 para autenticação e autorização será um ponto crítico. Será necessário definir como os tokens de autenticação (por exemplo, JWT) serão passados e validados entre as APIs.
- **Testes de Integração**: Dada a dependência da API CIS com a API de Usuário e o banco de dados legado, os testes de integração serão de suma importância para garantir que todas as partes do sistema funcionem harmoniosamente.
- **Segurança**: Além da autenticação, outras considerações de segurança, como validação de entrada, proteção contra injeção de SQL e tratamento de erros, devem ser aplicadas.
- **Documentação (OpenAPI/Swagger)**: A API CIS deve ser totalmente documentada usando OpenAPI/Swagger, detalhando todos os endpoints, modelos de dados e mecanismos de segurança. Isso facilitará o consumo da API por clientes e outras partes do sistema.
- **Manutenibilidade e Escalabilidade**: O código deve ser escrito de forma limpa, modular e testável, seguindo princípios como DRY, KISS, YAGNI e SOC, para garantir a manutenibilidade a longo prazo e a capacidade de escalar o sistema conforme a necessidade.

Tópicos Essenciais e Referências

Para aprofundar o conhecimento sobre os conceitos abordados nesta fase, consulte os seguintes recursos:

- **APIs RESTful**:
  - RESTful API Tutorial (https://restfulapi.net/)
- **Autenticação e Autorização**:

- JSON Web Tokens (JWT) (https://jwt.io/introduction)
- Autenticação de Sessão (MDN Web Docs) (https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Authentication)

- **Spring Boot**:
  - Site Oficial do Spring Boot (https://spring.io/projects/spring-boot)
- **OpenAPI/Swagger**:
  - Swagger na Prática (Klaus Laube) (https://klauslaube.com.br/2018/03/15/swagger-na-pratica.html)
- **API First**:
  - API First: Processo e Ferramentas (Klaus Laube) (https://klauslaube.com.br/2020/03/02/api-first-processo-e-ferramentas.html)
  - Adoção de API First: O que você precisa saber (API Brasil) (https://apibrasil.blog/adocao-de-api-first-o-que-voce-precisa-saber/)

Fase 2: Especificações Detalhadas

Na segunda fase do Projeto Capstone, o sistema deve atender aos seguintes requisitos, que detalham as funcionalidades e interações necessárias para a Solução de Ideação Colaborativa (CIS):

- **Extensão do Banco de Dados**:
  - **Requisito**: O banco de dados relacional (MySQL) existente, que atualmente armazena os dados de usuário, deve ser estendido.
  - **Detalhes**: Novas tabelas devem ser criadas para armazenar as entidades centrais do CIS:
    - **Tópicos**: Representam as categorias ou temas sob os quais as ideias são propostas.
    - **Ideias**: São as propostas detalhadas pelos usuários dentro de um tópico específico.
    - **Votos**: Registram a interação dos usuários com as ideias, permitindo votar a favor ou contra uma ideia.
  - **Implicação**: A modelagem dessas novas tabelas deve considerar as relações com a tabela de usuários existente e garantir a integridade referencial.
- **API CIS Moderna**:
  - **Requisito**: Desenvolver uma API moderna (preferencialmente RESTful) que exponha as funcionalidades do CIS.
  - **Operações CRUD**: A API deve suportar as operações CRUD completas para as novas entidades:
    - **Tópicos**: Criar, Ler (listar todos, ler por ID), Atualizar, Excluir.
    - **Ideias**: Criar (associada a um tópico e um usuário), Ler (listar todas, listar por tópico, ler por ID), Atualizar, Excluir.
    - **Votos**: Registrar voto (associado a uma ideia e um usuário), Cancelar voto.

    - **Lógica de Negócio**: A API deve encapsular a lógica de negócio para garantir a consistência e validade das operações (ex: um usuário só pode votar uma vez por ideia).

- **Autenticação**:
    - **Requisito**: Todas as operações da API CIS que exigem identificação do usuário (ex: criar tópico, criar ideia, votar) devem ser autenticadas.
    - **Integração**: A autenticação deve ser realizada através da **API de Usuário** desenvolvida na Fase 1. Isso implica que a API CIS atuará como um cliente da API de Usuário para validar as credenciais ou tokens de autenticação recebidos.
- **Cliente de Teste**:
    - **Requisito**: Desenvolver um cliente de API simples para operar e validar a API CIS.
    - **Simulação**: Este cliente deve ser capaz de simular o comportamento de "n" usuários de forma aleatória, realizando as seguintes ações:
        - Criação de tópicos.
        - Criação de ideias dentro de tópicos.
        - Votação e cancelamento de votação em ideias.
    - **Objetivo**: O objetivo é validar a funcionalidade, a robustez e o desempenho da API CIS sob carga e em cenários de uso variados.
- **Design da API**:
    - **Requisito**: Assim como na Fase 1, a API CIS deve ser formalmente definida antes de iniciar qualquer implementação de código.
    - **Ferramenta**: Utilizar uma especificação como **OpenAPI/Swagger** para descrever todos os endpoints, modelos de dados (schemas para Tópicos, Ideias, Votos), tipos de requisição/resposta, parâmetros e mecanismos de segurança. Esta prática é crucial para a comunicação entre equipes e para a geração automática de documentação.

Fase 3: Mudanças na Infraestrutura

O principal objetivo desta fase é modernizar a infraestrutura de persistência de dados do sistema, refletindo a evolução tecnológica e as necessidades de escalabilidade e flexibilidade.

Visão Geral Detalhada

Nesta fase, o foco principal é a transição da tecnologia de banco de dados para os dados do CIS (Collaborative Ideation Solution). Atualmente, o sistema utiliza um RDBMS (MySQL), mas devido à evolução tecnológica e às características dos dados do CIS (tópicos, ideias, votos), um banco de dados **NoSQL, especificamente o MongoDB**, será adotado como substituto.

O desenvolvimento nesta fase será construído sobre os artefatos e o conhecimento adquiridos na Fase 2. Continuaremos a seguir um **Ciclo de Vida de Desenvolvimento de Software (SDLC) ágil**, garantindo a adaptação e a entrega contínua. A escolha das tecnologias de software será orientada pela necessidade de integração com o novo banco de dados e pela manutenção da qualidade do sistema.

A arquitetura do sistema será reconfigurada para que o **MongoDB** se torne o banco de dados principal para todas as informações relacionadas ao CIS. A integração com o sistema legado de usuários (e seu RDBMS original) será cuidadosamente avaliada. Dependendo da estratégia de migração e da necessidade de manter a compatibilidade com a API de Usuário da Fase 1, a comunicação com o RDBMS legado pode ser mantida para dados de usuário, ou os dados de usuário podem ser migrados para o MongoDB, se a estratégia permitir.

Contexto e Boas Práticas para a Fase 3

Para o desenvolvimento desta fase, aplicaremos conceitos e boas práticas abordados no curso de Desenvolvimento de Software 3, com foco nos desafios específicos da Fase 3:

- **Bancos de Dados NoSQL (MongoDB)**: Será fundamental entender os princípios e as melhores práticas de modelagem de dados para bancos de dados NoSQL, que diferem significativamente dos RDBMS. A flexibilidade do esquema do MongoDB e suas capacidades de escalabilidade serão exploradas.

- **Adaptação de APIs**: As APIs desenvolvidas nas Fases 1 e 2 precisarão ser adaptadas para interagir com o MongoDB. Isso pode envolver a atualização de drivers de banco de dados, ORMs/ODMs (Object-Document Mappers) e a lógica de persistência.
- **Estratégias de Migração de Dados**: A migração dos dados existentes do MySQL para o MongoDB será um processo crítico. Será necessário planejar cuidadosamente a estratégia de migração para garantir a integridade, a consistência e a precisão dos dados durante a transição.
- **Continuidade de Negócio**: Um requisito chave é garantir que o sistema continue funcionando sem interrupções significativas durante e após a transição do banco de dados. Isso exigirá um planejamento rigoroso de testes e implantação.
- **Otimização de Desempenho**: A mudança para um novo banco de dados pode impactar o desempenho. Será crucial monitorar e otimizar o desempenho do sistema no MongoDB, ajustando consultas, índices e configurações conforme necessário.
- **Testes Abrangentes**: Testes de regressão, testes de integração e testes de desempenho serão essenciais para validar a funcionalidade e o desempenho do sistema após a mudança de infraestrutura.
- **CI/CD para Mudanças de Infraestrutura**: A automação através de pipelines de CI/CD será vital para gerenciar as mudanças na infraestrutura, garantindo que as implantações sejam consistentes e confiáveis.

Tópicos Essenciais e Referências

Para aprofundar o conhecimento sobre os conceitos abordados nesta fase, consulte os seguintes recursos:

- **Bancos de Dados NoSQL**:
  - O que é NoSQL? (MongoDB) (https://www.mongodb.com/pt-br/nosql-explained)
- **MongoDB**:
  - Site Oficial do MongoDB (https://www.mongodb.com/)
  - MongoDB Tutorial (MongoDB University) (https://university.mongodb.com/)

- **Testes de Desempenho**:
  - Testes de Desempenho: O que são e como fazer (Alura) (https://www.alura.com.br/artigos/testes-de-desempenho-o-que-sao-e-como-fazer)
- **CI/CD**:
  - CI/CD: O que é e por que é importante (Red Hat) (https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd)

Fase 3: Especificações Detalhadas

Nesta fase do Projeto Capstone, o foco principal é a transição da infraestrutura de persistência de dados. Os seguintes requisitos detalham as mudanças e os desafios a serem superados:

- **Substituição do Banco de Dados**:
  - **Requisito**: O banco de dados relacional (MySQL) atualmente em uso para os dados do CIS deve ser substituído por um banco de dados **NoSQL, especificamente o MongoDB**.
  - **Implicação**: Esta mudança exige uma reavaliação da modelagem de dados e da lógica de persistência em todas as APIs que interagem com os dados do CIS.
- **Adaptação das APIs**:
  - **Requisito**: As APIs desenvolvidas nas Fases 1 (API de Usuário) e 2 (API CIS) devem ser adaptadas para serem totalmente compatíveis e funcionais com o novo banco de dados NoSQL (MongoDB).
  - **Implicação**: Isso pode envolver a atualização de drivers de banco de dados, a reescrita de consultas e a adaptação de ORMs/ODMs para o paradigma NoSQL.
- **Continuidade Funcional**:
  - **Requisito**: Após a alteração do banco de dados, o sistema como um todo deve continuar a funcionar exatamente como antes, sem perda de funcionalidades ou regressões.
  - **Implicação**: Testes de regressão abrangentes serão cruciais para garantir que todas as funcionalidades existentes operem corretamente com o MongoDB.
- **Atualização do CLI**:
  - **Requisito**: O aplicativo de linha de comando (CLI) legado, que interage com os dados de usuário, será atualizado externamente para ser compatível com a nova infraestrutura.

    - **Implicação**: Embora a atualização do CLI não seja parte do escopo principal de desenvolvimento desta fase, o sistema deve ser projetado para suportar essa atualização externa, garantindo a compatibilidade da API de Usuário com o CLI.
- **Migração de Dados**:
    - **Requisito**: Todos os dados existentes no banco de dados original (MySQL) devem ser migrados para o MongoDB com total precisão e integridade.
    - **Implicação**: Será necessário desenvolver um plano e ferramentas de migração robustos para garantir que nenhum dado seja perdido ou corrompido durante o processo.
- **Desempenho**:
    - **Requisito**: O desempenho adequado do sistema deve ser mantido ou melhorado após a movimentação dos dados e a mudança na tecnologia de banco de dados.
    - **Implicação**: Monitoramento contínuo, otimização de consultas e indexação no MongoDB serão essenciais para garantir que a performance do sistema atenda aos requisitos.

Ferramentas e Tecnologias

Esta seção detalha as ferramentas, tecnologias e metodologias que formam a base do projeto.

Ambientes de Desenvolvimento (IDEs)

- **IntelliJ IDEA (https://www.jetbrains.com/idea/):** IDE robusta para desenvolvimento em Java, oferecendo análise de código avançada, depuração e ferramentas de build integradas.
- **Visual Studio Code (https://code.visualstudio.com/):** Um editor de código-fonte leve, mas poderoso, com suporte para uma vasta gama de linguagens, incluindo Java e C#, através de extensões.

Linguagens de Programação

- **Java (https://dev.java/):** Linguagem principal para o desenvolvimento do back-end, incluindo a API de Usuários e a integração com o sistema legado.
- **C# (https://learn.microsoft.com/en-us/dotnet/csharp/):** Utilizada no desenvolvimento da API do CIS na Fase 2. É uma linguagem moderna para a construção de microserviços no ecossistema .NET.

Bancos de Dados

- **MySQL (https://www.mysql.com/):** Utilizado na Fase 1 como o banco de dados relacional (RDBMS) para persistir os dados do sistema legado de usuários.
- **MongoDB (https://www.mongodb.com/):** Introduzido na Fase 3, é um banco de dados NoSQL orientado a documentos, escolhido pela sua flexibilidade e escalabilidade.

Frameworks e Bibliotecas de Teste

- **JUnit 5 (https://junit.org/junit5/):** Framework padrão para testes em Java.
- **Mockito (https://site.mockito.org/):** Framework de mocking para criar objetos de teste em Java, permitindo o isolamento de componentes.
- **xUnit (https://xunit.net/):** Uma ferramenta de teste unitário popular e focada na comunidade para o framework .NET.
- **Moq (https://github.com/moq/moq):** A biblioteca de mocking mais popular para C#, facilitando o isolamento de dependências em testes.

Ferramentas de Build e Dependência

- **Maven (https://maven.apache.org/):** Gerenciador de dependências e ferramenta de automação de build para projetos Java.

Clientes de API

Para testar e interagir com as APIs desenvolvidas, recomendamos os seguintes clientes:

- **Postman (https://www.postman.com/):** Uma plataforma completa para desenvolvimento de APIs, permitindo a criação, teste e documentação de requisições.
- **Insomnia (https://insomnia.rest/):** Um cliente de API de código aberto com uma interface limpa e foco em GraphQL e gRPC, além de REST.
- **Bruno (https://www.usebruno.com/):** Um cliente de API mais recente e de código aberto que armazena as coleções de requisições diretamente no sistema de arquivos, facilitando a colaboração via Git.

CI/CD e DevOps

- **GitLab CI/CD (https://docs.gitlab.com/ee/ci/):** Plataforma utilizada para automação dos processos de Integração Contínua e Entrega Contínua (CI/CD).
- **Docker (https://www.docker.com/):** Ferramenta para criar, implantar e executar aplicações em contêineres, garantindo ambientes consistentes.

- **YAML (https://yaml.org/):** Linguagem de serialização de dados usada para definir os pipelines no GitLab CI/CD e configurações do Docker.

Documentação

- **Writerside (https://www.jetbrains.com/writerside/):** A ferramenta da JetBrains que estamos utilizando para criar e manter a documentação técnica do projeto.
- **GitLab Wiki (https://docs.gitlab.com/ee/user/project/wiki/):** Ferramenta de documentação colaborativa integrada ao GitLab, ideal para manter notas, guias e conhecimento sobre o projeto.
- **Markdown (https://www.markdownguide.org/):** Linguagem de marcação leve utilizada para escrever em ambas as ferramentas de documentação.

Troubleshooting

Problemas Comuns do Users CLI

Aqui estão alguns problemas comuns e suas soluções relacionados ao Users CLI:

- **Erro:** `docker-compose: command not found`
  - **Causa:** Docker Compose não está instalado ou não está no PATH do seu sistema.
  - **Solução:** Siga o guia oficial de instalação do Docker Compose (https://docs.docker.com/compose/install/).
- **Erro:** `Cannot connect to the Docker daemon`
  - **Causa:** O serviço Docker não está em execução.
  - **Solução:** Inicie o daemon Docker no seu sistema. Isso geralmente é feito via `systemctl start docker` no Linux ou iniciando o aplicativo Docker Desktop no Windows/macOS.
- **Erro: Connection refused ou** `Can't connect to local MySQL server`
  - **Causa:** O contêiner do banco de dados não está em execução.
  - **Solução:** Certifique-se de que o contêiner Docker esteja ativo e em execução, executando `docker-compose ps`. Se não estiver em execução, inicie-o com `docker-compose up -d`.
- **Erro:** `Table 'sd3.users' doesn't exist`
  - **Causa:** A tabela `users` não foi criada após o início do contêiner.
  - **Solução:** Você precisa executar o comando para criar a tabela conforme especificado na seção de **Requisitos** do Users CLI.
- **Erro ao compilar:** `invalid target release: 19` **ou similar**
  - **Causa:** Você não está usando o JDK 19 para compilar o projeto.

- **Solução:** Certifique-se de ter o JDK 19 instalado e que ele seja o JDK padrão para o seu ambiente. Você pode verificar com `java -version` e `mvn -version`.

- **Erro: `no main manifest attribute` ou `NoClassDefFoundError` ao executar o JAR**
  - **Causa:** A aplicação não foi empacotada corretamente. O JAR executável deve incluir todas as suas dependências (um "fat JAR").
  - **Solução:** Certifique-se de ter compilado o projeto usando `mvn clean package`. Este comando está configurado para gerar o arquivo JAR executável correto na pasta `target/`.

- **Erro: `Access denied for user 'root'@'...'`**
  - **Causa:** As credenciais no seu arquivo `sd3.xml` não correspondem às credenciais esperadas pelo banco de dados.
  - **Solução:** Certifique-se de que as propriedades `username` e `password` em `sd3.xml` estejam definidas como `root` e `sd5`, respectivamente, para corresponder aos valores em `docker-compose.yml`.