Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Lucas Tadeu de Paula da Conceição

NoSQL com MongoDB e Scrum: Uma Abordagem Moderna para Dados e Projetos Ágeis

Este trabalho tem como objetivo apresentar os fundamentos dos bancos de dados NoSQL, com foco especial no MongoDB, uma das tecnologias mais utilizadas nesse paradigma. Inicialmente, é apresentado o que são bancos de não relacionais, destacando as vantagens da estrutura e escalabilidade oferecida pelo NoSQL. Em seguida, o MongoDB é explorado em mais detalhes, evidenciando seu funcionamento, suas características técnicas e suas aplicações no contexto moderno de desenvolvimento. Por fim, é realizada a análise de um artigo acadêmico que demonstra a utilização do MongoDB em conjunto com a metodologia ágil Scrum, evidenciando como a flexibilidade desse banco de dados pode contribuir para o gerenciamento e execução eficiente de projetos ágeis.

CAMPOS DO JORDÃO

2025

Lucas Tadeu de Paula da Conceição

NoSQL com MongoDB e Scrum: Uma Abordagem Moderna para Dados e Projetos Ágeis

Trabalho apresentado ao curso de análise e desenvolvimento de sistemas

Bando de dados 2-CJOBDD2

Professor: Paulo Giovani de Faria Zeferino

CAMPOS DO JORDÃO

2025

Sumário

1.Introdução

Com o crescimento exponencial do volume de dados e a necessidade de escalabilidade em sistemas modernos, os bancos de dados tradicionais baseados no modelo relacional passaram a enfrentar limitações quanto à flexibilidade, desempenho e agilidade. Nesse cenário, surgem os bancos de dados NoSQL, que oferecem uma alternativa mais adequada a diversas aplicações que demandam maior escalabilidade horizontal, manipulação de grandes volumes de dados não estruturados e maior velocidade nas operações. Segundo Alexandre Porcelli (2010), o movimento NoSQL (Not Only SQL) teve sua origem em 2009, com o objetivo de promover um encontro entre profissionais e pesquisadores da área, organizado por Johan Oskarsson, para discutir novas abordagens em armazenamento de dados que fugissem do paradigma relacional tradicional. Dentre as diversas soluções NoSQL, o MongoDB foi a escolhida neste trabalho, por sua simplicidade de uso e flexibilidade no armazenamento de documentos no formato BSON (uma extensão do JSON).

1.1 Objetivo Geral

Este artigo tem como objetivo geral apresentar os conceitos fundamentais dos bancos de dados NoSQL, com ênfase no MongoDB, explorando suas características, vantagens, desvantagens e aplicação em um projeto.

1.2 Objetivo Específicos

Este trabalho apresenta os conceitos fundamentais da tecnologia NoSQL, com ênfase no banco de dados MongoDB, tem como objetivos específicos:

- Apresentar o artigo *"Desenvolvimento de aplicação para o controle de Scrum com MongoDB"*, destacando sua proposta, arquitetura e justificativas técnicas.

- Explicar o que são bancos de dados NoSQL

- Apresentar de forma aprofundada o funcionamento, a estrutura e os principais recursos do banco de dados MongoDB, destacando suas vantagens, aplicações práticas e diferenciais em relação a outras soluções NoSQL.

1.3 justificativa

o crescimento acelerado da quantidade de dados gerados por aplicações modernas, torna-se evidente a necessidade de soluções que ofereçam maior flexibilidade, escalabilidade e desempenho do que os bancos de dados relacionais tradicionais. **Os bancos NoSQL** surgem como alternativas eficazes para essas demandas, com modelagens mais dinâmicas, escalabilidade horizontal e melhor desempenho em aplicações distribuídas. Assim, este trabalho justifica-se pela necessidade de explorar e apresentar o **NoSQL** com o **MongoDB** como uma solução robusta e atual para o desenvolvimento de sistemas que exigem alta disponibilidade, eficiência e adaptabilidade no tratamento de dados.

1.4 Aspectos Metodológicos

Este trabalho adota uma abordagem metodológica de natureza **qualitativa**, com base em dois eixos principais: a **pesquisa bibliográfica** e a **realização de um estudo de caso** sobre o projeto de **SANTOS, Raduan Silva.** *Desenvolvimento de aplicação para o controle de Scrum com MongoDB. 2011*.

1.5 Aporte Teórico

Esta pesquisa fundamenta-se, principalmente, na documentação oficial do MongoDB disponível em *MongoDB.org*, bem como no trabalho de conclusão de curso de **SANTOS, Raduan Silva dos** (2011), intitulado *"Desenvolvimento de aplicação para o controle de Scrum com MongoDB"*

2.Metodologia

Esta pesquisa adota uma abordagem **qualitativa e aplicada**, dividindo-se em duas etapas principais: **pesquisa bibliográfica** e **estudo de caso prático**. Na primeira etapa, realizou-se uma **pesquisa bibliográfica** com o objetivo de levantar e analisar conceitos, fundamentos e características relacionados aos bancos de dados NoSQL, com ênfase no MongoDB. As principais fontes consultadas incluem artigos científicos, livros técnicos, documentação oficial da ferramenta (*MongoDB.org*), além do trabalho de Santos (2011), que serviu como referência prática para o desenvolvimento da aplicação. A segunda etapa consistiu em um **estudo de caso prático**, por meio do desenvolvimento de uma aplicação web com foco na gestão de grandes volumes de dados, utilizando o MongoDB como banco de dados principal. Esse estudo teve como finalidade demonstrar, na prática, os benefícios da tecnologia NoSQL em termos de escalabilidade, desempenho e flexibilidade estrutural. A aplicação foi construída com tecnologias web modernas e integrada ao MongoDB, explorando recursos como **sharding** e **consultas orientadas a documentos**. Durante essa fase, foram observados aspectos técnicos do funcionamento do banco de dados, como desempenho em operações de leitura e escrita, modelagem de dados e adaptação a estruturas dinâmicas. Por fim, os resultados obtidos foram analisados à luz da fundamentação teórica, a fim de avaliar a eficácia do MongoDB no contexto proposto e validar a hipótese de que bancos de dados NoSQL representam uma alternativa viável aos modelos relacionais em sistemas que exigem alta escalabilidade e desempenho.

2.1 MongoDB

O **MongoDB** é um banco de dados **NoSQL orientado a documentos**, desenvolvido em C++, que se destaca por sua alta escalabilidade tanto vertical quanto horizontal , suporte robusto a consultas complexas, e por oferecer **índices completos em todos os campos**. Além disso, conta com o **GridFS**, um sistema eficiente para armazenamento de arquivos de qualquer tamanho, permitindo o gerenciamento simplificado de grandes volumes de dados (*MongoDB, 2024*). De acordo com **Alexandre Porcelli (2010)**, o MongoDB foi

criado por **Dwight Merriman** e **Eliot Horowitz**, fundadores da empresa **10gen**, atualmente conhecida como **MongoDB Inc**. A estrutura do MongoDB é baseada no modelo de dados **orientado a documentos**, permitindo armazenar informações em formato BSON (uma extensão binária do JSON), o que garante maior flexibilidade. Uma característica que o diferencia de outros bancos NoSQL é o **suporte a consultas ricas**, permitindo operações complexas e expressivas sobre grandes conjuntos de dados.

2.1.1 Sharding

Para aplicações de grande porte e alta complexidade, o MongoDB pode ser utilizado sem comprometer o desempenho, graças à tecnologia de **auto-sharding**. Essa funcionalidade permite a divisão automática dos dados em **shards**, que são partições horizontais distribuídas entre diferentes servidores. Cada shard armazena uma parte do volume total de dados e é gerenciado de forma autônoma pelo MongoDB, sendo responsável por processar operações de leitura e escrita relacionadas aos dados sob sua responsabilidade. Cada shard está associado a um **conjunto de réplicas** (replica set), composto por servidores secundários que mantêm cópias atualizadas dos dados do shard principal. Em caso de falha de um shard primário, uma réplica assume automaticamente suas funções, garantindo a continuidade da operação e a alta disponibilidade do sistema. As operações de escrita e leitura com consistência garantida são realizadas diretamente nos shards primários, enquanto as réplicas podem atender a leituras secundárias, conforme necessário. O sistema também conta com **servidores de configuração**, que mantêm o mapeamento de quais dados pertencem a quais shards, e com **roteadores (mongos)**, responsáveis por direcionar as requisições de forma eficiente entre os diferentes shards. Assim, o mecanismo de sharding no MongoDB proporciona escalabilidade horizontal, resiliência a falhas e desempenho otimizado para grandes volumes de dados em ambientes distribuídos.

2.1.2 utilização

O primeiro passo para iniciar a utilização do MongoDB é a instalação do banco de dados do ambiente, que esta disponível para download na pagina oficial do MongoDB4. Por padrão o MongoDB utiliza o caminho ***/data/db***, mas este caminho não é criado automaticamente pelo banco. No diretório raiz onde foi extraído o MongoDB, crie a estrutura de pastas ***/data/db***. Opcionalmente, pode ser definido o caminho do banco na inicialização do MongoDB, passando o parâmetro --***dbpath*** na inicialização do mongodb.exe, mas isto só pode ser feito via prompt de comando. Depois de criada a estrutura de pastas necessária, deve-se executar o arquivo ***mongod.exe*** para iniciar o serviço do Mongo, e a janela vai ficar ativa na tela caso tenha sido iniciado com sucesso. Pode-se também abrir uma interface para administração dos dados pelo ***mongo.exe***, no qual se podem inserir bancos e ***collections***, assim como inserir documentos e buscar também.

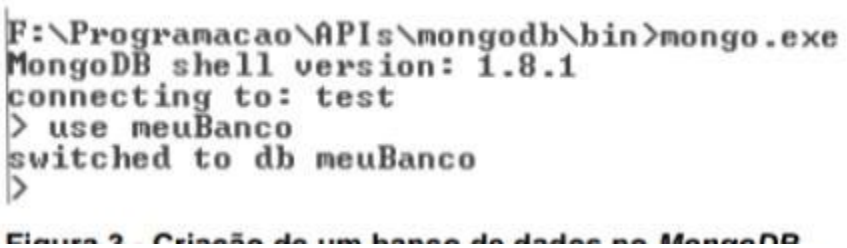

Figura 1 – Criação de um banco de dados no MongoDB

Para inserir um dado simples com MongoDB, deve-se primeiro ter o banco iniciado. Depois entre no mongo.exe, que é a interface para uso que vem por padrão no MongoDB. Como visto na Figura 1, o comando “use **meuBanco**”,

troca para o **db** do *test*(que é o padrão), para o “**meuBanco**”. O Banco “**meuBanco**” não é automaticamente criado neste momento, e sim no momento que o primeiro dado é inserido no banco. O MongoDB não tem tabelas, como um banco de dados convencional. Os dados são inseridos em documentos, e esses documentos são guardados em collections. As collections são criadas automaticamente no momento da primeira inserção de documento nela.

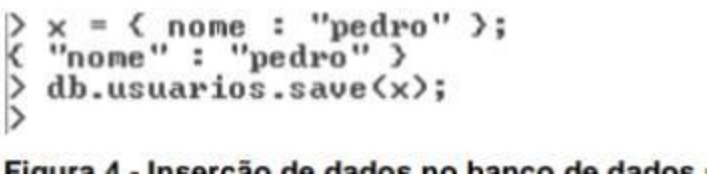

Figura 2 – Inserção de dados no banco de dados selecionado

Como pode ser visto na Figura 2, foi criado um documento com nome “x”, na qual foi adicionado um atributo “nome” com o valor “**pedro**”. Neste caso, x vai ser o documento a ser inserido no banco. Ao executar o comando **“db.usuarios.save(x);”** , como é a primeira inserção no banco “**meuBanco**”, vai ser criado automaticamente o banco “**meuBanco**” e a collection “**usuarios**”, e o documento “x” é inserido na *collection* “**usuarios**”. Ao executar a inserção, o prompt não mostra que foi inserido, então para ter certeza que foi inserido, uma busca deve ser feita, que retornara os dados do documento “x” que foi inserido. Para executar uma busca simples em todos os dados da *collection* “**usuarios**” , no prompt do MongoDB digite ***“db.usuarios.find();”***

Figura 3 – Busca com o método *Find*

Como visto na Figura 3, o comando find sem receber parâmetros, busca por todos os dados na *collection* da qual foi chamado, no caso a *collection* “usuarios”. Antes na inserção, somente o dado de “nome” havia sido passado, mas agora ao buscar o banco retornou dados de “nome” e “_id”, isto porque o próprio MongoDB gera um id automaticamente nas inserções. Caso o id esteja preenchido no documento na hora de salvar, ele salva por cima do documento correspondente ao id, fazendo uma operação de update. Deve-se tomar cuidado com inserções com id preenchido, pois o documento anterior com id não pode ser recuperado após uma operação de *save* que passa o id preenchido. Há casos, em que uma busca de documento com base no id precisará ser feita, para este caso pode-se usar o método “*findOne*” do MongoDB, ou simplesmente usando o próprio método find, passando para ele um documento com o “_id” preenchido. A diferença nestes dois casos, é que o método find, pode retornar mais de um resultado e o método *findOne* retorna obrigatoriamente um resultado somente. O uso apropriado destes métodos deve ser previamente Estudado para melhor aproveitamento do MongoDB em cada situação. Há vários casos também, em que o são necessários buscas mais precisas, Passando por parâmetro campos que não é o id, por exemplo, uma busca de todos Os usuários que contenham o nome “**pedro**”. Para estes casos, podem ser feitas *Querys*, que junto com o método find filtram os resultados encontrados de acordo Com a Query que foi feita.

Figura 4 – Query com documento explícito

Utilizando-se de *querys* no MongoDB, várias pesquisas ricas podem ser feitas Sem utilização de SQL, de maneira muito facilitada e rápida. Podem ser criados Documentos complexos e passando como parâmetro para o método find. O Método mais simples de criar *querys* é passar direto ao método find, os campos. Mas pode-se criar *querys* também simplesmente criando um documento e passando Ao método find como parâmetro. Podem-se ordenar os resultados usando *sort*, e passando para o *sort* o Valor a ser ordenado como parâmetro. Exemplo: ***"db.usuarios.find().sort({nome:1})"*** Usando **$gt** pode-se pesquisar por todos que sejam Maior que. Exemplo: ***"db.usuarios.find({idade: {$gt:33}})"*** pesquisa por todos na *collection* Usuários que tenham idade maior que 33. Usando $ne pode-se pesquisar por todos excluindo algum resultado da Busca. Exemplo: ***"db.usuarios.find({idade: {$ne: 33}})"*** pesquisa por Todos na *collection* usuários menos pelos que tenham idade igual a 33. Usando o caractere "/" antes ou depois da palavra a ser passada como Parâmetro a um método find, pode pesquisar por partes especificas dentro de uma Query, semelhante ao comando *like*, que a grande maioria dos bancos de dados Relacionais tem. Exemplo: "***db.usuarios.find({nome: /Joao/})" faz uma Query no MongoDB equivalente a "select * from usuários where nome = '%Joao%'"*** em query SQL relacional. Com uso de *Limit* é possível limitar o número de resultados de acordo com Sua necessidade. Por exemplo: **"db.usuarios.find().limit(10)"** pesquisa Todos na *collection* usuários mas retorna somente 10 resultados. Se usado o comando/método *Skip* pode-se definir quantos resultados serão Pesquisados além de limitar seus resultados, muito útil para ser utilizado em Pesquisar dinâmicas. Por exemplo: ***"db.usuarios.find().limit(10).skip(20)"*** vai pesquisar somente ate 20

Documentos, e retornar os 10 primeiros; Pará saber o número de documentos adicionados a uma ***collection***, pode ser Usado o método *count*. Por exemplo: ***"db.usuarios.count()"*** retorna o número De documentos adicionados a ***collection* usuarios**. Pode-se ainda utilizar o *count* junto com outros métodos na *query*, por Exemplo : ***"db.usuarios.find({idade: {$gt: 33}}).count()"*** retorna o Número de usuários que tem idade com maior que 33. SANTOS, Raduan Silva dos (2011).

2.2 NoSQL

Segundo a Amazon Web Services (2024), os bancos de dados NoSQL são sistemas de armazenamento de dados não relacionais, projetados para lidar com grandes volumes de dados de forma escalável, flexível e com alta performance. Diferentemente dos bancos de dados relacionais, os NoSQL não utilizam um esquema fixo, permitindo que os dados sejam organizados em modelos variados, como documentos, chave-valor, grafos e colunas. Esse modelo é ideal para aplicações modernas que exigem alta disponibilidade, agilidade no desenvolvimento e escalabilidade eficiente. O termo NoSQL surgiu em 1998, criado por Carlo Strozzi, porém passou a ser amplamente utilizado a partir de 2009, para descrever bancos de dados que não seguem o modelo relacional tradicional. Esses bancos de dados se destacam pela facilidade de desenvolvimento, pela capacidade de entregar alta performance em ambientes de grande escala e pelo seu custo-benefício, tornando-se uma solução amplamente adotada em aplicações web, mobile, Internet das Coisas (IoT) e sistemas distribuídos.

- Chave-valor: armazena os dados de forma muito semelhante a um Map *(java.util.Map*) do Java, assim armazena-se a chave e o valor que pode ser qualquer informação.
- Orientado a Documentos: armazena os dados em forma de uma estrutura de dados composta de um número variado de componentes com tipos de dados diversos, da mesma forma que é feito com um arquivo XML ou JSON.

- Família de colunas (semelhante ao *BigTable*): se tornou popular através do *BigTable* do Google, que foi publicado no ano de 2006.
- Grafo: armazenam os dados de forma semelhante à estrutura de dados De uma rede social, onde os dados se ligam aos outros, de forma a Criar uma estrutura muito facilmente navegável.

2.2.1 Vantagens de se utilizar NoSQL

Os bancos de dados NoSQL oferecem uma série de vantagens que os tornam altamente recomendados para aplicações que lidam com grandes volumes de dados e que demandam flexibilidade e escalabilidade. Entre seus principais benefícios, destacam-se:

- **Flexibilidade**: Possuem esquemas dinâmicos, permitindo que os dados sejam armazenados de maneira mais livre, sem a rigidez de um modelo relacional. Isso facilita a manipulação de dados semiestruturados e não estruturados, além de possibilitar um desenvolvimento mais ágil e iterativo.

- **Escalabilidade**: São projetados para escalabilidade horizontal, ou seja, a capacidade de distribuir dados e cargas de trabalho entre vários servidores ou nós. Esse modelo permite que as aplicações cresçam de forma eficiente, sem depender de servidores de alto custo.

- **Alto desempenho:** Por serem otimizados para modelos de dados específicos e padrões de acesso definidos, os bancos NoSQL oferecem desempenho superior em operações de leitura e escrita, especialmente em cenários de grandes volumes de dados e acesso simultâneo.

- **Alta funcionalidade:** Fornecem APIs robustas e modelos de dados adequados para diferentes tipos de necessidades, como chave-valor, documentos, grafos e colunas, permitindo maior aderência às particularidades de cada aplicação.

  .

Segundo a Amazon Web Services (2024), essas características tornam os bancos de dados NoSQL ideais para atender às demandas de aplicações modernas, que exigem alta disponibilidade, desempenho e flexibilidade no tratamento de dados.

2.3 Orientado a Documentos

Segundo o site da AWS, um **banco de dados orientado a documentos** é um tipo de banco de dados **NoSQL** projetado para armazenar, consultar e gerenciar dados na forma de documentos semiestruturados, geralmente no formato **JSON, BSON ou XML**. Cada documento é uma unidade independente que contém dados e a sua própria estrutura, permitindo que diferentes documentos na mesma coleção tenham esquemas distintos. Isso oferece **flexibilidade para armazenar dados complexos**, como objetos aninhados e listas, sem a necessidade de um esquema fixo, como acontece nos bancos de dados relacionais. Os bancos de dados orientados a documentos são ideais para aplicações que exigem rápida iteração e desenvolvimento ágil, pois possibilitam alterações no modelo de dados sem a necessidade de migração estrutural. Além disso, esses bancos oferecem escalabilidade horizontal e alto desempenho em consultas baseadas em documentos.

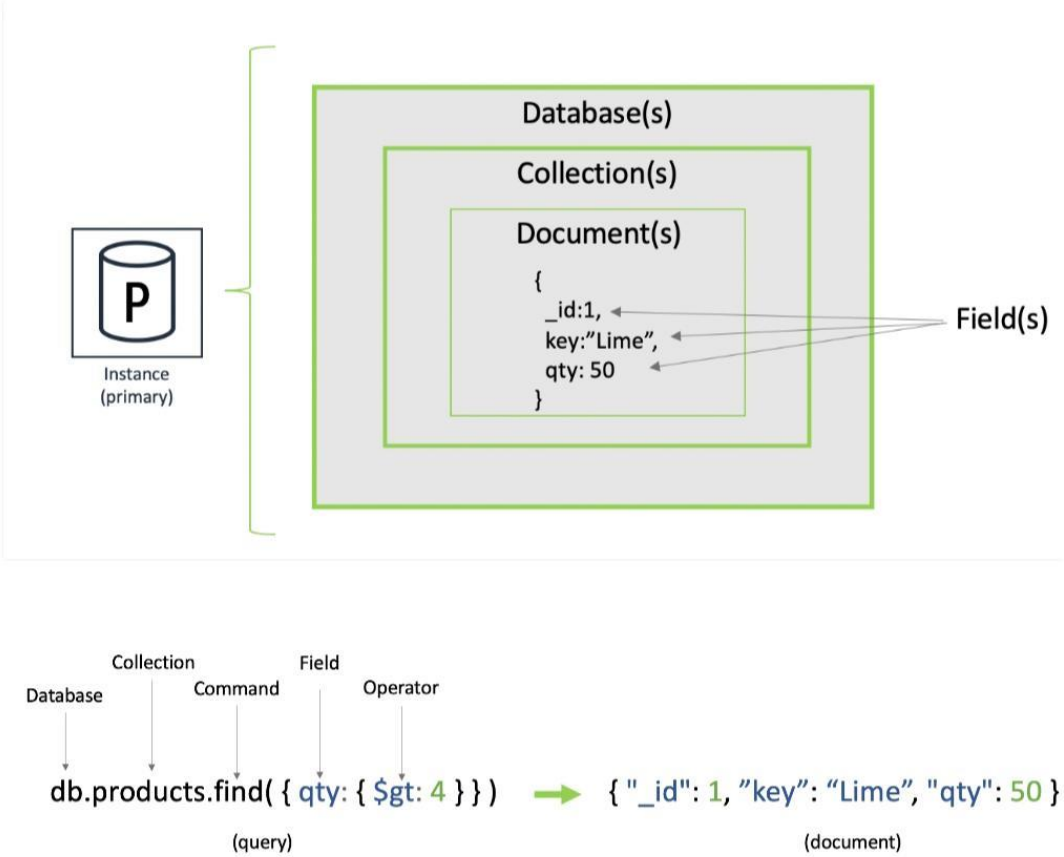

Figura 5 – Exemplo de Orientado a Documentos

2.3.1 Vantagens de utilizar banco de dados orientado a documentos :

De acordo com a Amazon Web Services (AWS, 2024), os bancos de dados orientados a documentos apresentam diversas vantagens em relação aos modelos tradicionais. Uma das principais características é a **flexibilidade de esquema**, que permite armazenar documentos com estruturas variadas, sem a obrigatoriedade de seguir um modelo fixo, o que facilita a adaptação a mudanças nos requisitos dos dados. Outra vantagem significativa é a **facilidade no desenvolvimento**, visto que os documentos são estruturados em formatos como JSON, BSON ou XML, que possuem grande similaridade com os objetos

utilizados nas linguagens de programação modernas. Isso reduz a complexidade na conversão dos dados entre a aplicação e o banco, tornando o desenvolvimento mais ágil e eficiente. Além disso, esses bancos oferecem **escalabilidade horizontal**, possibilitando a distribuição dos dados em múltiplos servidores, o que garante maior capacidade de armazenamento e desempenho conforme o crescimento da aplicação. Destaca-se também o **alto desempenho nas consultas**, especialmente em dados complexos e aninhados, visto que não há necessidade de operações de junção (JOIN) como nos bancos de dados relacionais. Os bancos orientados a documentos ainda proporcionam **alta disponibilidade e tolerância a falhas**, por meio de mecanismos de replicação automática e recuperação em caso de falhas, assegurando a continuidade do acesso aos dados. Por fim, sua estrutura é especialmente adequada para aplicações modernas, como catálogos de produtos, gerenciamento de conteúdos, sistemas de Internet das Coisas (IoT) e aplicações móveis, onde os dados são altamente dinâmicos e frequentemente alterados (AWS, 2024).

2.3.2 Casos de uso de bandos de dados a documentos

De acordo com a AWS (2024), os bancos de dados orientados a documentos são especialmente adequados para determinados cenários devido à sua flexibilidade de esquema, capacidade de armazenamento de dados semiestruturados e escalabilidade horizontal. Abaixo, alguns dos principais casos de uso:

- **Gerenciamento de conteúdo**
    - Ideal para plataformas como blogs, portais de notícias e serviços de vídeo. Cada entidade da aplicação pode ser representada por um único documento, facilitando a atualização de dados sem necessidade de alterar o esquema de dados da base ou interromper o serviço.
- **Catálogos de produtos (e-commerce)**

  - Em sistemas de comércio eletrônico, produtos frequentemente possuem atributos variados. Com bancos de dados orientados a documentos, cada produto pode ser descrito em um único documento contendo apenas os atributos relevantes, suportando consultas rápidas e mantendo alto desempenho.
- **Gerenciamento de sensores (IoT)**
  - Aplicações que coletam dados de sensores se beneficiam da natureza semiestruturada e flexível destes bancos, já que os documentos podem evoluir com o tempo para acomodar novas métricas ou formatos .
- **Perfis de usuários e aplicações com dados heterogêneos**
  - Em sistemas com milhões de usuários cujos perfis contêm diferentes tipos de informação, a flexibilidade de documento e a escalabilidade horizontal permitem armazenar dados variados sem duplicação estrutural.

2.3.3 Como Funcionam os Bancos de Dados Orientado a Documentos

Os bancos de dados de documentos armazenam dados como pares de valores-chave no formato JSON. Você pode ler e gravar documentos JSON nos bancos de dados programaticamente.

Estruturas de Documentos JSON

Valor chave

Os pares de valores-chave são registrados entre colchetes. A chave é uma string e o valor pode ser qualquer tipo de dado, como inteiro, decimal ou booleano. Por exemplo, um valor-chave simples é {“year”: 2013}.

Matriz

Uma matriz é uma coleção ordenada de valores definidos entre colchetes esquerdo ([) e direito (]). Os itens na matriz são separados por vírgula. Por exemplo, {“fruit”: [“apple”, “mango”]}.

Objetos

Um objeto é uma coleção de pares de valores-chave. Essencialmente, os documentos JSON permitem que os desenvolvedores incorporem objetos e criem pares aninhados. Por exemplo, {“address”: {“country”: “USA”, “state”: “Texas”}}.

Exemplo de documentos JSON

No exemplo a seguir, um documento semelhante a JSON descreve um conjunto de dados de filme.

[

{

“year” : 2013,

“title” : “Turn It Down, Or Else!”,

“info” : {

“directors” : [“Alice Smith”, “Bob Jones”],

“release_date” : “2013-01-18T00:00:00Z”,

“rating” : 6.2,

“genres” : [“Comedy”, “Drama”],

“image_url” : http://ia.media-imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg,

```
            “plot” : “A rock band plays their music at high volumes, annoying the neighbors.”,
            “actors” : [“David Matthewman”, “Jonathan G. Neff”]
        }
    },
    {
        “year”: 2015,
        “title”: “The Big New Movie”,
        “info”: {
            “plot”: “Nothing happens at all.”,
            “rating”: 0
        }
    }
]
```


Você pode observar que o documento JSON contém valores, matrizes e objetos simples de forma bastante flexível. Você pode até mesmo ter uma matriz com objetos JSON dentro dela. Portanto, bancos de dados orientados a documentos permitem criar uma hierarquia de nível ilimitado de objetos JSON incorporados. Depende inteiramente de você qual esquema você deseja fornecer ao seu armazenamento de documentos.

2.4 Scrums

Segundo o artigo de Raduan Silva dos Santos, o Scrum é definido como um framework utilizado para desenvolvimento e manutenção de produtos complexos, especialmente software. O autor, com base em Ken Schwaber (1995), destaca que o Scrum não é uma metodologia rígida, mas sim um conjunto de práticas e regras que orientam o desenvolvimento ágil**, com foco na adaptação constante e na entrega incremental de valor.

O Scrum é baseado no conceito de processo empírico, que se apoia em três pilares fundamentais:

- **Transparência**: os aspectos importantes do processo devem ser visíveis a todos os envolvidos, garantindo compreensão e alinhamento comum sobre o trabalho.
- **Inspeção**: os usuários do Scrum devem inspecionar frequentemente os artefatos e o progresso do projeto para identificar possíveis desvios.
- **Adaptação**: sempre que algo estiver fora dos padrões aceitáveis, deve-se fazer ajustes imediatos no processo ou no produto, para garantir que o projeto continue no rumo certo.

Papéis no Scrum

Product Owner: responsável por maximizar o valor do produto e do trabalho da equipe, além de gerenciar o backlog do produto, priorizando os itens.

Scrum Master: garante que o Scrum seja compreendido e aplicado corretamente, atuando como facilitador e removendo impedimentos que possam atrapalhar o time.

***Development Team* (Time de Desenvolvimento)**:equipe auto-organizada e multifuncional, que é responsável por entregar os incrementos do produto de acordo com as prioridades estabelecidas.

Ciclos de Scrum

O Scrum organiza o trabalho em ciclos curtos e bem definidos, chamados de Sprints, que normalmente duram de duas a quatro semanas. Cada Sprint gera uma versão potencialmente utilizável do produto.

Dentro desse ciclo ocorrem eventos importantes, como:

- *Sprint Planning* (Planejamento da Sprint)
- *Daily Scrum* (Reuniões diárias)Sprint Review (Revisão da Sprint)
- *Sprint Retrospective* (Retrospectiva da Sprint)

Figura 6 – Ciclo de Scrum

Quadro de Scrum

Durante o desenvolvimento, os requisitos são organizados em um Quadro de Scrum, que permite acompanhar visualmente o progresso dos itens. Os requisitos são representados em cartões (geralmente post-its) movidos entre colunas que representam os estados do trabalho, como:

- Aguardando
- Em Desenvolvimento
- Em Teste
- Concluído

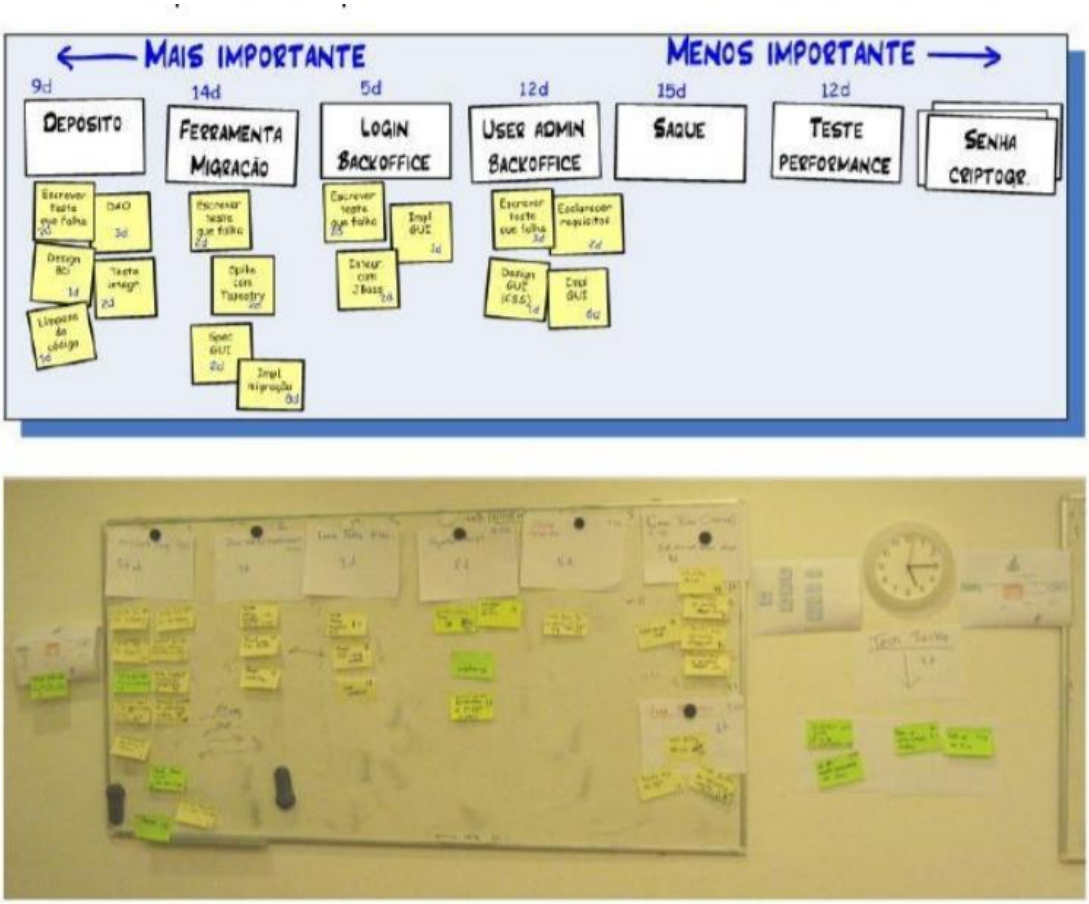

Figura 7 – Exemplo de Quadro de Scrums

3. Resultados obtidos

O trabalho desenvolvido por Raduan Silva dos Santos, intitulado “Desenvolvimento de Aplicação para o Controle de Scrum com MongoDB”, consiste na elaboração de uma aplicação web destinada ao gerenciamento de projetos que utilizam a metodologia ágil Scrum. Este trabalho foi realizado como requisito parcial para a obtenção do título de Tecnólogo no curso de Análise e Desenvolvimento de Sistemas pela Universidade Tecnológica Federal do Paraná (UTFPR), campus Medianeira.

O autor propôs-se a desenvolver uma solução capaz de atender às demandas de equipes que trabalham com Scrum, oferecendo um sistema que auxilie no acompanhamento e controle das etapas de desenvolvimento de projetos, de forma prática, intuitiva e eficiente. Um dos grandes diferenciais deste projeto reside na escolha da tecnologia de banco de dados utilizada: o MongoDB, um banco de dados NoSQL, orientado a documentos, que proporciona maior flexibilidade, escalabilidade e desempenho em comparação aos tradicionais bancos relacionais.

Na primeira etapa do trabalho, Raduan realiza uma revisão bibliográfica aprofundada, abordando os principais conceitos relacionados ao Scrum, às bases de dados NoSQL e às ferramentas e frameworks utilizados no desenvolvimento da aplicação. O Scrum é apresentado como um framework de desenvolvimento ágil que permite melhor adaptação às constantes mudanças durante o ciclo de desenvolvimento, favorecendo a transparência, a inspeção e a adaptação, pilares fundamentais desse modelo.

No contexto tecnológico, o autor opta pelo MongoDB devido às suas características de armazenamento flexível, sem a necessidade de um esquema fixo, e pela capacidade de lidar com grandes volumes de dados de maneira distribuída e eficiente. O MongoDB, por ser um banco orientado a documentos, armazena os dados em estruturas JSON, conhecidas como BSON (Binary JSON), facilitando a manipulação dos dados no desenvolvimento de aplicações web. O autor explica, ainda, como a tecnologia de sharding, nativa do MongoDB,

permite a distribuição horizontal dos dados, garantindo alta disponibilidade e escalabilidade, aspectos essenciais para sistemas que necessitam suportar múltiplos usuários e grandes quantidades de informação.

Para viabilizar a integração entre a aplicação desenvolvida em Java e o banco de dados MongoDB, Raduan utilizou o framework Morphia, que tem como principal função realizar o mapeamento objeto-documento (ODM). Essa ferramenta permite que as classes Java sejam diretamente convertidas em documentos no banco, simplificando significativamente as operações de persistência e recuperação dos dados. O Morphia oferece ainda recursos como geração automática de identificadores, criação de índices e construção de consultas de maneira intuitiva, o que torna o desenvolvimento mais produtivo e organizado.

Outro recurso tecnológico de destaque utilizado no desenvolvimento foi o framework Google Guice, responsável pela injeção de dependências. Com ele, o autor organizou a arquitetura do sistema, reduzindo o acoplamento entre as classes e facilitando tanto a manutenção quanto a escalabilidade da aplicação. A combinação dessas tecnologias (MongoDB, Morphia e Google Guice) resultou em uma aplicação robusta, modular e eficiente, alinhada às boas práticas de desenvolvimento de software.

O sistema desenvolvido, denominado GPA (Gerenciador de Projetos Ágeis), permite que os usuários cadastrem projetos, produtos, stakeholders e requisitos, além de acompanhar todo o ciclo de desenvolvimento de um projeto ágil. Uma das funcionalidades centrais do sistema é o quadro Scrum virtual, que simula o quadro físico utilizado pelas equipes, organizando os requisitos em diferentes estados, como “Aguardando”, “Em Desenvolvimento”, “Em Testes” e “Concluído”. Essa funcionalidade facilita a visualização do progresso dos projetos e contribui para uma gestão mais eficiente das tarefas.

Na modelagem do banco de dados, o MongoDB mostrou-se altamente adequado, principalmente pela flexibilidade na manipulação dos dados e pela ausência da necessidade de esquemas rígidos, característica que se alinha perfeitamente com a dinâmica de projetos ágeis, onde alterações e ajustes são

constantes. Além disso, a utilização do MongoDB proporcionou uma melhoria no desempenho, especialmente em operações de leitura e escrita, uma vez que o banco é otimizado para trabalhar com grandes volumes de dados e com estruturas não normalizadas. O autor também discute os desafios encontrados, como a necessidade de conversão entre os objetos utilizados na parte cliente da aplicação, desenvolvida com Google Web Toolkit (GWT), e os objetos persistidos no MongoDB. Essa integração exigiu a adoção de padrões como Value Objects (VO) para garantir que os dados trafegassem corretamente entre cliente e servidor, superando incompatibilidades, especialmente no tratamento dos identificadores do MongoDB.

Ao concluir o trabalho, Raduan destaca que o sistema atendeu plenamente aos objetivos propostos, demonstrando que o uso do MongoDB em conjunto com frameworks como Morphia e Guice é não apenas viável, como altamente recomendado para aplicações que exigem agilidade, flexibilidade e escalabilidade. Além disso, o sistema desenvolvido se apresenta como uma alternativa gratuita e acessível em comparação às ferramentas de mercado, que muitas vezes possuem custos elevados e não são focadas exclusivamente na metodologia Scrum. O trabalho contribui significativamente para demonstrar, na prática, como as tecnologias emergentes, como os bancos NoSQL, podem ser aplicadas de forma eficiente no desenvolvimento de soluções para gerenciamento ágil de projetos. Ele serve como um excelente exemplo de integração entre práticas ágeis e tecnologias modernas, reforçando a importância da adoção de ferramentas adequadas para apoiar metodologias que exigem adaptação constante e rápida resposta às mudanças.

3.1 Análise e Projeto do Sistema

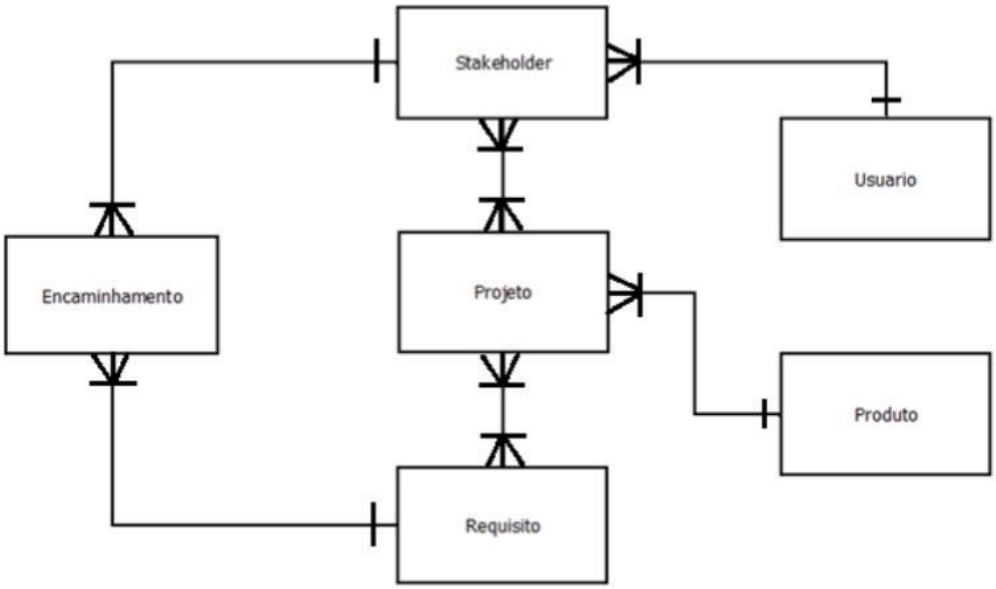

Figura 8 – MER da Aplicação

O cadastro do usuário tem objetivo de salvar os dados para login, nome e informações relevantes a usuários do sistema. A *collection* que guardará os dados tem nome de “usuários”, e seus campos serão os seguintes:

- Id – campo que guardará o ID da *collection* usuarios. Seu tipo será *ObjectID*, que é uma classe da API do driver Java do MongoDB que tem objetivo de guardar as informações de id, e gerar automaticamente este valor para novos registros.

- *Login* – campo que combinado com uma senha, dará acesso para o usuário ao sistema. Seu tipo é *String* e seu valor é único em todo sistema, ou seja, não terão cadastrados dois usuários com o mesmo *login*.

- Senha – campo que combinado com um, dará acesso para o usuário ao sistema. Seu tipo é *String*.

- Nome – campo de tipo *String*, contendo informações de nome do usuário.

- Administrador – campo boolean que tem objetivo de dizer se o usuário tem permissões de administrador. Somente usuários administradores poderão cadastrar novos projetos, e pesquisar por todos os projetos.

O cadastro de stakeholder tem por objetivo cadastrar os usuários, que serão diretamente envolvidos em cada projeto. Os campos envolvidos na *collection Stakeholders* são:

- Id – Campo de tipo ObjectID, que é controlado pelo MongoDB, e pode Ser usado para buscas, pois é o identificador único da *collection Stakeholders*.

- Nome – Campo do tipo *String* que guardará dados do nome do *Stakeholder*.

- Papel *Stakeholder* – Campo que indica qual será o papel realizado pelo *Stakeholder* no projeto. Os papeis de *stakeholder* que serão disponíveis no sistema serão: dono, gerente de projeto, fornecedor de requisitos, desenvolvedor, tester, analista de sistemas e analista de negócios.

- Usuário – é ligado a *collection* “usuarios”. Guarda informação de qual usuário é responsável por este stakeholder.

O cadastro de produto tem intuito simples de guardar informações sobre um produto, o qual será referenciado no cadastro de projetos. Um produto pode ter vários projetos sendo executados sobre ele, e obrigatoriamente todo projeto deve Ser executado para um produto. Inicialmente, o cadastro de produtos terá somente Dois campos, o Id e o Nome. O cadastro de projeto é à base de todo sistema. A partir do cadastro de projeto, produtos serão relacionados, os stakeholders existentes serão selecionados, E o quadro de scrum será gerado. os campos do cadastro de projeto são os Seguintes:

- Id – identificador único da *collection* “projetos”. Seu tipo é *ObjectID* Tal como os Ids das outras collections do sistema.

- Nome – campos *String*, único em todo sistema, ou seja, não poderão ser cadastrados dois projetos com o mesmo nome.

- Data Início – campo do tipo Date, que guarda a data em que o projeto terá início.

- Data Fim – campo do tipo Date, que guarda a data em que o projeto será finalizado.

- Stakeholders – campo do tipo List, que guarda todos os stakeholders envolvidos no projeto.

- Produto – campo ligado a collection “produtos” que indica qual Produto esta sendo criado/modificado pelo projeto.

A partir do cadastro de projeto um novo cadastro será habilitado, o cadastro de requisito. Os requisitos serão os passos que os stakeholders terão que tomar para a realização do projeto, e cada requisito estará ligado a um projeto, e também guardará os dados de encaminhamentos a serem. Os dados da *collection* “requisitos” serão:

- Id – identificador único do cadastro de requisito. Seu tipo é ObjectID eo identificador único da collection “requisitos”.

- Titulo – campo String que guarda o titulo do projeto. O título do projeto é usado para identificar o projeto na lista de projetos do usuário, na tela principal, e é usado na busca.

- Descrição – campo String que guarda informações sobre o projeto.

- Prioridade Requisito – campo que guarda informações referentes à prioridade em que o requisito deve ser implementado. A partir deste Dado, a cor do “post-it” muda. As prioridades requisito presentes no sistema serão Alto, Médio e baixo, posteriormente podendo haver outros requisitos implementados. Seu tipo é um *Enum*.

- Tempo Estimado – campo do tipo Integer que guarda o tempo estimado para desenvolvimento do requisito, em horas.

- Encaminhamentos – campo do tipo List, que guarda os Encaminhamentos do requisito. No momento em que o requisito é cadastrado no sistema, um requisito padrão é cadastrado.

- Data Cadastro – campo do tipo Date, que guarda a data em que o requisito foi cadastrado no sistema. É preenchido automaticamente pelo sistema.

- Tempo Total – campo do tipo Integer, que indica o tempo total utilizado para realização do requisito, em horas.

- Projeto – campo ligado a collection “projetos” que indica o projeto que terá este requisito implementado.

A partir do requisito, os encaminhamentos serão gerados. Encaminhamentos indicarão em que estado anda o desenvolvimento de cada requisito, assim possibilitando o controle básico do quadro de scrum.

Todo encaminhamento terá um campo chamado StatusRequisito, que indica em qual estado o requisito se encontra atualmente. O campo StatusRequisito é um *enum* e os estados disponíveis no sistema são:

- Aguardando – indica que o requisito ainda não foi escolhido por nenhum stakeholder para nenhuma tarefa.
- Desenvolvendo – indica que o requisito se encontra em desenvolvimento.
- Testando – indica que o desenvolvimento do requisito já foi concluído, e testes estão sendo feitos sobre aquele requisito.
- Concluído – indica que o requisito já foi desenvolvido e testado, e passou nos testes.

A collection de encaminhamentos conta ainda com outros campos usados para controle e melhor detalhamento do estado em que o requisito se encontra. são eles:

- Id – índice único da collection “Encaminhamentos”.

- Stakeholder - ligado a collection “stakeholders”, guarda a informação sobre qual é o stakeholder responsável pelo encaminhamento, no presente StatusRequisito.
- Data – campo do tipo Date que indica qual a data em que o encaminhamento foi criado.
- Descrição – campo do tipo String que armazena informações sobre o encaminhamento.

No sistema, cada stakeholder estará ligado a um usuário, mas o usuário não vai estar ligado a nenhum stakeholder, de modo que um único usuário possa estar em mais de um stakeholder. Cada projeto vai ter um produto, mas o produto não vai ter adicionado internamente o projeto, cada projeto tem também um ou mais stakeholders e um ou mais requisitos adicionados. Cada requisito terá um ou mais encaminhamentos, e cada encaminhamento terá obrigatoriamente ligação com um Stakeholder.

As Classes geradas na implementação do sistema foram muitas, tornando muito difícil a geração de um diagrama de classes do sistema, por este motivo será somente citadas as classes de acordo com seus pacote correspondente. As classes Internas não serão citadas.

No pacote com.geekvigarista.scrummanager.server.actionhanders foram Implementadas as classes necessárias para a ligação com a parte cliente da Aplicação:

- cadastros.encaminhamento.ExcluirEncaminhamentoActionHandlEr

- cadastros.encaminhamento.SalvarEncaminhamentoActionHandleR

- cadastros.produto.BuscarTodosProdutosActionHandler

- cadastros.produto.LoadProdutoActionHandler
- cadastros.produto.SalvarProdutoActionHandler
- cadastros.projeto.BuscarProjetoLikeActionHandler
- cadastros.projeto.BuscarProjetosByUsuarioActionHandler
- cadastros.projeto.CarregarRequisitosNoProjetoActionHandler
- cadastros.projeto.LoadProjetoActionHandler
- cadastros.projeto.SalvarProjetoActionHandler
- cadastros.requisito.ExcluirRequisitoActionHandler
- cadastros.requisito.LoadRequisitoActionHandler
- cadastros.requisito.SalvarRequisitoActionHandler
- cadastros.stakeholder.BuscarStakeholderActionHandler
- cadastros.stakeholder.ExcluirStakeholderActionHandler

- cadastros.stakeholder.LoadStakeholderActionHandler

- cadastros.stakeholder.SalvarStakeholderActionHandler

- cadastros.usuario.BuscarUsuarioActionHandler

- cadastros.usuario.ExcluirUsuarioActionHandler

- cadastros.usuario.LoadUsuarioActionHandler

- cadastros.usuario.login.LoginHandler

- cadastros.usuario.login.LogoutHandler

- cadastros.usuario.login.VerificaUsuarioLogadoHandler

- cadastros.usuario.SalvarUsuarioActionHandler

No pacote com.geekvigarista.scrummanager.server.beans foram implementadas os beans com anotações do Morphia, usados na persistência:

- EncaminhamentoPOJO
- ProdutoPOJO
- ProjetoPOJO

- RequisitoPOJO
- StakeholderPOJO
- UsuarioPOJO

No pacote com.geekvigarista.scrummanager.server.filters somente a classe CacheFilter foi necessária.

No pacote com.geekvigarista.scrummanager.server.guice foram Implementadas as classes responsáveis pela configuração da injeção de Dependência na parte servidor:

- DAOModule

- DispatchServletModule

- GuiceServletConfig

- ServerModule

No pacote com.geekvigarista.scrummanager.server.interfaces foram armazenadas as interfaces usadas pelo DAO:

- dao.IDao
- dao.IDaoEncaminhamento
- dao.IDaoProduto
- dao.IDaoProjeto
- dao.IDaoRequisito
- dao.IDaoStakeholder

- dao.IDaoUsuario

No pacote com.geekvigarista.scrummanager.server.persistencia as classes responsáveis efetivamente pela persistência foram salvas:

- dao.DaoEncaminhamento
- dao.DaoProduto
- dao.DaoProjeto
- dao.DaoRequisito
- dao.DaoStakeholder
- dao.DaoUsuario
- utils.MongoConnection

Já no pacote com.geekvigarista.scrummanager.shared foram armazenadas todas as classes usadas tanto na parte servidor quanto na parte cliente da aplicação:

- commands.encaminhamento.excluir.ExcluirEncaminhamentoAction
- commands.encaminhamento.excluir.ExcluirEncaminhamentoResult
- commands.encaminhamento.salvar.SalvarEncaminhamentoAction
- commands.encaminhamento.salvar.SalvarEncaminhamentoResult
- commands.produto.busca.BuscarProdutoListResult
- commands.produto.busca.BuscaTodosProdutosAction

- commands.produto.load.LoadProdutoAction
- commands.produto.load.LoadProdutoResult
- commands.produto.salvar.SalvarProdutoAction
- commands.produto.salvar.SalvarProdutoResult
- commands.projeto.load.BuscarProjetoLikeAction
- commands.projeto.load.BuscarProjetoListResult
- commands.projeto.load.BuscarProjetosByUsuarioAction
- commands.projeto.load.CarregarRequisitosNoProjetoAction
- commands.projeto.load.LoadProjetoAction
- commands.projeto.load.LoadProjetoResult
- commands.projeto.salvar.SalvarProjetoAction
- commands.projeto.salvar.SalvarProjetoResult

- commands.requisito.buscar.BuscarRequisitoByIdAction
- commands.requisito.buscar.BuscarRequisitoObjResult
- commands.requisito.excluir.ExcluirRequisitoAction
- commands.requisito.excluir.ExcluirRequisitoResult
- commands.requisito.salvar.SalvarRequisitoAction
- commands.requisito.salvar.SalvarRequisitoResult
- commands.stakeholder.buscar.BuscarStakeholderAction
- commands.stakeholder.buscar.BuscarStakeholderByIdAction
- commands.stakeholder.buscar.BuscarStakeholderListResult
- commands.stakeholder.buscar.BuscarStakeholderObjResult
- commands.stakeholder.excluir.ExcluirStakeholderAction
- commands.stakeholder.excluir.ExcluirStakeholderResult

- commands.stakeholder.salvar.SalvarStakeholderAction
- commands.stakeholder.salvar.SalvarStakeholderResult
- commands.usuario.buscar.BuscarUsuarioAction
- commands.usuario.buscar.BuscarUsuarioByIdAction
- commands.usuario.buscar.BuscarUsuarioListResult
- commands.usuario.buscar.BuscarUsuarioObjResult
- commands.usuario.excluir.ExcluirUsuarioAction
- commands.usuario.excluir.ExcluirUsuarioResult
- commands.usuario.login.LoginUsuarioAction
- commands.usuario.login.LogoutUsuarioAction
- commands.usuario.login.VerificaUsuarioLogadoAction
- commands.usuario.salvar.SalvarUsuarioAction

- commands.usuario.salvar.SalvarUsuarioResult
- dtos.ProjetoStakeholderDTO
- enums.AcaoEncaminhar
- enums.PapelStakeholder
- enums.PrioridadeRequisito
- enums.StatusRequisito
- utils.EncaminharUtil
- vos.Encaminhamento
- vos.Produto
- vos.Projeto
- vos.Requisito
- vos.Stakeholder

- vos.Usuario

3.2 Ligação Entre o Banco de Dados ea Aplicação

O banco de dados adotado na aplicação não segue o modelo relacional tradicional, ou seja, não utiliza estruturas baseadas em tabelas, registros ou schemas fixos. A tecnologia escolhida foi o **MongoDB**, uma das principais soluções disponíveis no universo dos bancos **NoSQL**, cuja seleção se justifica pela facilidade de compreensão do seu modelo orientado a documentos. Para possibilitar a comunicação entre o MongoDB e a aplicação desenvolvida em Java, foi utilizado um **driver específico**, disponibilizado diretamente no site oficial do MongoDB, em formato **JAR**. Com a inclusão desse JAR no projeto, o desenvolvedor passa a ter acesso a um conjunto de classes que simplifica significativamente a interação com o banco de dados, tornando o desenvolvimento mais prático e eficiente. Buscando otimizar ainda mais o processo de desenvolvimento e reduzir o tempo de codificação, optou-se pela utilização do **framework Morphia**, responsável pelo mapeamento dos objetos da aplicação diretamente para o MongoDB. Esse framework, também disponibilizado na forma de um JAR leve, permite o mapeamento automático dos objetos Java para documentos no banco de dados, facilitando a persistência dos dados. No desenvolvimento da interface cliente, foi empregada a linguagem **GWT (Google Web Toolkit)**. Para viabilizar a comunicação eficiente entre o cliente e o servidor, utilizou-se o framework **Google Guice**, que oferece suporte à **injeção de dependências**. Por meio desse recurso, é possível estabelecer a conexão entre as camadas da aplicação sem a necessidade de implementar manualmente os vínculos, minimizando erros e tornando o código mais limpo e modular.

3.3 implementação do método excluir

```
54      @Override
55      public boolean excluir(Projeto t)
56      {
57          try
58          {
59              this.deleteById(new ProjetoPOJO(t).getId());
60              return true;
61          }
62          catch(Exception e)
63          {
64              e.printStackTrace();
65              return false;
66          }
67      }
```

Figura 9 – Método Excluir

3.4 Implementação do método Buscar, BuscarTodos e ToValueObject

```
75      @Override
76      public Projeto buscar(String id)
77      {
78          ProjetoPOJO projPojo = this.findOne("id", new ObjectId(id));
79          if(projPojo == null)
80          {
81              return null;
82          }
83          return projPojo.getProjeto();
84      }
```

Figura 10- Método Buscar

```
69      @Override
70      public List<Projeto> buscarTodos()
71      {
72          return toValueObject(this.find());
73      }
```

Figura 11- Método BuscarTodos

```
158     @Override
159     public List<Projeto> toValueObject(QueryResults<ProjetoPOJO> resultadoBusca)
160     {
161         List<Projeto> retorno = new ArrayList<Projeto>();
162         for(ProjetoPOJO projPojo : resultadoBusca.asList())
163         {
164             retorno.add(projPojo.getProjeto());
165         }
166         return retorno;
167     }
```

Figura 12- Método *toValueObject*

3.5 Value Objects, POJOS e Anotações do Morphia

```
public class Produto implements Serializable
{
    private static final long serialVersionUID = 1L;

    private String id;
    private String descricao;

    public Produto(){}

    public Produto(String descricao)
    {
        super();
        this.descricao = descricao;
    }

    public Produto(String id, String descricao)
    {
        super();
        this.id = id;
        this.descricao = descricao;
    }
```

Figura 13- *value objecte* de produto

```
@Entity("produtos")
public class ProdutoPOJO
{
    @Id
    ObjectId id;
    private String descricao;

    public ProdutoPOJO()
    {
    }

    public ProdutoPOJO(Produto produto)
    {
        if(produto == null)
        {
            this.id = null;
            this.descricao = null;
        }
        else
        {
            if(produto.getId() != null)
            {
                this.id = new ObjectId(produto.getId());
            }
            this.descricao = produto.getDescricao();
        }
    }
```

Figura 13 – Produto Pojo

```
@Transient
private List<RequisitoPOJO> requisitos;

@Transient
private Projeto projeto;

public ProjetoPOJO()
{
}

/**
 * @param projeto
 */
public ProjetoPOJO(Projeto projeto)
{
    super();
    if(projeto == null)
        projeto = new Projeto();
    this.projeto = projeto;
    this.dataFim = projeto.getDataFim();
    this.dataInicio = projeto.getDataInicio();
    this.nome = projeto.getNome();
    this.produto = new ProdutoPOJO(projeto.getProduto());
    if(projeto.getRequisitos() != null)
    {
        this.requisitos = new ArrayList<RequisitoPOJO>();
        for(Requisito r : projeto.getRequisitos())
        {
            RequisitoPOJO rPojo = new RequisitoPOJO(r);
            this.requisitos.add(rPojo);
        }
    }
```

Figura 13- Objeto transiente para conversao de Pojo para *value object*

```
public Produto getProduto()
{
    if(this.id != null)
    {
        return new Produto(this.id.toString(), this.descricao);
    }
    else
    {
        return new Produto(this.descricao);
    }
}
```

Figura 14- Método de conversão de Pojo para *value object*

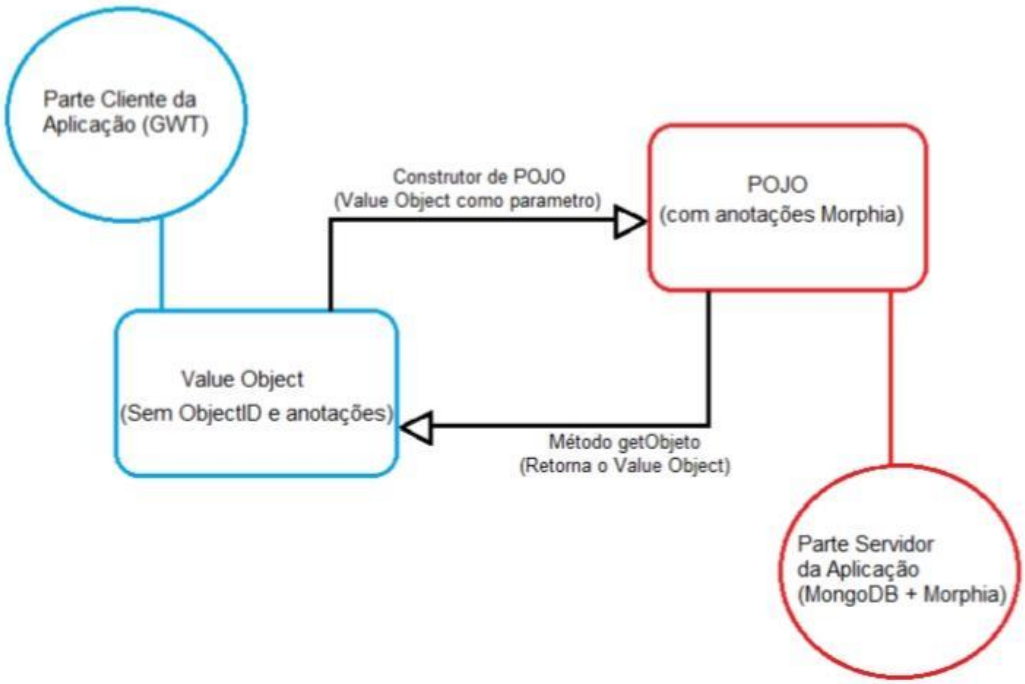

Figura 15 – Diagrama de Funcionamento da Conversão de Pojo para *value object*

3.6 DAO

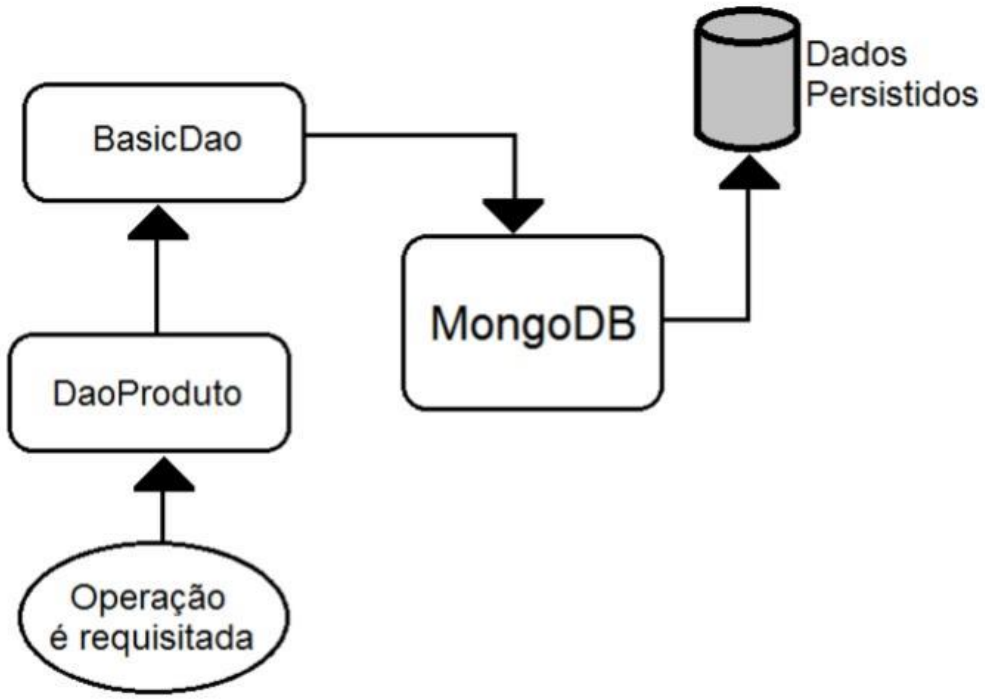

Figura 16 – Diagrama do Funcionamento Básico dos DAOs

3.7 Resultados alcançados

programação eliminam muitos dos possíveis problemas que podem ser gerados.

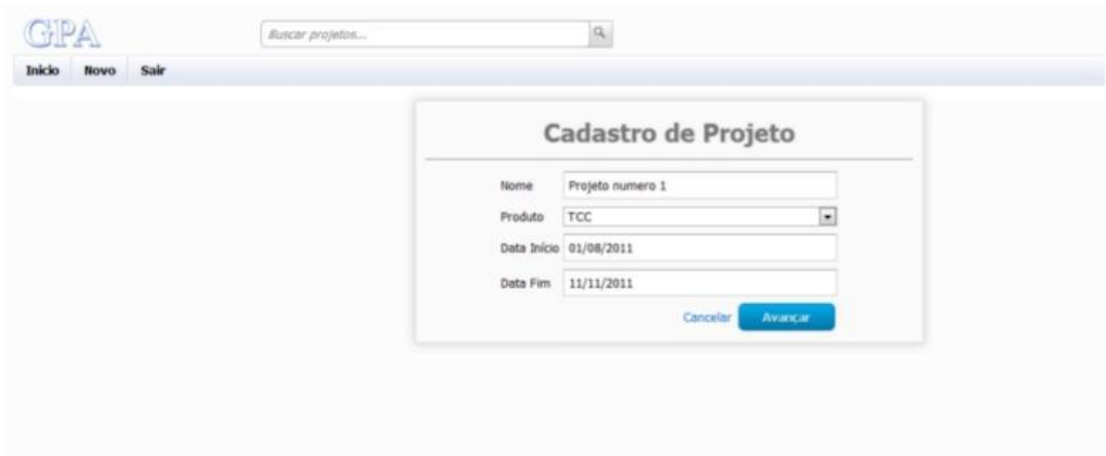

Figura 17 – Tela de Cadastro

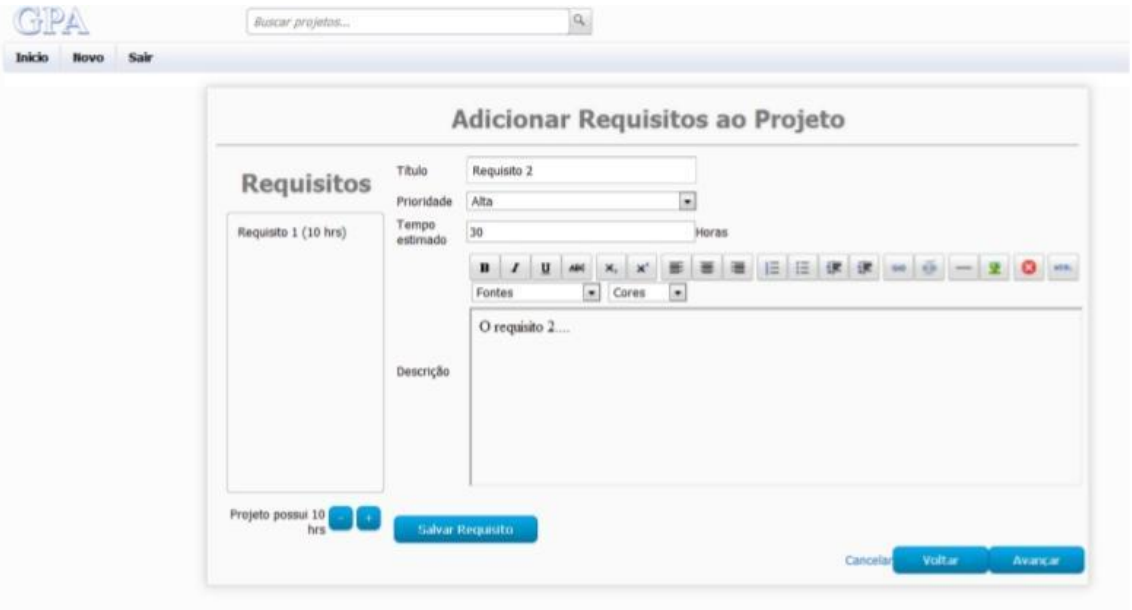

Figura 18 – Tela de Cadastro de Requisitos

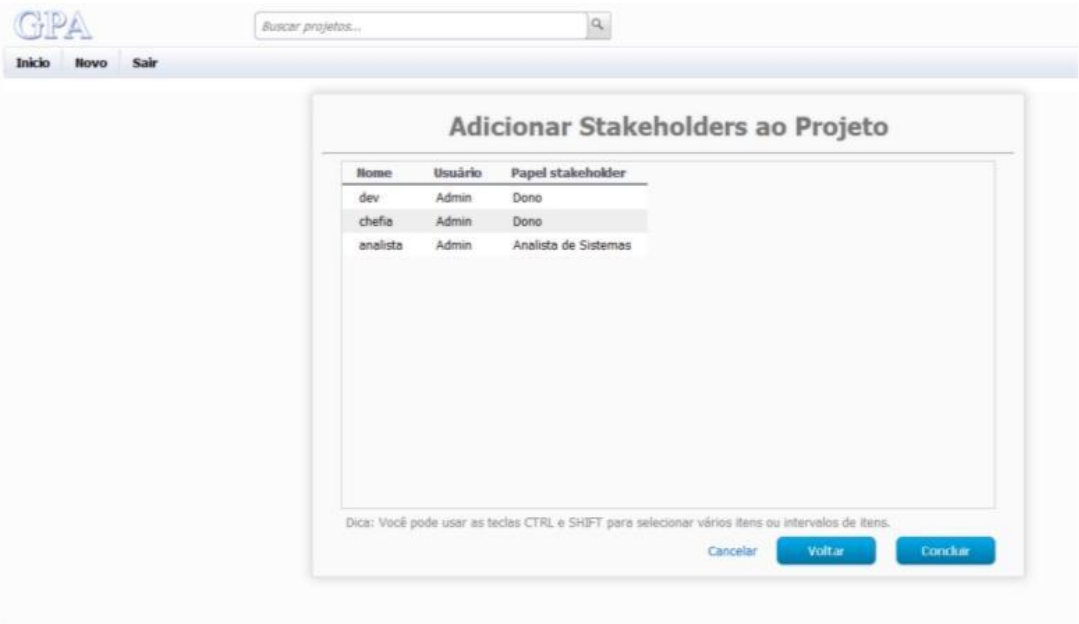

Figura 19 – Tela de seleção de Stakeholders

Figura 20 – Quadro de Scrums na Principal do sistema

4.CONCLUSÃO

Através deste trabalho, foi possível compreender os fundamentos teóricos do NoSQL, bem como as características, vantagens e funcionalidades do MongoDB, destacando sua estrutura orientada a documentos, a escalabilidade horizontal via sharding e os mecanismos de alta disponibilidade proporcionados pelos replica sets.O estudo de caso desenvolvido, com uma aplicação web integrada ao MongoDB, demonstrou na prática como essa tecnologia pode ser aplicada de forma eficiente, proporcionando agilidade no desenvolvimento, facilidade na manipulação de dados semiestruturados e melhoria no desempenho de operações em ambientes distribuídos. Conclui-se, portanto, que o MongoDB representa uma solução robusta e eficiente para sistemas que exigem alta disponibilidade, flexibilidade na modelagem de dados e escalabilidade. Além disso, a combinação das ferramentas e frameworks utilizados, como **Morphia, Google Web Toolkit (GWT) e Google Guice**, contribuiu para um desenvolvimento mais produtivo, organizado e aderente às necessidades de aplicações modernas. Por fim, o presente trabalho reforça a importância da adoção de bancos de dados NoSQL no cenário atual da engenharia de software, especialmente para projetos que lidam com grandes volumes de dados e exigem alto desempenho e flexibilidade.

4.1 Possível Melhoria no Trabalho de Santos

Uma possível melhoria para o sistema desenvolvido por **Santos (2011)** é a sua integração com serviços de computação em nuvem. Atualmente, a aplicação foi projetada para ser executada em um ambiente local, o que limita sua escalabilidade, disponibilidade e acesso remoto. A adoção de uma infraestrutura baseada em nuvem oferece diversas vantagens, tanto no aspecto técnico quanto operacional.

4.2 Possível Melhoria Neste Trabalho

Uma possível melhoria que poderia ter sido incorporada ao trabalho de **Sant** é a realização de uma análise comparativa entre o **MongoDB** e outros bancos de dados, tanto **NoSQL** quanto **relacionais**, aplicados aos mesmos processos e funcionalidades desenvolvidos no sistema.

Referências Bibliográficas

AMAZON WEB SERVICES. *O que é um banco de dados NoSQL?* Disponível em: https://aws.amazon.com/pt/nosql/. Acesso em: 09 jun. 2025.

STROZZI, Carlo. *NoSQL - A Relational Database Management System Without SQL Interface.* 1998. Disponível em: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page. Acesso em: 14 jun. 2025.

AMAZON WEB SERVICES. Bancos de dados orientados a documentos. AWS. Disponível em: https://aws.amazon.com/pt/nosql/document/. Acesso em: 16 jun. 2025.

SANTOS, Raduan Silva dos. *Desenvolvimento de aplicação para o controle de Scrum com MongoDB.* 2011. 58 f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Universidade Tecnológica Federal do Paraná, Campus Medianeira, 2011.

PORCELLI, Alexandre. NoSQL – Bancos de dados não relacionais. 2010. Disponível em: https://www.infoq.com/br/articles/nosql-introduction/. Acesso em: 17 jun. 2025.

MONGODB INC. MongoDB Manual. 2024. Disponível em: https://www.mongodb.com/docs/manual/. Acesso em: 10 jun. 2025.

CÓDIGO FONTE TV. NoSQL // *Programmer's Dictionary* [vídeo]. YouTube, 2018. Disponível em: https://youtu.be/1B64oqE8PLs. Acesso em: 06 jun. 2025.