**Relatório de Arquitetura de Software utilizando o Padrão ADD**

## **Identificação**

## Eduarda Paiva Domingues, Gabriel Vinicius Santos Lima, Leonardo Moret De Oliveira, Maria Alice Brito Nazaré, Stefano Arantes Matarangas e Victor Julio Garcia.

|  |  |  |
| --- | --- | --- |
| Nome Proposto para o projeto:  Projeto Back-End | Prazo de Início:  07/03/2024 | Prazo do Fim:  24/06/2024 |
| Unidade Proponente:  Faria Lima | | |

**Projeto:**

Sistema de Simulador para o Gerenciamento de Ocorrências nas Linhas do Trem da CPTM.

**Introdução**

Este relatório apresenta a arquitetura de software do projeto "Sistema de Simulação para o Gerenciamento de Ocorrências nas Linhas do Trem da CPTM", utilizando o padrão Architectural Design Decisions (ADD). O objetivo principal é documentar as decisões arquiteturais tomadas durante o processo de desenvolvimento, fornecendo uma visão abrangente da estrutura do sistema e justificando as escolhas realizadas.

**Objetivos do projeto**

O principal objetivo do projeto é desenvolver um sistema de ocorrências eficiente e confiável para a CPTM, que permita o registro, gerenciamento e análise de incidentes e ocorrências nas linhas de trem operadas pela empresa. Além disso, os objetivos específicos incluem:

Melhorar a Gestão de Ocorrências: Facilitar o processo de registro e acompanhamento de ocorrências, garantindo uma resposta rápida e eficaz por parte da CPTM.

Facilitar a Análise de Dados: Coletar e armazenar dados sobre ocorrências para análise posterior e geração de relatórios estatísticos e gerenciais.

Melhorar a Gestão de Incidentes: Desenvolver um sistema eficiente para registrar, monitorar e responder a incidentes e ocorrências nas linhas de trem operadas pela CPTM, visando uma gestão mais ágil e eficaz.

Para o melhor desenvolvimento do projeto foi estabelecido uma divisão em **duas etapas**, especificadas com datas e dentro do prazo a ser seguido.

**Visão Geral da Arquitetura**

A arquitetura de software do projeto é composta por diversos componentes interrelacionados que trabalham em conjunto para atender aos requisitos funcionais e não funcionais definidos. A seguir, são apresentadas as principais características da arquitetura:

• Componentes Principais:

• Controlador de Ocorrências.

• Simulador de Ocorrências.

• Gerenciador de Usuários.

• Serviço de Ocorrências.

• Servidor de Banco de Dados.

• Servidor de Aplicação.

• Comunicação entre Componentes.

• Padrões Arquiteturais Utilizados:

• MVC (Model-View-Controller) que contribui para a organização e modularidade do código, facilitando sua compreensão e reutilização. O padrão MVC, por exemplo, separa as responsabilidades do sistema em três camadas distintas: modelo, visão e controle, promovendo maior clareza e testabilidade.

• Microsserviços para os serviços de Back End.

Padrões de projeto e desenvolvimento:

Abstract Factory:

Neste projeto, desenvolvemos uma simulação para o gerenciamento de ocorrências nas linhas de trem da CPTM. O sistema é projetado para lidar com uma variedade de incidentes, como acidentes, falhas técnicas e condições climáticas adversas. Para facilitar a criação e gerenciamento desses diferentes tipos de ocorrências, utilizamos o padrão de design Abstract Factory.

O padrão Abstract Factory fornece uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Isso é especialmente útil em sistemas complexos onde diferentes tipos de objetos precisam ser criados de maneira consistente.

**Requisitos Funcionais e Não Funcionais:**

|  |  |  |
| --- | --- | --- |
| **Código** | **Tipo** | **Descrição** |
| 01 | RF | O sistema deve informar se houve ocorrência |
| 02 | RF | O sistema deve gerar relatório de cada ocorrência |
| 03 | RF | O sistema deve apresentar o histórico de ocorrências |
| 04 | RNF | O sistema deve ter rápida latência |
| 05 | RF | O sistema deve realizar cálculos estatísticos |
| 06 | RF | O sistema deve classificar as ocorrências |
| 07 | RF | O sistema deve comunicar a equipe |
| 08 | RF | O sistema deve comunicar os passageiros |
| 09 | RF | O sistema possuir teste para evitar falso positivo |
| 10 | RF | O sistema deve retornar o tempo de resposta de acordo com a linha e ocorrência |
| 11 | RNF | O sistema ser seguro |
| 12 | RF | O sistema deve retornar e receber arquivo em Json |
| 13 | RF | O sistema deve computar média de tempo e comunicar atraso |
| 14 | RF | O sistema deve apresentar sugestões na correção do tráfego |
| 15 | RNF | O sistema deve ser escalável para lidar com grande volume de dados |

**Decisões Arquiteturais**

A estrutura do sistema é delineada por uma série de decisões arquiteturais significativas, cada uma delas contribuindo para a funcionalidade e desempenho do software. Abaixo, são destacadas as principais decisões tomadas:

Decisão Arquitetural 1:

Estruturação do Sistema em Microsserviços

• **Contexto:** Visando a modularidade e escalabilidade do sistema.

• **Alternativas Consideradas:** Microsserviços.

• **Justificativa:** Os Microsserviços permitem o desenvolvimento independente de componentes, facilitando a manutenção e escalabilidade do sistema.

Decisão Arquitetural 2:

Utilização de um Banco de Dados JSON

• **Contexto:** Escolha da tecnologia de armazenamento de dados, neste caso o JSON.

• **Alternativas Consideradas:** JSON.

• **Justificativa:** Um banco de dados JSON oferece consistência e integridade de dados necessárias para o sistema de gerenciamento de ocorrências.

**Metodologia Ágil e Histórias de Usuário**

**Backlog do Produto**

1. Registro de Ocorrências

* Criar formulário para registro de ocorrências.
* Adicionar campos para data, hora, linha de trem, descrição, categoria e status.
* Validar entradas do formulário.

1. Consulta de Ocorrências

* Criar a consulta de ocorrências.
* Implementar filtro para data, hora, linha de trem, descrição, categoria e status.
* Exibir os resultados.

1. Relatórios e Estatística

* Criar toda a parte de métodos quantitativos.
* Exibir as estatísticas em formato de JSON.

**Planejamento das sprints**

#### Sprint 1

* Registro de Ocorrências
* Criação das classes e métodos

#### Sprint 2

* Consulta de Ocorrências
* Criação de Protótipos
* Implementação do framework Spring

#### Sprint 3

* Revisão das classes
* Implementação da parte de Estrutura de Dados
* Criação dos Arquivos JSON

#### Sprint 4

* Relatórios e Estatísticas
* Finalização e testes

**Histórias de Usuário**

Como administrador do sistema de transporte ferroviário, quero poder cadastrar e atualizar facilmente as informações de estações, linhas e trens, para assegurar que todas as alterações operacionais sejam refletidas em tempo real no sistema. Isso é crucial para fornecer aos passageiros dados precisos e confiáveis que eles dependem para planejar suas viagens.

Como responsável pela logística, preciso de uma ferramenta que me ajude a planejar e otimizar os horários e rotas com base no comportamento atual e esperado dos passageiros, para que possamos oferecer um serviço confiável e eficiente.

Como operador de monitoramento, quero registrar rapidamente qualquer incidente que ocorra na malha ferroviária, para que a equipe de resposta possa agir prontamente, minimizando atrasos e garantindo a segurança dos passageiros.

Como analista, desejo acessar um histórico detalhado de viagens e incidentes, para identificar tendências e áreas que necessitam de melhorias, ajudando a melhorar continuamente o serviço prestado aos passageiros.

Como gestor operacional, preciso de relatórios que resumem o desempenho do sistema ferroviário, para que eu possa apresentar resultados e justificar investimentos, bem como identificar oportunidades de otimização.

## **Diagrama de Casos de Uso**

## Representam os principais fluxos de interação do usuário com o sistema em diferentes situações.

Diagrama

Descrição gerada automaticamente

## **Diagrama de Classe**

O diagrama de classes é uma ferramenta importante na modelagem de sistemas orientados a objetos e serve para representar a estrutura estática do sistema, mostrando as classes do sistema, seus atributos:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

## **Diagrama de Componentes**

O diagrama de componentes serve para representar a estrutura física ou de implantação do sistema, mostrando os componentes do sistema e as dependências entre eles.

Diagrama, Esquemático

Descrição gerada automaticamente

## **Diagrama Implantação**

## O diagrama de implantação serve para representar a distribuição física dos componentes do sistema em hardware ou ambientes de execução

Diagrama

Descrição gerada automaticamente

## **Ferramentas de Modelagem e Desenvolvimento**

## Foram escolhidas as seguintes ferramentas para suportar o processo de desenvolvimento:

* UML (Unified Modeling Language) para modelagem da arquitetura de software.
* Visual Studio Code/Eclipse para desenvolvimento de código.
* GitHub para versionamento e armazenamento do projeto.
* Trello para organizar e acompanhar o progresso de tarefas e projetos de acordo com o que cada integrante do grupo foi designado.
* Atlas MongoDB, uma base de dados NoSQL para armazenar as ocorrencias e valores estatísticos de forma dinâmica.

### **Ferramentas e Bibliotecas no Projeto Java**

#### **Spring Boot Starter Web**

* **Descrição**: Parte do ecossistema Spring Boot.
* **Funcionalidade**: Fornece tudo o que é necessário para construir uma aplicação web baseada no Spring, incluindo Tomcat incorporado e suporte para RESTful web services.

#### **Spring Context**

* **Descrição**: Um dos módulos centrais do Spring Framework.
* **Funcionalidade**: Fornece funcionalidades de injeção de dependências e gerenciamento de componentes através de um contêiner de IoC (Inversão de Controle).

#### **Spring Data MongoDB**

* **Descrição**: Parte do Spring Data.
* **Funcionalidade**: Facilita o uso do MongoDB em aplicações Spring, fornecendo abstrações e repositórios prontos para uso.

#### **MongoDB Driver Sync**

* **Descrição**: Driver síncrono oficial para MongoDB.
* **Funcionalidade**: Permite que a aplicação se conecte ao banco de dados MongoDB, execute operações CRUD (Create, Read, Update, Delete) e outras operações de banco de dados.

#### **BSON**

* **Descrição**: Biblioteca de serialização para o formato BSON (Binary JSON).
* **Funcionalidade**: Utilizada para converter objetos Java para BSON e vice-versa, permitindo armazenamento e recuperação eficiente de documentos no MongoDB.

#### **MongoDB Driver Core**

* **Descrição**: Parte essencial do driver MongoDB, fornecendo funcionalidades de baixo nível para comunicação com o banco de dados.
* **Funcionalidade**: Suporte para conexões, operações de leitura e escrita, e gerenciamento de clusters.

#### **Apache Commons Math3**

* **Descrição**: Biblioteca de matemática e estatísticas da Apache Commons.
* **Funcionalidade**: Fornece implementações robustas de algoritmos matemáticos e estatísticos, facilitando cálculos complexos na aplicação.

#### **JUnit Jupiter API**

* **Descrição**: Parte do JUnit 5, a API para escrever testes.
* **Funcionalidade**: Oferece anotações, classes e métodos para facilitar a criação de testes unitários em Java.

#### **Spring Boot Starter Test**

* **Descrição**: Parte do Spring Boot.
* **Funcionalidade**: Inclui diversas bibliotecas de teste, como JUnit, Mockito, Hamcrest e AssertJ, facilitando a escrita e execução de testes em aplicações Spring.

#### **Spring Boot Maven Plugin**

* **Descrição**: Plugin do Maven para Spring Boot.
* **Funcionalidade**: Simplifica a construção e execução de aplicações Spring Boot através do Maven, oferecendo comandos específicos para empacotamento, execução e testes.

Ambiente virtual Trello para organização das etapas de entregas, utilizando a metodologia ágil scrum:

* Material para ser implementado de cada matéria:

Interface gráfica do usuário

Descrição gerada automaticamente

* Organização de tarefas atualizado 02/05/2024:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Conclusão da primeira etapa**

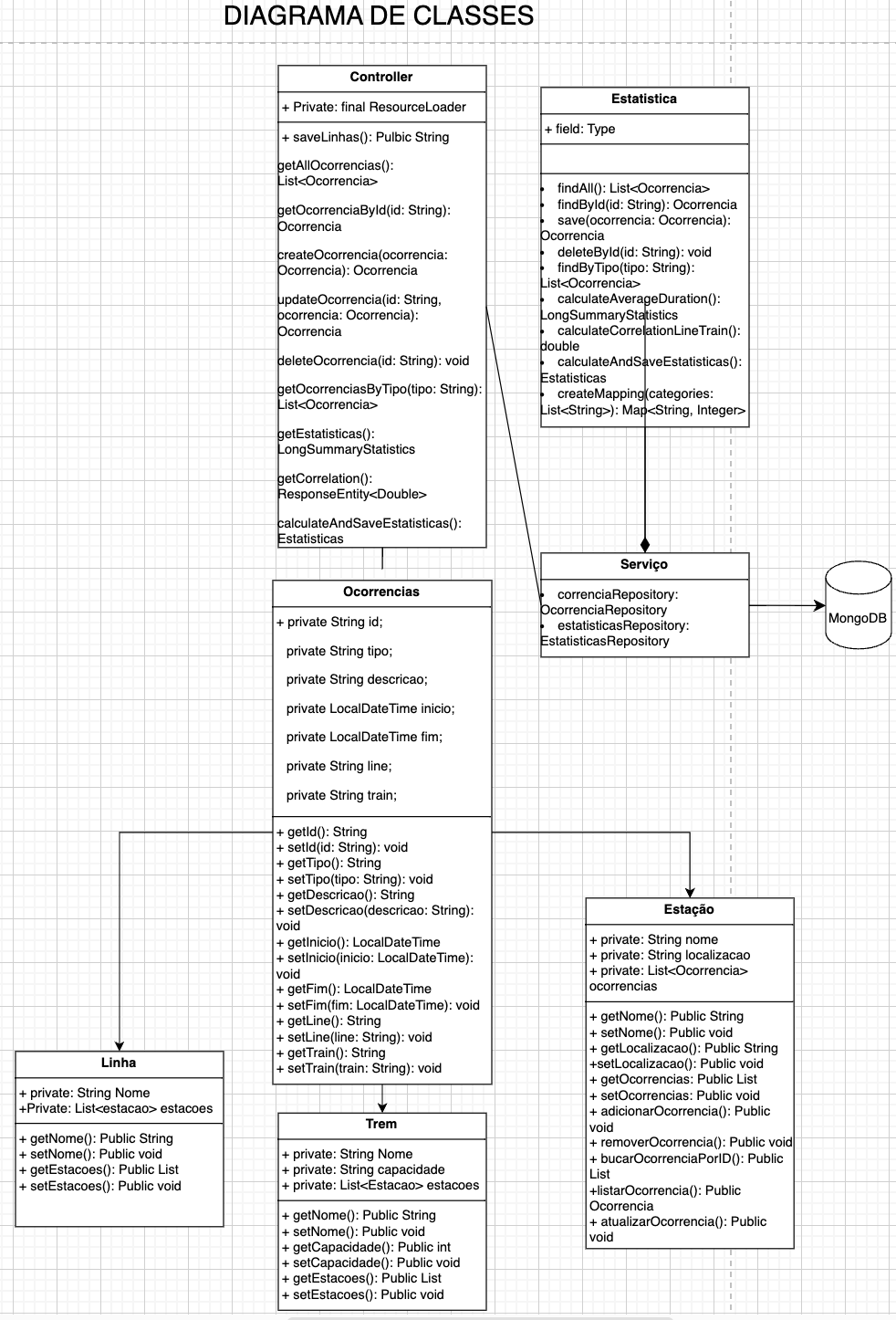
O sistema de gerenciamento de ocorrências nas linhas do trem da CPTM foi projetado levando em consideração os princípios e práticas de arquitetura de software. As decisões arquiteturais tomadas visaram atender aos requisitos funcionais e não funcionais do sistema. Tendo em vista, a organização projetada, consideramos a realização da primeira etapa do projeto, com duração até o dia 16/05/24. Dentro desse período, foi realizado a elaboração de todos os diagramas e requisitos estabelecidos, para dar início a segunda etapa, que tem como seu propósito principal, refinar a implementação geral, trazendo maior eficiência e clareza na didática utilizada.

**Início da segunda etapa**

A segunda parte do projeto conta com a organização geral do projeto, garantindo a clareza nos processos utilizados, eficiência da implementação dos códigos e revisar se todos os conteúdos estão sendo abordados. Após todo o primeiro período, foi determinado a restruturação do código, visto que existiam muitas classes e pastas que não serviam de nada, apenas atrapalhavam visualmente o processo. Consequentemente a isso, também foi decidido ampliar a documentação, seguindo os padrões ADD e inserindo apenas as informações que foram atualizadas. Para as duas etapas serão adicionados vídeos com link no youtube, para facilitar a comparação do processo.

## **Diagrama de Classe atualizado**

O diagrama de classes é uma ferramenta importante na modelagem de sistemas orientados a objetos e serve para representar a estrutura estática do sistema, mostrando as classes do sistema, seus atributos:

­­­­

## **Diagrama de Componentes atualizado**

O diagrama de componentes serve para representar a estrutura física ou de implantação do sistema, mostrando os componentes do sistema e as dependências entre eles.

Diagrama

Descrição gerada automaticamente

## **Diagrama Implantação atualizado**

## O diagrama de implantação serve para representar a distribuição física dos componentes do sistema em hardware ou ambientes de execução

Diagrama

Descrição gerada automaticamente

**Considerações de Manutenibilidade**

**Documentação Adequada**

A documentação detalhada do código-fonte e da arquitetura do sistema é crucial para a compreensão e manutenção. Uma documentação completa deve incluir:

* **Descrição detalhada das classes, métodos e interfaces:** incluindo seus objetivos, parâmetros, valores de retorno e casos de uso.
* **Diagramas de arquitetura:** ilustrando a organização geral do sistema, seus componentes e interações.
* **Explicação dos requisitos funcionais e não funcionais:** detalhando as funcionalidades do sistema e as restrições que ele deve atender.
* **Manual do usuário:** fornecendo instruções passo a passo para a utilização do sistema.

Uma documentação bem elaborada facilita a organização de novos desenvolvedores, reduz o tempo de depuração e contribui para a tomada de decisões mais assertivas em futuras modificações.

**Benefícios da Manutenibilidade**

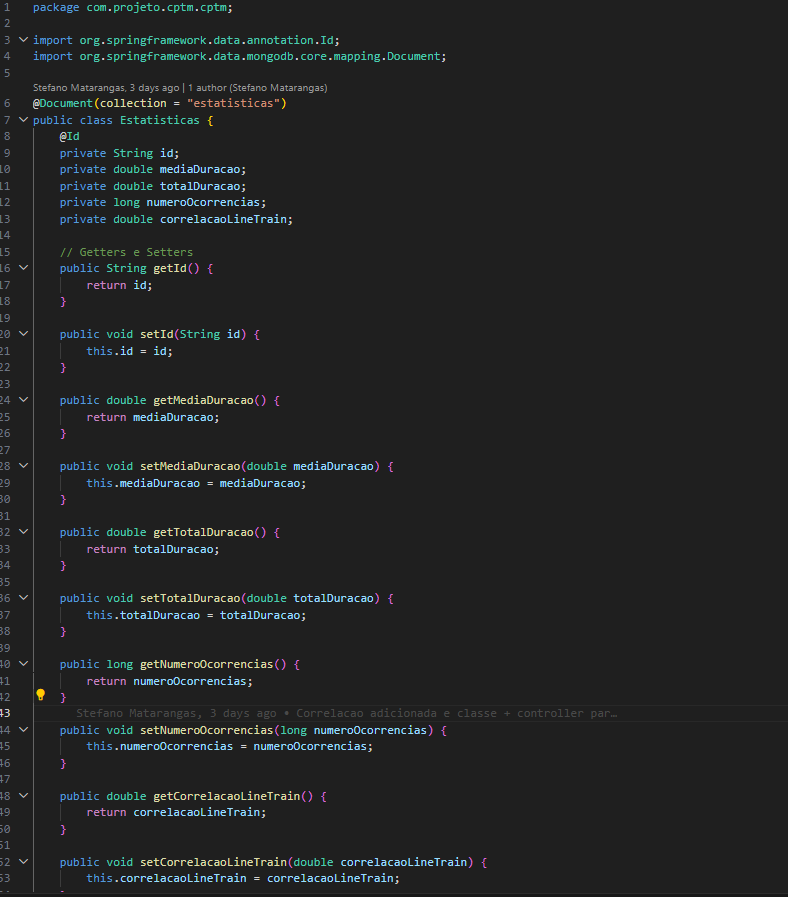
Investir em manutenibilidade durante o desenvolvimento do sistema de gerenciamento de ocorrências da CPTM traz diversos benefícios:

* **Redução de custos:** Menor tempo e esforço para corrigir bugs, implementar novas funcionalidades e adaptar o sistema às mudanças de negócio.
* **Maior agilidade:** Possibilidade de realizar modificações no sistema de forma rápida e segura, sem comprometer sua estabilidade.
* **Melhoria da qualidade:** Menor índice de erros e maior confiabilidade do sistema.
* **Maior vida útil:** O sistema se torna mais adaptável às novas tecnologias e necessidades do negócio, prolongando sua vida útil.

**Algumas partes do código**

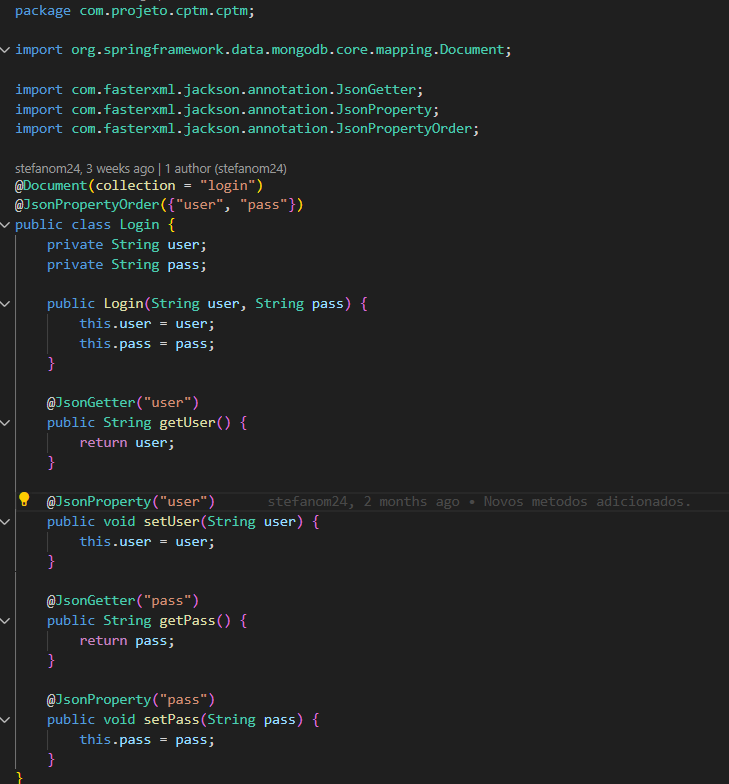
Parte de Estatística:

A classe “Estatisticas” utiliza o framework Spring Data MongoDB para mapear objetos Java para documentos na coleção "estatisticas" do MongoDB.



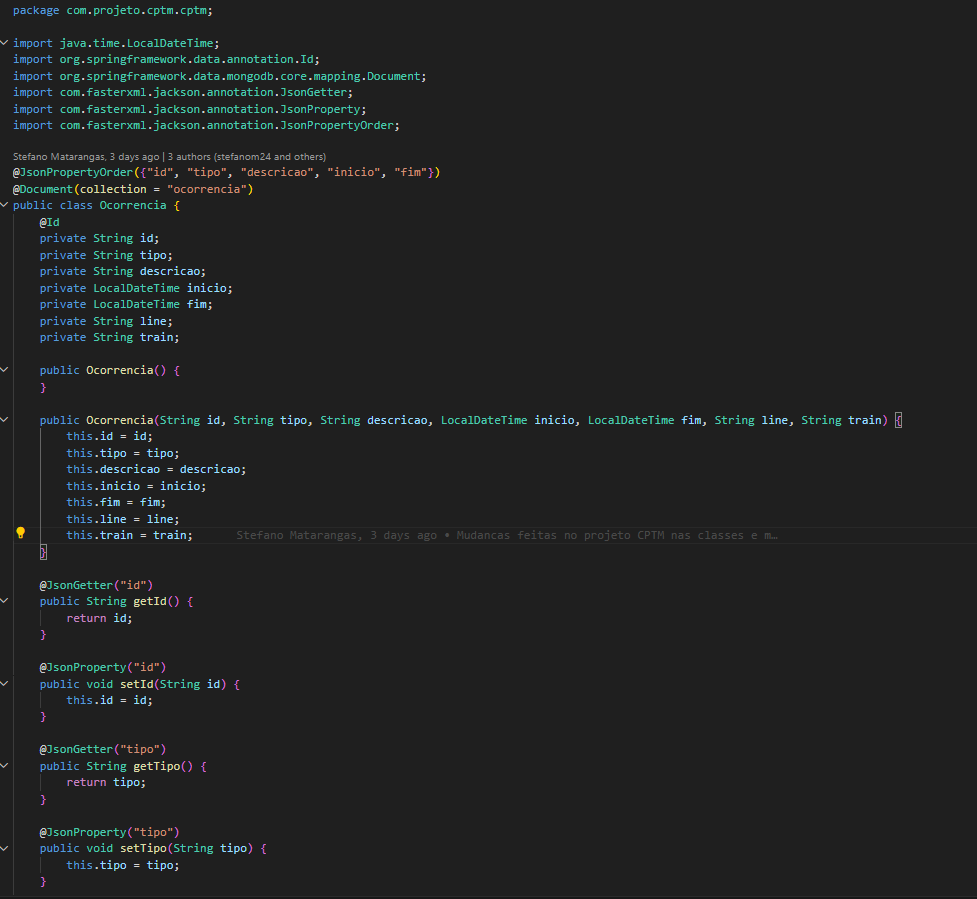
Parte de Login:

Ela é responsável por representar as informações de login dos usuários e utiliza o framework Spring Data MongoDB para mapear objetos Java para documentos na coleção "login" do MongoDB. Além disso, a classe faz uso de anotações do Jackson para controle da serialização/deserialização JSON.



Classe de ocorrências:

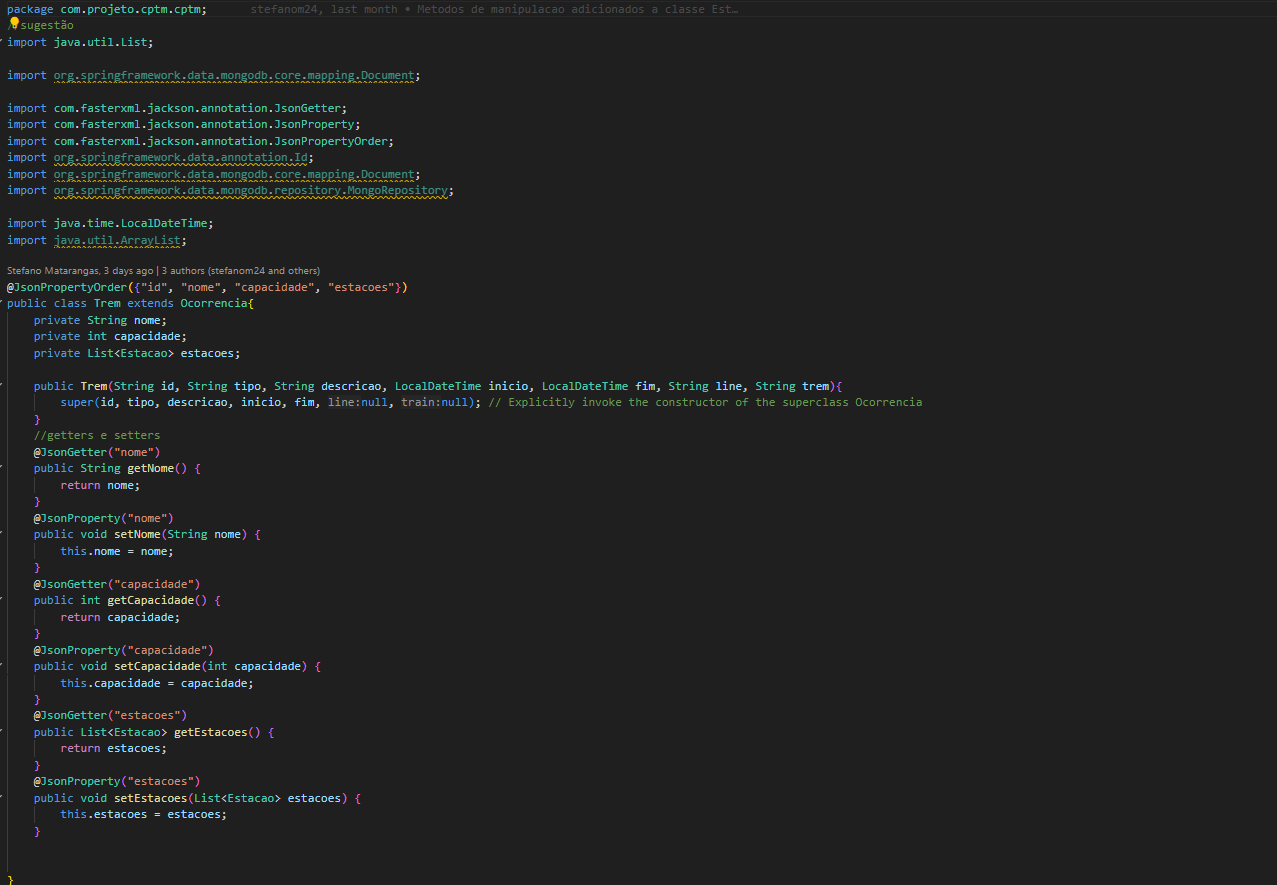
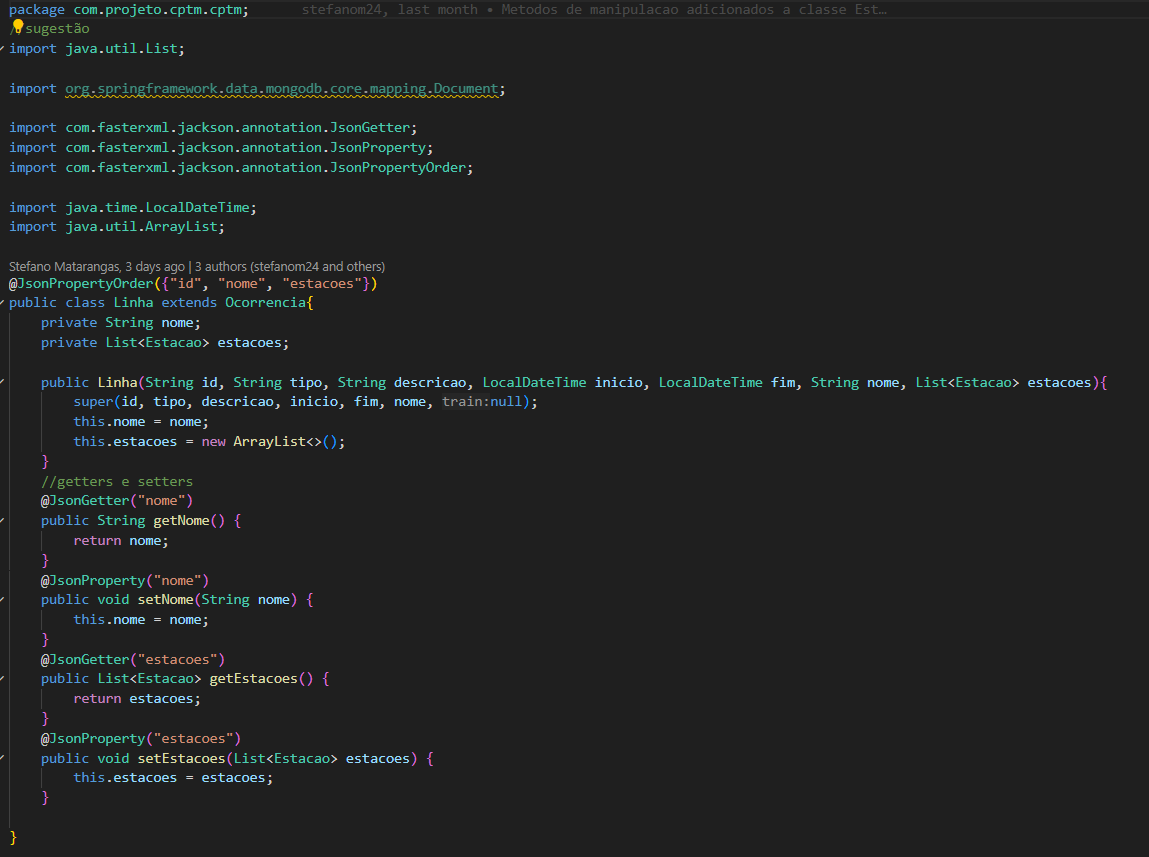
Representa e armazena informações detalhadas sobre as ocorrências nas linhas e trens da CPTM, permite persistir objetos na coleção "ocorrencia" do MongoDB, facilitando operações CRUD e utiliza anotações do Jackson para controlar a forma como os dados são convertidos para JSON e lidos de JSON, garantindo compatibilidade com APIs e outros serviços que utilizam JSON.



Classes de linha e trem:

* Linha representa uma linha de trem, incluindo suas estações.
* Trem representa um trem específico, incluindo suas estações e capacidade.

Ambas reutilizam atributos e métodos da classe “Ocorrencia”, permitindo gerenciamento consistente de dados e utilizam anotações do Jackson para controlar a forma como os dados são convertidos para JSON e lidos de JSON, garantindo compatibilidade com APIs e outros serviços que utilizam JSON.



**Melhorias Futuras**

* Testes Unitários/testes automatizados.
* Organização do Código.
* Adicionar funcionalidades, como, cálculos estatísticos e funções do MongoDB.
* Integração do projeto com os outros projetos desenvolvidos para a CPTM.

**Conclusão**

O sistema de gerenciamento de ocorrências nas linhas do trem da CPTM foi projetado considerando os princípios e práticas de arquitetura de software, com especial atenção aos padrões de design e desenvolvimento. Utilizando o padrão Architectural Design Decisions (ADD), documentamos detalhadamente as decisões arquiteturais tomadas ao longo do desenvolvimento, assegurando que o sistema atenda tanto aos requisitos funcionais quanto aos não funcionais estabelecidos.

A implementação do padrão Abstract Factory facilitou a criação e o gerenciamento de diversos tipos de ocorrências, proporcionando flexibilidade e extensibilidade ao sistema. A escolha por uma arquitetura baseada em microsserviços e a utilização de um banco de dados JSON foram decisões cruciais que promoveram a modularidade, escalabilidade e integridade dos dados.

Durante a primeira etapa do projeto, realizamos a elaboração de todos os diagramas e requisitos, estabelecendo uma base sólida para o desenvolvimento. Na segunda etapa, focamos na organização do projeto, refatoração do código e ampliação da documentação, seguindo os padrões ADD, para garantir clareza, eficiência e uma documentação abrangente.

As considerações de manutenibilidade, como a documentação adequada, foi essencial para assegurar a continuidade e a qualidade do sistema. Com essas práticas, o sistema de simulação para o gerenciamento de ocorrências não só atende às necessidades atuais da CPTM, mas também está preparado para adaptações e melhorias futuras.

Em resumo, o projeto atingiu seus objetivos principais, proporcionando à CPTM uma ferramenta robusta para o sistema de simulação para o gerenciamento de ocorrências nas linhas de trem.

**Referências**

* [*https://trello.com/b/siPs2DGn/backlog*](https://trello.com/b/siPs2DGn/backlog)
* <https://github.com/Leonardo0liveira/Projeto-Backend>
* [https://spring.io](https://spring.io/)
* <https://www.thunderclient.com>
* Primeira etapa: <https://www.youtube.com/watch?v=hKaF4f0gYns>
* Segunda etapa: <https://youtu.be/TK2JZIuGgFA>