
# Relatório da Entrega 2 - AGENDUSP

## Projeto de Laboratório de Orientação a Objetos

André Akira Horigoshi Maximino

Diego Fontes de Avila

Gabriel Lopes Prodossimo

João Victor Costa Teodoro

Matheus Davi Leão

Sophia Soares Mariano

Professor Denis Deratani Mauá

MAC 0321

2025

# Definição do projeto

O projeto AGENDUSP é um **Assistente para Organização Pessoal** e tem como objetivo implementar uma Aplicação Web que, utilizando a API do Google Calendar e a ferramenta de Inteligência Artificial Ollama, é capaz de criar, alterar e sugerir futuros eventos e datas de compromissos para o usuário. Além disso, o programa pode fazer alterações no calendário principal, acessando o Google Calendar do usuário por meio de requisições.

Dessa forma, ao permitir a criação de eventos em grupo e ter sua classificação bem definida, o usuário tem facilidade em entender que tipo de compromisso que está em sua agenda, além de ver quanto tempo cada uma delas demanda e seu tempo total de reuniões na semana. O sistema embutido de IA também poderá gerar texto, o qual explica ao usuário seus compromissos, e sugerir melhores horários e dias, caso o usuário queira marcar um evento futuro.

# Lista consolidada de integrantes do grupo

1. André Akira Horigoshi Maximino
2. Diego Fontes de Avila
3. Gabriel Lopes Prodossimo
4. João Victor Costa Teodoro
5. Matheus Davi Leão
6. Sophia Soares Mariano

# Funcionalidades implementadas e exemplos de uso

O AGENDUSP foi pensado para ser executado com uma separação entre frontend, escrito em javascript, e backend, escrito em Java. Desse modo, a arquitetura MVC é implementada para que o frontend possa “consumir” a REST API do backend. Foram implementadas classes em vários formatos. As classes denominadas “Repository" são as que definem as interfaces que estendem o sistema de armazenamento de dados que foi criado utilizando o MongoDB, para gerar as classes de eventos, calendário e usuário que foram implementadas nas classes denominadas “Resource". Já nas classes de “Controller”,

são implementados os métodos que realizam as funções de inserir, pegar, deletar, listar, atualizar parcialmente e atualizar totalmente os atributos dos calendários e dos eventos. A classe *DataContoller* implementa de fato as funções de edição dos atributos dos calendários e eventos, enquanto as classes "Local" implementam apenas a chamada dessas funções, passando os parâmetros corretos para dentro delas.

**Autenticação via requisições http:**

O sistema de autenticação do calendário está configurado para funcionar já na interface web, que está em um local host, utilizando um sistema de login do Google. A conexão com o ecossistema Google é realizada através de requisições HTTP, por meio do controlador REST, que permite a criação, cancelamento e modificação de compromissos na base de dados.

**Visualização dos eventos pelo usuário**

Para que o usuário veja seus compromissos, ele deve acessar o calendário de sua escolha, utilizando o método `getCalendar()` implementado no *LocalCalendarListController*. Em seguida, ele deve chamar o método `listEvents()`, que mostra todos os eventos marcados naquele calendário. Esses métodos estão descritos no arquivo *LocalEventsController*.

**Acessos aos calendários pelos usuários**

Diferentes papéis de acesso permitem ao usuário executar diferentes funções com os calendários e os eventos. No calendário, o atributo `accessRole` determina a relação que o usuário terá com o calendário. Uma pessoa pode ser uma criadora ou convidada. O nível mínimo de acesso que um usuário pode ter a um calendário é o de "Leitura", ou seja, pode usar a função `getCalendar()` para vê-lo. Apenas os "writers" e "owners" podem dar update, patch e delete em um calendário, mas com diferenças. Por exemplo, quando um "owner" apaga um calendário, ele apaga esse calendário da lista de todos os usuários que possuíam esse calendário também.

**Relações entre eventos e usuários**

Já para os eventos, o sistema é um pouco diferente. O Google Calendar já possui os parâmetros "creator" e "organizer", ou criador e organizador. O criador e o organizador

podem ser pessoas diferentes, sendo o criador apenas quem adicionou o evento, e o organizador o responsável por marcar, desmarcar, adicionar participantes, atualizar o horário, etc. Para convidar uma pessoa, o criador, "owner", deve usar o método `addAttendeeToEvent()`, implementado no *DataController*, em que o "owner" cria um array de "attendees", que o permite adicionar novos participantes ao seu evento apenas utilizando a ID do usuário que se deseja convidar. Para compatibilidade do sistema local com o da Google, foram implementados todos os parâmetros idênticos ao da API.

A classe *EventsResource* possui dois atributos que separam creator e organizer, de forma a diferenciar compromissos criados pelo usuário principal e compromissos criados por outros usuários. Dois métodos diferentes podem ser utilizados para adicionar eventos em um calendário específico. `createEvent()`, onde o usuário precisa necessariamente ser o dono do calendário, cria um compromisso inteiramente novo, e `addEvent()`, que adiciona compromissos de outros calendários não necessariamente pertencentes ao usuário.

**Cancelamento e remoção de eventos**

No arquivo *DataController*, através do método `cancelEvent()`, compromissos cancelados recebem o status "cancelled" na agenda. O método `removeEvent()`, embora semelhante, apenas remove o compromisso do calendário do usuário, e o compromisso pode ainda existir em diferentes calendários do mesmo usuário ou de outros.

**Armazenamento de eventos e calendários em repositórios**

Através do MongoDB, os compromissos são armazenados no arquivo *EventsRepository* através da interface *MongoRepository*. De mesmo modo, o arquivo *UserRepository* e o arquivo *CalendarRepository* armazenam em um banco de dados as informações dos usuários e dos calendários respectivamente.

**Framework e construção do projeto**

O framework utilizado no projeto é o Spring Boot, baseado no Spring Framework, que permite criar aplicações standalone e automatizar muitas tarefas comuns. O Spring Boot utiliza diversos padrões de projeto, entre eles principalmente Inversão de Controle, Injeção de Dependência, Singleton, Factory Method, MVC e Observer. Algumas das

principais vantagens de utilizar o Spring Boot são a facilidade de testes e manutenção, o código mais limpo e desacoplado e a maior clareza na arquitetura.

Para construir uma aplicação Spring Boot que funcione com a API do Google, é necessário lidar com dados no formato JSON, um padrão de troca de informações entre o cliente e o servidor. Aliado ao Gson, que é uma biblioteca do Google para conversão entre objetos Java e JSON, funciona perfeitamente com a API do Google Calendar.

O AGENDUSP, que é o cliente, envia uma requisição HTTP POST para o Spring Boot, que recebe o JSON e usa o Gson para converter. Para os projetos que usam Spring Boot, é comum utilizar a dependência "spring-boot-starter-oauth2-client", também utilizada neste projeto, que serve para obter e renovar o token de autenticação automaticamente.

**Testes implementados no projeto**

Após a implementação dos requisitos, foram implementados testes. Para que os testes possam alterar dados, é necessário utilizar o *MongoTestContainer*, que cria um container docker temporário com uma base de dados para os testes. O teste mais bem sucedido foi o *UserRepositoryTest*, que testa as operações de bases de dados, e testa se as Query de usuário estão funcionando. Esse teste mostra o funcionamento de diversos métodos, como a adição do *UserCalendarList*, a lista de calendários do usuário. Foram implementados outros testes para o *DataController* e para a REST API, no entanto, houveram dificuldades em relação à forma como o Spring Boot gerencia seu fluxo de autorização, de modo que, no momento, não foi possível testar a REST API por conta de problemas de permissões. Ademais, pela ausência do frontend, ainda não é possível executar o sistema sem que se use requisições HTTP puras, o que é complicado por conta dos Tokens de autorização do OAuth2.

## Instruções para a instalação de dependência e a execução do sistema

Para utilizar o sistema, basta instalar a linguagem Java em seu dispositivo e baixar o repositório no branch **feature/129-datacontroller**, que é o branch mais recente. Em seguida, na pasta do projeto, deve-se rodar o sistema no terminal utilizando o comando `./gradlew build` seguido de `./gradlew bootRun` se estiver em um sistema Linux/macOS. Caso esteja

em um sistema Windows, deve-se usar `gradlew.bat build` seguido de `gradlew.bat bootRun`.