# Sistema de Pré-Matrícula CC-UFCG

## 1. Contextualização e Introdução

O Curso de Ciência da Computação da UFCG precisa de um sistema online que gerencie as pré-matrículas dos seus alunos nas disciplinas ofertadas pelos departamentos da universidade.

### 1.1. Problema

A necessidade por um sistema deste tipo surge porque é preciso um mecanismo que mensure o interesse dos alunos e, consequentemente, tomar medidas para que o melhor escalonamento das disciplinas e vagas aconteça.

O coordenador precisa saber a intenção dos alunos, ou seja, quais são as disciplinas mais procuradas e, a partir disso, agir para que não falte vagas aos alunos solicitantes.

### 1.2. Tecnologias

Para o back-end, foi escolhido Java com o auxílio do framework Spring boot, que facilita grande parte do trabalho de deploy. Esta característica é interessante porque nos ajuda na produtividade do desenvolvimento.

Como tecnologia para o front-end, foi escolhido Angular, que é um framework Javascript construído sobre o padrão MVVM (Model-View-View-Model). Isto facilita tanto o desenvolvimento quanto os testes da aplicação.

# 2. Requisitos

No sistema de Pré-Matrícula CC-UFCG, será possível realizar as pré-matrículas, acompanhar a disponibilidade de vagas de cada disciplina, verificar a validade das matrículas e oferecer suporte a outras atividades da universidade.

Estas funcionalidades foram extraídas dos requisitos especificados pelo cliente (coordenação do Curso de Computação) e, para pensar na solução, a etapa de análise de requisitos é crucial para o desenvolvimento de um software funcional e com poucos bugs.

## 2.1. User Stories

As user stories compõem a especificação e são parte importante da documentação, pois elas nos dão um ponto de vista do usuário final e, pelo fato da descrição ser feita em linguagem natural, o entendimento é mais fácil. Os casos de uso do sistema estão separados pelos atores e estão listados a seguir:

***Usuário:***

1. Eu, como usuário, gostaria de acessar o sistema através de um link na web.

***Aluno:***

1. Eu, como aluno, gostaria de logar no sistema usando meu email institucional *(@ccc)* para ter acesso às funcionalidades destinadas a mim. Obs: Na primeira vez que o aluno logar no sistema deve ser solicitada e armazenada a sua matrícula e se ele está na grade nova ou antiga. A partir da matrícula é possível saber o período de entrada do aluno.
2. Eu, como aluno, gostaria de realizar a pré-matrícula semestral pelo sistema, selecionando quais disciplinas eu desejo cursar no semestre atual da universidade.
3. Eu, como aluno, na realização da minha matrícula, gostaria de visualizar as disciplinas em que eu posso me matricular. Considerar todas as disciplinas ofertadas para a grade que o aluno está (nova ou antiga).

***Coordenador:***

1. Eu, como coordenador, gostaria de logar no sistema usando meu email institucional (@ccc), para ter acesso às funcionalidades destinadas a mim.
2. Eu, como coordenador, gostaria de cadastrar uma nova disciplina no sistema, informando o nome da disciplina, código, nº de créditos, carga horária e se é ofertada na grade nova/antiga/ambas.
3. Eu, como coordenador, gostaria que o sistema atribuísse automaticamente o status das pré-matrículas dos alunos, sendo INVÁLIDA para total de créditos menor que 16 ou maior que 24, e sendo VÁLIDA para total de créditos maior

igual que 16 e menor igual que 24. Apenas pré-matrículas com status VÁLIDA são aceitas no sistema.

4. Eu, como coordenador, gostaria de consultar a lista de pré-matrículas dos alunos. O sistema retorna um arquivo csv com as seguintes colunas: nome do aluno, e-mail institucional, matrícula, período de entrada no curso, disciplinas escolhidas na pré-matrícula.

# 3. Sistema de Pré-Matrícula CC-UFCG

## 3.1. Arquitetura

Em uma visão mais alto nível, o sistema de pré-matrícula segue a típica arquitetura web: front-end (a interação com aluno e coordenador acontece neste componente) e back-end (um servidor HTTP que hospedará a aplicação e um servidor de banco de dados que persistirá os dados).

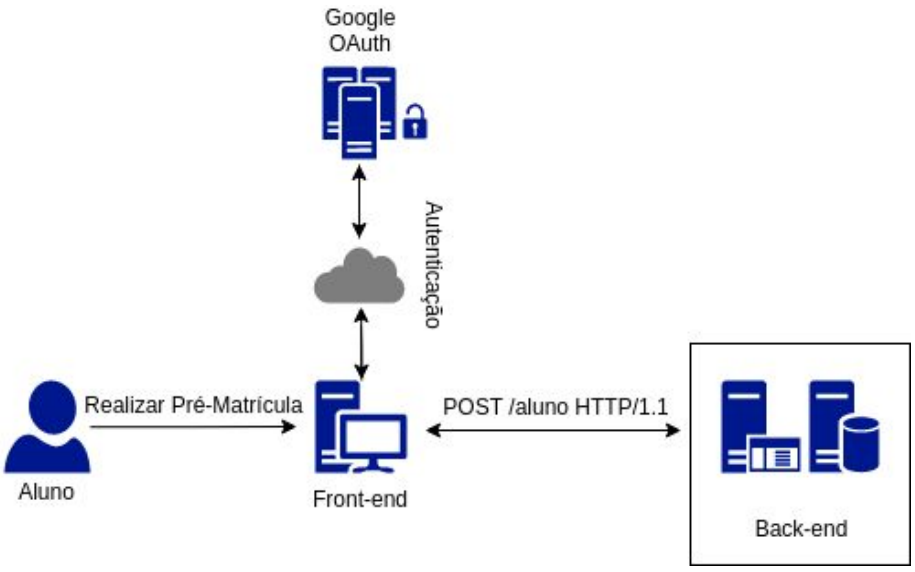


**Figura 1: Visão Geral do Sistema de Pré-Matrícula CC-UFCG**

### 3.1.1. Back-end

O back-end do sistema é responsável pela lógica de negócio, persistência e processamento das requisições. As interações com este componente acontecem através de uma API REST que foi pensada para ser fácil e intuitiva de ser utilizada.

Interessante chamar atenção para como a persistência de dados acontece e quais os efeitos das escolhas feitas. Algumas características importantes a sistemas distribuídos são habilitadas quando há a distribuição pensada dos componentes, como por exemplo a disponibilidade (o acesso à persistência é delegado para outro host) e a falha à tolerância (o sistema é stateless, ou seja, não guarda estado).

### 3.1.2. Front-end

O front-end se comunicará com os endpoints definidos no *repositório do sistema de pré matrícula*. É importante frisar que o front-end precisará de um ponto de autenticação remoto, já que a entrada do aluno ao sistema de pré-matrícula somente será permitida pela autenticação com o Google. As ações de autenticação e autorização serão determinadas pelo protocolo OAuth.

## 3.2. Design

Decisões de design bem tomadas podem definir a vida útil de um sistema. Trabalhar com as melhores práticas permitem aos desenvolvedores um código claro e de fácil manutenção. Por exemplo. se este sistema estiver bem especificado, a arquitetura bem definida e as decisões de design fizerem sentido, é muito possível que ocorra uma integração do Sistema de Pré-Matrícula CC-UFCG ao Controle Acadêmico da UFCG, podendo este ser ampliado para um sistema de pré-matrícula para todos os cursos da UFCG.

### 3.2.1. Back-end

Alguns padrões foram incorporados ao back-end do sistema. Dentre eles estão: *MVC, Controller* e *Singleton*. Nesta seção, justificamos o porquê destes padrões existirem.

*Obs:* Para uma visão geral do design do sistema e para o melhor entendimento das decisões de design que foram tomadas, é imprescindível consultar o diagrama de classes do sistema.

> ***MVC (Model-View-Controller):*** O objetivo do MVC aqui é separar a apresentação da lógica de negócio do sistema. A divisão de responsabilidades entre o **M**odelo, **V**isão e **C**ontrole facilita a manutenção e desenvolvimento do código. O *Model* (Aluno, Coordenador e Disciplina) armazena a lógica de negócio para cada entidade existente. A *View* é definida como a forma de apresentação para o usuário do back-end e é utilizada através de uma API REST. O *Controller* é o componente responsável pela alteração do estado do *Modelo* ou da *Visão,* e esta alteração será baseada nas entradas do usuário.
>
> ***Controller:*** Para cada recurso da API, define-se um controller que serve como ponto de entrada e como gerenciador dos recursos. Cada controller detém a lógica e responsabilidade do seu recurso (*CoordenadorController* lida com as requisições feitas para o recurso *Coordenador*).
>
> ***Singleton:*** A fim de atender a especificação que define que só existe um coordenador no sistema, implementa-se o padrão de projeto Singleton. Este padrão garante que existe apenas uma instância de uma classe (no nosso caso, *Coordenador*), mantendo um ponto global de acesso ao seu objeto.

Tão importante quanto saber quando usar um padrão é saber quando não usar. Alguns padrões conhecidos como *Facade*, *Factory*, *State* e *Strategy* não foram utilizados ou pelo trade off existente ou por simplesmente não haver a necessidade. A seguir algumas justificativas para os padrões citados:

1. No caso do *Facade* há um trade off explícito: a quantidade de elementos que precisarão de alteração caso alguma funcionalidade precise ser adicionada é maior.
2. A arquitetura não exige troca de tipos em tempo de execução, os decorators do Spring boot fazem este papel de checar a validade, ou seja, não há a necessidade de *Factory*.
3. *State* e *Strategy* estão fora, pois em nenhuma situação do sistema é necessária a troca de estado/contexto de um objeto

Também é válido destacar decisões de design que não incorporam padrões explícitos, como por exemplo a classe *Aluno* que apenas armazena os códigos das disciplinas e não os objetos. Isto permite um acoplamento baixo entre as classes *Aluno* e *Disciplina*.