Plano de Especificação: Sistema de Agendamento de Serviços Acadêmicos

Versão 2.1

2024

Sumário

1. Plano de especificação do ASA

Este documento é responsável por apresentar o plano de especificação formal para o desenvolvimento do ASA - Sistema de Agendamento de Serviços Acadêmicos. A especificação garante a precisão e integridade dos requisitos funcionais e não funcionais do sistema.

2. Resumo

O Sistema de Agendamento de Serviços Acadêmicos, desenvolvido em C#, tem como objetivo principal centralizar e automatizar os processos de agendamento de serviços especializados na UFERSA. Atualmente, esses serviços são gerenciados por meio de formulários dispersos no site da universidade. O novo sistema integrará todas essas informações em um único local, simplificando o acesso e tornando o processo mais eficiente e transparente para os usuários.

Como requisitos, o sistema permitirá que os usuários se cadastrem e façam alterações em suas senhas, além de gerenciar os serviços acadêmicos disponíveis. Os usuários poderão visualizar os horários disponíveis para cada serviço e agendar ou cancelar atendimentos de forma prática. Isso visa não apenas facilitar o acesso dos profissionais e estudantes aos serviços acadêmicos, mas também melhorar a transparência e a eficiência do processo de agendamento.

O desenvolvimento do sistema adota uma metodologia híbrida que combina os valores e princípios do XP (Extreme Programming) com as técnicas organizacionais do SCRUM. No XP, o foco será em simplicidade, feedback, comunicação e respeito, enquanto no SCRUM, serão implementados artefatos e eventos, como sprints e reuniões, para garantir uma gestão ágil e colaborativa das entregas contínuas ao longo do projeto. Assim, o sistema será desenvolvido de maneira eficiente e adaptável, atendendo às necessidades específicas dos usuários e do ambiente acadêmico da UFERSA.

3. Pessoas envolvidas

- LAVINIA DANTAS DE MESQUITA
- CRISTIANA DE PAULO

4. Recursos necessários

4.1. Recursos Humanos

4.1.1. **Desenvolvedores de Software (C#)**: Aqueles com experiência em desenvolvimento de software utilizando a linguagem C#, capazes de implementar as funcionalidades do sistema.

4.1.2. **Especialistas em Metodologias Ágeis**: Aqueles com experiência em XP e SCRUM para gerenciar o desenvolvimento, garantindo a aplicação adequada das práticas ágeis.

4.1.3. **Analistas de Sistemas**: Responsáveis por entender os requisitos dos usuários e traduzir esses requisitos em especificações técnicas.

4.1.4. **Designers de UI/UX**: Responsáveis pelo design da interface do usuário, garantindo que o sistema seja intuitivo e fácil de usar.

4.1.5. **Testadores de Software**: Equipe dedicada à validação e verificação do sistema, garantindo que ele funcione conforme o esperado e esteja livre de falhas.

4.1.6. **Escritores:** Responsáveis pela constituição dos documentos que detalham as funcionalidades do sistema, os requisitos, interações e mapeamento do progresso do sistema.

4.2. Recursos Tecnológicos

4.2.1. **Ambiente de Desenvolvimento**: Ferramentas e ambiente para desenvolvimento em C#, utilizando o framework Blazor para a criação de interfaces web interativas e responsivas com o auxílio do ASP.NET. Para o frontend, além da edição visual usando o CSS, também usaremos o bootstrap para responsividade. Além desses, o Visual Studio e o repositório disponível no GitHub.

4.2.2. **Servidores de Aplicação e Banco de Dados**: Infraestrutura necessária para hospedar o sistema e armazenar dados, garantindo disponibilidade e segurança, no momento, o sistema está sendo hospedado localmente em um servidor PostgreSQL

e está sendo mapeado automaticamente usando o Entity Framework Core

4.2.3. **Ferramentas de Modelagem:** Ferramentas que auxiliam na prototipagem e na criação dos diagramas usados. Para a modelagem do sistema, diagramas de caso e uso e diagrama de classe foi usada a Astah UML; Para o modelo lógico do banco de dados, o BRModelo, como ferramenta auxiliar na prototipação, o Figma; E, para a elaboração das Redes de Petri, utilizaremos a PIPE.

4.2.4. **Ferramentas de Gestão de Projetos**: Softwares para acompanhar o progresso das tarefas, sprints, e backlog do projeto, no momento a organização está sendo feita através de branchs dentro do GitHub e Planilhas do Google.

4.2.5. **Ferramentas de Comunicação**: Plataformas para facilitar a comunicação entre a equipe, no momento, para reuniões será utilizado o Google Meet, e para troca de mensagens e documentos, o GitHub e o Whatsapp.

4.3. Recursos de Integração

4.3.1. **Ferramentas de Automação de Testes**: Para realizar testes automáticos e garantir a qualidade do código ao longo do desenvolvimento, utilizaremos as seguintes: Selenium, NUnit, Lighthouse, DevTools e SonarLint, valendo salientar que os resultados mais importantes serão filtrados e tais recursos podem mudar com o tempo.

4.3.2. **Planos de Teste**: Com o uso da ferramenta citada anteriormente, a cada implementação será executado um novo teste e o registro dos seus resultados para futuras alterações e garantia de qualidade.

4.3.3. **Revisões de Código**: Toda semana, juntamente com as reuniões, o progresso e estado do código será revisado em caso de erros não identificados pelo programa de testes, no caso, erros referentes a má interpretação e falhas humanas. Isto irá garantir que o objetivo do ASA seja cumprido com excelência.

5. Metodologia

Este projeto seguirá uma metodologia formal baseada em etapas claramente definidas para garantir que os requisitos funcionais e não funcionais sejam atendidos.

5.1. Etapas de Especificação

5.1.1. **Levantamento dos Requisitos:** Coleta e análise de requisitos.
5.1.2. **Modelagem com Redes de Petri:** Criação de modelos que representam os processos de agendamento, serviços e gerenciamento de usuário.
5.1.3. **Verificação e Validação:** Análise dos modelos com o objetivo de garantir consistência e ausência de conflitos.
5.1.4. **Refinamento e Transformação:** Ajustes nos modelos e preparação para a implementação.
5.1.5. **Implementação:** Desenvolvimento do sistema com base nos modelos formais.
5.1.6. **Teste e Validação:** Testes para garantir que o sistema atenda aos requisitos especificados.
5.1.7. **Entrega e Manutenção:** Entrega do sistema e suporte contínuo.

6. Métodos e Linguagem Formal

6.1. Método Formal

Após a análise das abordagens de especificação apresentadas em sala de aula, o método de especificação formal adotado será com base em **Code Contracts**.

Os Code Contracts oferecem uma abordagem clara e eficiente para definir invariantes, pré-condições e pós-condições nos métodos e classes do sistema, o que será essencial para nossa equipe composta de duas disciplinas diferentes e com níveis de experiência em engenharia de software distintos.

A especificação explícita das condições que devem ser atendidas antes e depois da execução de uma função facilita a compreensão da lógica do sistema de agendamento de serviços. Além disso, os Code Contracts são eficazes para garantir que os métodos sigam um comportamento previsível e correto, ajudando a evitar erros de lógica no sistema. Algumas vantagens do seu uso são:

- **Verificação Estática e Dinâmica:** Permite a verificação de contratos durante o desenvolvimento, tanto estática quanto dinamicamente, ajudando a detectar possíveis violações de contrato antes mesmo da execução do código, reduzindo o número de falhas durante o processo de desenvolvimento.

- **Documentação Automática:** Funcionamento semelhante a uma "documentação viva", visto que as pré-condições, pós-condições e invariantes são expressas diretamente no código, tornando-o autoexplicativo para desenvolvedores.
- **Garantia de Corretude:** A especificação formal através de contratos garante que os métodos e as classes se comportem conforme o esperado, assegurando maior robustez e previsibilidade no sistema de agendamento.
- **Modularidade:** Pode ser aplicado de forma modular e escalável, permitindo que partes do sistema sejam modificadas ou expandidas sem comprometer a consistência global.
- **Facilidade de Evolução:** À medida que novos requisitos surgem ou o sistema evolui, é fácil ajustar os contratos para refletir novas condições ou comportamentos esperados, mantendo a flexibilidade.

Apesar das vantagens, os Code Contracts podem apresentar alguns desafios, que precisam ser considerados:

1. **Sobrecarga de Desenvolvimento:** A criação de contratos explícitos para métodos e classes pode aumentar o tempo de desenvolvimento, especialmente em sistemas complexos com muitas condições a serem especificadas. Para resolver isso, será priorizada a criação de contratos para as partes críticas do sistema, como o cadastro e a gestão de serviços, o restante terá uma aplicação menos rígida ou nenhuma.
2. **Impacto na Performance:** A execução de verificações de contratos em tempo de execução pode impactar levemente a performance do sistema, especialmente se houver muitas verificações, portanto, a verificação só existirá durante o desenvolvimento e testes.
3. **Complexidade em Cenários de Concorrência:** Embora Code Contracts ajudem na verificação de corretude, eles não lidam diretamente com problemas complexos de concorrência, como condições de corrida ou deadlocks. Neste caso, se for necessário, utilizaremos as Redes de Petri como uma segunda verificação, pois sua visualização modular e visual é perfeita para cobrir essa falta.

Para a especificação formal do sistema de agendamento de serviços usando Code Contracts, serão incluídos:

- A definição de contratos que asseguram as condições necessárias para o cadastro e edição de serviços;
- A especificação formal das restrições e condições que devem ser atendidas pelo sistema, como a prevenção de conflitos de horários.

Por fim, a escolha dos Code Contracts foi fundamentada nas vantagens de especificação explícita, verificação formal, modularidade e facilidade de evolução.

6.2. Especificação Formal

6.2.1. Requisitos Funcionais

Agendamento

0. [RF] - O sistema deve permitir que usuários **façam login** em suas contas.

☑ Alto	☐ Médio	☐ Baixo

Disponível em DiscenteService.cs:

```
public async Task<LoginResponseDto> LoginDiscenteAsync(LoginDiscente login)
{
    if (login == null) throw new ArgumentNullException(nameof(login));

    Contract.Requires(login.Email != null, "Email não pode ser nulo ou vazio.");
    Contract.Requires(login.Senha != null, "Senha não pode ser nula ou vazia.");

    var discente = await _context.Discentes.SingleOrDefaultAsync(d => d.Email == login.Email);

    if (discente == null || string.IsNullOrEmpty(discente.Senha) || string.IsNullOrEmpty(discente.Salt))
    {
        return null;
    }

    if (!VerificarSenha(login.Senha, discente.Senha, discente.Salt))
    {
        return null;
    }

    var token = GerarTokenJwt(discente.IdDiscente.ToString(), discente.Email ?? string.Empty);

    return new LoginResponseDto
    {
        UserId = discente.IdDiscente.ToString(),
        Token = token
    };
}
```

```
public async Task<LoginResponseDto> LoginProfissionalAsync(LoginProfissional login)
{
    if (login == null) throw new ArgumentNullException(nameof(login));

    Contract.Requires(login.Email != null, "Email não pode ser nulo ou vazio.");
    Contract.Requires(login.Senha != null, "Senha não pode ser nula ou vazia.");

    var profissional = await _context.Profissionais.SingleOrDefaultAsync(p => p.Email == login.Email);

    if (profissional == null || string.IsNullOrEmpty(profissional.Senha) || string.IsNullOrEmpty(profissional.Salt))
    {
        return null;
    }

    if (!VerificarSenha(login.Senha, profissional.Senha, profissional.Salt))
    {
        return null;
    }

    var token = GerarTokenJwt(profissional.IdProfissional.ToString(), profissional.Email ?? string.Empty);

    return new LoginResponseDto
    {
        UserId = profissional.IdProfissional.ToString(),
        Token = token
    };
}
```

1. [RF] - O sistema deve permitir que usuários do tipo discente **solicitem** o agendamento dos serviços.

☑ Alto	☐ Médio	☐ Baixo

Disponível em AgendamentoService.cs:

```
public async Task<Agendamento> SolicitarAgendamentoAsync(SolicitarAgendamentoDto dto)
{
    Contract.Requires(dto != null, "O objeto de solicitação de agendamento não pode ser nulo.");
    Contract.Requires(dto.DiscenteId > 0, "O ID do discente deve ser válido.");
    Contract.Requires(dto.HorarioId > 0, "O ID do horário deve ser válido.");
    Contract.Requires(dto.ProfissionalId > 0, "O ID do profissional deve ser válido.");
    Contract.Requires(dto.ServicoId > 0, "O ID do serviço deve ser válido.");
    Contract.Requires(dto.Data != default(DateTime), "A data do agendamento deve ser válida.");
    Contract.Requires(!string.IsNullOrEmpty(dto.Status), "O status do agendamento não pode ser nulo ou vazio.");

    // Verifica se o horário está disponível
    var horarioDisponivel = await _context.HorarioDisponivel
        .FirstOrDefaultAsync(h => h.IdHorario == dto.HorarioId && h.ProfissionalId == dto.ProfissionalId);

    if (horarioDisponivel == null)
    {
        return null; // Horário indisponível
    }

    // Cria um novo agendamento
    var novoAgendamento = new Agendamento
    {
        Data = dto.Data,
        DiscenteId = dto.DiscenteId,
        HorarioId = dto.HorarioId,
        ProfissionalId = dto.ProfissionalId,
        ServicoId = dto.ServicoId,
        Status = dto.Status
    };
```

```
    _context.Agendamento.Add(novoAgendamento);
    await _context.SaveChangesAsync();

    Contract.Ensures(novoAgendamento != null, "O novo agendamento deve ter sido criado com sucesso.");

    return novoAgendamento;
}
```

2. [RF] - O sistema deve permitir que os usuários **visualizem** os detalhes do agendamento cadastrado.

☐ Alto	☑ Médio	☐ Baixo

Disponível em AgendamentoService.cs:

```
public async Task<List<Agendamento>> ListarAgendamentosPorDiscenteAsync(int discenteId)
{
    Contract.Requires(discenteId > 0, "O ID do discente deve ser maior que zero.");

    return await _context.Agendamento
        .Where(a => a.DiscenteId == discenteId)
        .ToListAsync();
}
```

3. [RF] - O sistema deve permitir que usuários do tipo discente e do tipo profissional **cancelem** um agendamento antes da data marcada.

☐ Alto	☑ Médio	☐ Baixo

Disponível em AgendamentoService.cs:

```
public async Task<bool> CancelarAgendamentoAsync(int agendamentoId)
{
    Contract.Requires(agendamentoId > 0, "O ID do agendamento deve ser maior que zero.");

    var agendamento = await _context.Agendamento.FindAsync(agendamentoId);
    if (agendamento == null)
    {
        return false; // Agendamento não encontrado
    }

    _context.Agendamento.Remove(agendamento);
    await _context.SaveChangesAsync();

    Contract.Ensures(agendamento == null, "O agendamento deve ter sido removido do banco de dados.");

    return true;
}
```

4. [RF] - O sistema deve **notificar** o cancelamento do agendamento para as partes envolvidas na consulta. (**Não Implementado**)

☐ Alto	☐ Médio	☑ Baixo

5. [RF] - O sistema deve **notificar** a confirmação do agendamento para o discente no ato da aceitação e 24h antes da data marcada. (**Não Implementado**)

☐ Alto	☐ Médio	☑ Baixo

6. [RF] - O sistema deve permitir que usuários do tipo discente **visualizem** os horários disponíveis para cada serviço ofertado.

☑ Alto	☐ Médio	☐ Baixo

Disponível em AgendamentoService.cs:

```
public async Task<List<HorarioDisponivel>> ListarHorariosDisponiveisAsync(int profissionalId)
{
    Contract.Requires(profissionalId > 0, "O ID do profissional deve ser maior que zero.");

    return await _context.HorarioDisponivel
        .Where(h => h.ProfissionalId == profissionalId)
        .ToListAsync();
}
```

```
public async Task<IEnumerable<AgendamentoDto>> BuscarAgendamentosPorProfissionalAsync(int profissionalId)
{
    Contract.Requires(profissionalId > 0, "O ID do profissional deve ser maior que zero.");

    var agendamentos = await _context.Agendamento
        .Where(a => a.ProfissionalId == profissionalId && a.Data >= DateTime.Today)
        .Join(_context.HorarioDisponivel,
            a => a.HorarioId,
            h => h.IdHorario,
            (a, h) => new AgendamentoDto
            {
                IdAgendamento = a.IdAgendamento,
                Data = a.Data,
                DiscenteId = a.DiscenteId,
                ProfissionalId = a.ProfissionalId,
                ServicoId = a.ServicoId,
                HorarioId = a.HorarioId,
                HoraInicio = h.HoraInicio,
                HoraFim = h.HoraFim,
                Status = a.Status
            })
        .ToListAsync();

    return agendamentos;
}
```

Serviço

7. [RF] - O sistema deve **conter** os serviços de psicólogo, pedagogo, nutricionista e assistente social para a comunidade acadêmica. (**Banco de Dados)**

☑ Alto	☐ Médio	☐ Baixo

8. [RF] - Os serviços de psicólogo e nutricionista devem **disponibilizar** as opções de consulta e retorno. (**Banco de Dados**)

☐ Alto	☑ Médio	☐ Baixo

9. [RF] - Um serviço poderá ter mais de um profissional disponível. (**Banco de Dados**)

☐ Alto	☐ Médio	☑ Baixo

Usuários

10. [RF] - O sistema deve permitir o **cadastro** de usuários por meio de um e-mail e de uma senha.

☑ Alto	☐ Médio	☐ Baixo

11. [RF] - O sistema deve permitir após a aprovação do e-mail e senha, realizar o **cadastro** das informações de Nome, Sobrenome, Matrícula e Telefone (opcional).

☑ Alto	☐ Médio	☐ Baixo

Disponível em DiscenteService.cs:

```
// RF010 e RF011
public async Task<Discente> RegistrarDiscenteAsync(RegistrarDiscente registro)
{
    // Verificação de nulidade
    if (registro == null) throw new ArgumentNullException(nameof(registro));

    Contract.Requires(registro.Nome != null, nameof(registro.Nome));
    Contract.Requires(registro.Email != null, nameof(registro.Email));
    Contract.Requires(
        registro.Email.EndsWith("@alunos.ufersa.edu.br"),
        "O Email deve conter o domínio '@alunos.ufersa.edu.br'.");
    Contract.Requires(registro.Senha != null, nameof(registro.Senha));
    Contract.Requires(registro.Matricula != null, nameof(registro.Matricula));

    // Criptografar a senha e gerar o salt
    var (senhaCriptografada, salt) = CriptografarSenha(registro.Senha);

    var discente = new Discente
    {
        Nome = registro.Nome,
        Email = registro.Email,
        Senha = senhaCriptografada,
        Salt = salt, // Armazenar o salt no banco
        Matricula = registro.Matricula,
        Telefone = string.IsNullOrEmpty(registro.Telefone) ? null : registro.Telefone,
        Curso = registro.Curso
    };
```

```
    _context.Discentes.Add(discente);
    await _context.SaveChangesAsync();

    Contract.Ensures(discente != null, "O objeto Discente não deve ser nulo ao final do método.");
    Contract.Ensures(discente.Email.EndsWith("@alunos.ufersa.edu.br"), "O email deve conter o domínio correto após a inserção

    return discente;
}
```

```
// RF010 e RF011
public async Task<Profissional> RegistrarProfissionalAsync(RegistrarProfissional registro)
{
    if (registro == null) throw new ArgumentNullException(nameof(registro));

    Contract.Requires(registro.Nome != null, nameof(registro.Nome));
    Contract.Requires(registro.Email != null, nameof(registro.Email));
    Contract.Requires(
        registro.Email.EndsWith("@ufersa.edu.br"),
        "O Email deve conter o domínio '@ufersa.edu.br'.");
    Contract.Requires(registro.Senha != null, nameof(registro.Senha));
    Contract.Requires(registro.ServicoId != null, nameof(registro.ServicoId));

    // Verificar se o ServicoId é válido
    var servicoExiste = await _context.ServicoDisponivel.AnyAsync(s => s.IdServico == registro.ServicoId);
    if (!servicoExiste)
    {
        throw new ArgumentException("O serviço fornecido não é válido.");
    }

    var (senhaCriptografada, salt) = CriptografarSenha(registro.Senha);

    var profissional = new Profissional
    {
        Nome = registro.Nome,
        Email = registro.Email,
        Senha = senhaCriptografada,
        Salt = salt,
        ServicoId = registro.ServicoId, // Associar o ServicoId ao profissional
        Descricao = registro.descricao

    };
```

```
    _context.Profissionais.Add(profissional);
    await _context.SaveChangesAsync();

    Contract.Ensures(profissional != null, "O objeto Profissional não deve ser nulo ao final do método.");
    Contract.Ensures(profissional.Email.EndsWith("@ufersa.edu.br"), "O email deve conter o domínio correto após a

    return profissional;
}
```

12. [RF] - O sistema deve permitir que usuários, uma vez cadastrados, possam **alterar** a senha e seu perfil.

☐ Alto	☑ Médio	☐ Baixo

Disponível em DiscenteService.cs:

```
public async Task<bool> AtualizarPerfilAsync(AtualizarPerfilDto atualizarPerfil)
{
    // Buscando o discente no banco de dados
    var usuarioDiscente = await _context.Discentes.SingleOrDefaultAsync(d => d.Email == atualizarPerfil.Email);

    Contract.Requires(atualizarPerfil != null, "O objeto atualizarPerfil não pode ser nulo.");

    if (usuarioDiscente != null)
    {
        // Atualizando campos que não são nulos
        if (!string.IsNullOrEmpty(atualizarPerfil.Nome))
        {
            usuarioDiscente.Nome = atualizarPerfil.Nome;
        }
        // Atualizando o telefone, que agora é uma string
        if (!string.IsNullOrEmpty(atualizarPerfil.Telefone))
        {
            usuarioDiscente.Telefone = atualizarPerfil.Telefone;
        }

        // Salvar mudanças no banco de dados
        await _context.SaveChangesAsync();
        return true;
    }

    Contract.Ensures(usuarioDiscente != null, "O perfil do usuário foi atualizado com sucesso.");
    Contract.Ensures(usuarioDiscente.Nome != null, "O perfil do usuário foi atualizado com sucesso.");

    return false; // Se o usuário não for encontrado
}
```

```
public async Task<bool> AlterarSenhaAsync(AlterarSenhaDto alterarSenha)
{
    // Buscando em ambas as tabelas: Discentes e Profissionais separadamente
    var usuarioDiscente = await _context.Discentes.SingleOrDefaultAsync(d => d.Email == alterarSenha.Email);
    var usuarioProfissional = await _context.Profissionais.SingleOrDefaultAsync(p => p.Email == alterarSenha.Email);

    Contract.Requires(alterarSenha != null, "O objeto alterarSenha não pode ser nulo.");

    if (usuarioDiscente != null && VerificarSenha(alterarSenha.SenhaAtual, usuarioDiscente.Senha, usuarioDiscente.Salt))
    {
        var (novaSenhaCriptografada, novoSalt) = CriptografarSenha(alterarSenha.NovaSenha);

        usuarioDiscente.Senha = novaSenhaCriptografada;
        usuarioDiscente.Salt = novoSalt;

        await _context.SaveChangesAsync();

        Contract.Ensures(usuarioDiscente.Senha != null, "A senha do usuário foi atualizada com sucesso.");

        return true;
    }
    else if (usuarioProfissional != null && VerificarSenha(alterarSenha.SenhaAtual, usuarioProfissional.Senha, usuarioPro
    {
        var (novaSenhaCriptografada, novoSalt) = CriptografarSenha(alterarSenha.NovaSenha);

        usuarioProfissional.Senha = novaSenhaCriptografada;
        usuarioProfissional.Salt = novoSalt;
```

```
        await _context.SaveChangesAsync();

        Contract.Ensures(usuarioProfissional.Senha != null, "A senha do usuário foi atualizada com sucesso.");

        return true;
    }

    return false; // Nenhum usuário encontrado ou senha incorreta
}
```

13. [RF] - Cada usuário do tipo profissional deve estar ligado a um único serviço específico. (**Banco de Dados**)

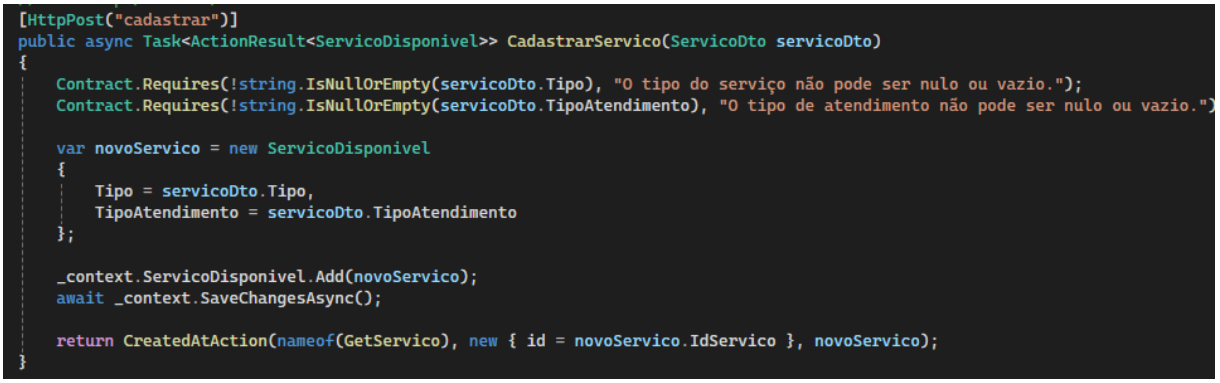
☐ Alto	☑ Médio	☐ Baixo

Recursos

14. [RF] - O sistema deve permitir que usuários do tipo profissional **cadastrem** informações acerca do serviço prestado.

☑ Alto	☐ Médio	☐ Baixo

Disponível em ServicoController.cs:



```
[HttpPost("cadastrar")]
public async Task<ActionResult<ServicoDisponivel>> CadastrarServico(ServicoDto servicoDto)
{
    Contract.Requires(!string.IsNullOrEmpty(servicoDto.Tipo), "O tipo do serviço não pode ser nulo ou vazio.");
    Contract.Requires(!string.IsNullOrEmpty(servicoDto.TipoAtendimento), "O tipo de atendimento não pode ser nulo ou vazio.")

    var novoServico = new ServicoDisponivel
    {
        Tipo = servicoDto.Tipo,
        TipoAtendimento = servicoDto.TipoAtendimento
    };

    _context.ServicoDisponivel.Add(novoServico);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetServico), new { id = novoServico.IdServico }, novoServico);
}
```

15. [RF] - Os horários disponíveis para atendimento devem ser definidos pelo profissional.

☑ Alto	☐ Médio	☐ Baixo

6.2.2. Requisitos não funcionais

Usabilidade

1. [RNF] - **Estética da interface do usuário:** O sistema deve apresentar uma interface com cores e tipografia padronizados, além de ícones personalizados, obedecendo ao contexto de uso.

☑ Alto	☐ Médio	☐ Baixo

2. [RNF] - **Proteção ao erro do usuário:** O sistema deve retornar um alerta/mensagem afirmando sucesso ou fracasso ao finalizar as operações no sistema.

☐ Alto	☑ Médio	☐ Baixo

Segurança

3. [RNF] - **Integridade:** O sistema deve criptografar a senha dos usuários para armazená-la em seu banco de dados.

☑ Alto	☐ Médio	☐ Baixo

4. [RNF] - **Confidencialidade:** O sistema deve garantir a segurança dos dados pessoais dos discentes e profissionais.

☑ Alto	☐ Médio	☐ Baixo

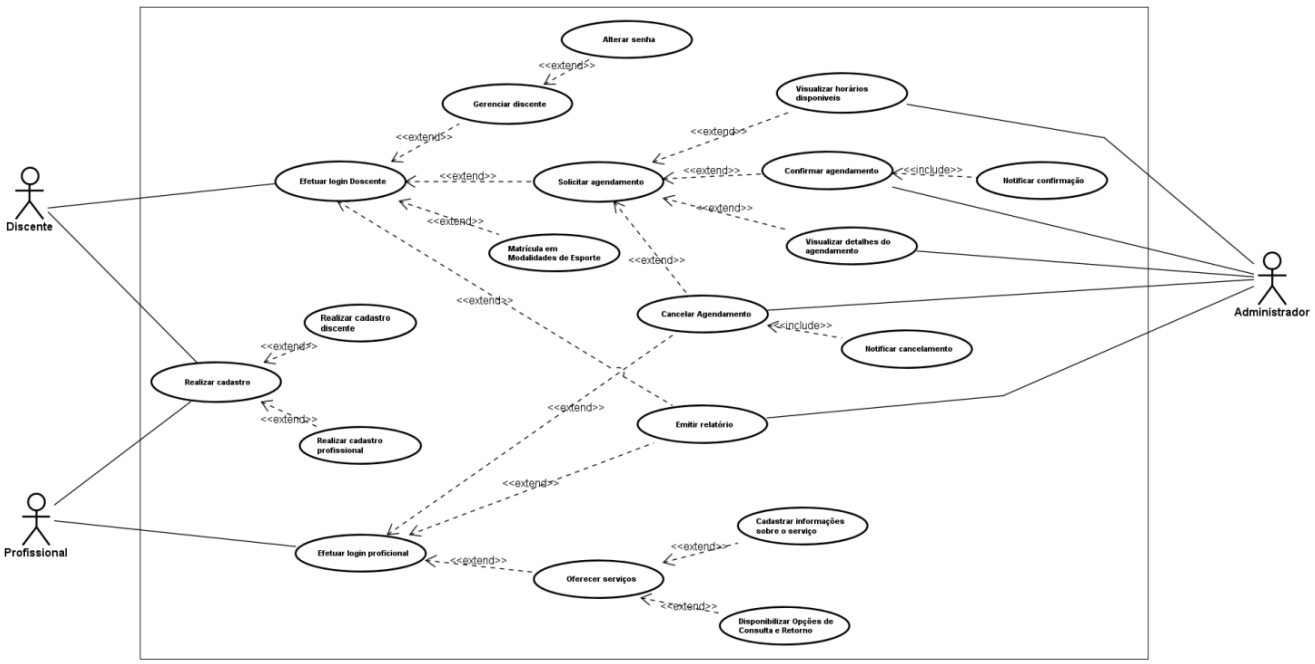
Portabilidade

5. [RNF] - **Adaptabilidade:** O sistema deve ser acessível por meio de um navegador web, programado para se adaptar a diferentes plataformas (Chrome, Firefox e Edge).

☐ Alto	☐ Médio	☑ Baixo

7. Cenários de Uso

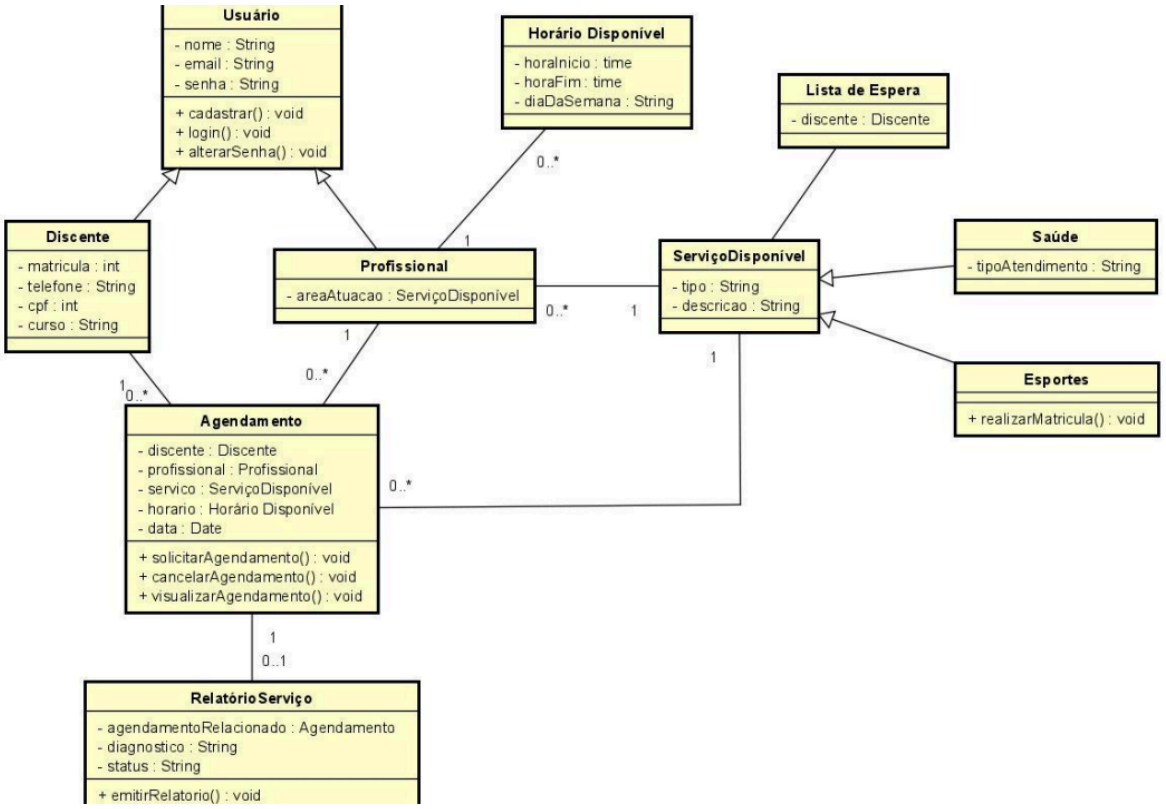
7.1. Diagramas de Caso de Uso



7.2. Detalhamento de Caso e Uso

O detalhamento está disponível a partir **deste link**.

7.3. Diagrama de Classes



8. Cronograma

O cronograma de desenvolvimento do sistema é mantido nessa planilha, na qual são registradas todas as datas de alterações realizadas e o progresso contínuo do projeto. Esse documento serve como um acompanhamento formal das etapas, permitindo monitorar o desenvolvimento de maneira clara e organizada.

Além disso, estão sendo realizadas reuniões semanais aos domingos, nas quais os objetivos para a semana seguinte são discutidos. Durante essas reuniões, novos objetivos são definidos com base nas necessidades e avanços do projeto e são imediatamente integrados ao cronograma. Esse processo de atualização constante garante que o planejamento reflita com precisão o estado atual do sistema, facilitando a adaptação a novas demandas e mudanças.