## REQUISITOS FUNCIONAIS

Os requisitos funcionais é uma maneira de avaliar “O que” o sistema deve fazer, isto é, quais são suas características de acordo com entradas, processamentos e saídas esperadas em diferentes condições. Abaixo serão listados todos os requisitos funcionais do sistema:

- **Cadastrar professores e equipamentos:** O sistema deve permitir que dados essenciais de professores, como: Nome, Disciplina e Turmas; sejam inseridas em campos de texto do tipo **TextBox**. Assim poderão ser salvos ao clicar em um botão do tipo **Button** em um arquivo XML Professor.xml. Da mesma forma os equipamentos, que requere os dados Equipamento, Tipo e Quantidade, dos tipos TextBox, sendo este último como formato numérico (Tipo **NumericUpDown**), salvando os dados pela classe XDocument em um arquivo Equipamento.xml. Antes do registro, no caso do Professor, o campo Nome deve ser verificado via Regex, que verifica se o campo contém nome e sobrenome separados por espaços, se não, ele exibe um erro na tela. Da mesma forma se algum dos campos estiverem vazios, em ambos os casos (Professor e Equipamento), um erro deve ser apresentado em MessageBox (Consultar Apêndice A). Os métodos envolvidos para esse fim são: Professor.Registrar() e Equipamento.Registrar(), após as instâncias destas classes.

- **Agendar empréstimos em datas livres e poder visualizá-los:** O sistema inicialmente deve recolher os dados Professor, Equipamento, Data Inicial, Hora Inicial, Data Final e Hora Final, todos do tipo **TextBox** e verifica se estão vazios, se estão, apresentar um erro em MessageBox (Consultar Apêndice B), se não estão, prosseguir verificando o formato de datas e horas utilizando Regex. Tanto a data e a hora, como o Nome, passam por expressões regulares diferentes e se estes testes não passarem, também apresentar um erro, senão, verificar a disponibilidade de datas pelo Controller Agendamento.VerificarDisponibilidade() que é uma classe estática que vai retornar verdadeiro se a data inicial e final passadas como parâmetro estiverem livres para agenda. Se retornar falso, uma mensagem é apresentada dizendo que as datas escolhidas não são possíveis, se não, a instância de empréstimo é criada usando os parâmetros fornecidos pelo usuário e após isto, executado o método Emprestimo.Registrar(). Os dados são salvos no arquivo Emprestimo.xml utilizando a classe XDocument. Em outras janelas, o usuário deve ser capaz de filtrar e listar estes dados, lidos do mesmo arquivo, usando a classe DataGridView e o método Coletar();

- **Método estático para gerenciamento de datas livres:** Agendamento é uma classe estática do controlador que comporta métodos estáticos como: VerificarDisponibilidade() e AtribuirDisponibilidade(), onde este último recebe a sobrecarga (overloading), um incluindo parâmetro de Data inicial e final e o outro sem parâmetros. O 1ª método de verificação, executa Emprestimo.ColetarLivres() para capturar todos os dados XML do arquivo Livres.xml e em seguida, realiza um loop filtrando cada Item desse “Array” de elementos XML. Para cada item, é verificado se a data final contém o termo “Data Indefinida”, se sim, ele identifica se as datas passadas estão “A partir” do intervalo de data inicial livre, caso esteja, ele apenas verifica se data inicial é diferente da data inicial livre, ou é igual, no 1ª caso um novo elemento XML é adicionado com a data indefinida, alterando a data indefinida anterior para uma data inicial, no 2ª caso a data inicial livre é modificada com a data final do usuário. Porém, se a data passada não está a partir do intervalo inicial, ele será menor, então ele continua o loop a procura de novas datas, definindo horarioLivre para “False” temporariamente (esse valor pode ser retornado no final). No entanto, se a data final não contém o termo “Data Indefinida”, esta será uma faixa de valores a ser verificados, tudo o que o código faz é identificar se as datas passadas estão “Entre” esta faixa, podendo utilizar quase a mesma lógica de modificação ou criação de elemento de XML a depender da diferença ou igualdade das datas, da mesma forma, atribuir modificações em ordens inversas. Em um dos casos, a data livre atual pode ser removida, se caso não houver mais uma faixa livre entre elas. Se em todas verificações do loop não houver uma faixa disponível, a variável horarioLivre=false é retornada no fim. AtribuirDisponibilidade() pode registrar uma data livre padrão do dia de hoje ou atribuir uma nova data livre imediatamente.

- **Empréstimos devem ser cancelados e fechados, os fechados podem ser monitorados:** Assim como os outros elementos, o cancelamento e fechamento dependem de operações da interface. O cancelamento por exemplo, assim como o fechamento, atribui um evento de tecla para um campo de texto (Nome ou Item) em janelas de listagem, caso for ENTER, é inicializado um menu do tipo **ContextMenuStrip** que é aberto e listado dados vindo da coleta de Professor ou Equipamento, isto é, Professor.Coletar() ou Equipamento.Coletar(). O conjunto retornado em uma das classes, será listada no menu e atribuída evento de clique (sendo do mouse ou teclado), e este evento vai utilizar a classe **DataGridView** para listar em formato de tabela dados específicos pelo método Emprestimo.Verificar(), onde a instância deve conter como argumento o “texto do item”, que pode ser o nome do equipamento clicado ou o nome do professor. Em cada linha da tabela filtrada, será atribuída o ID de cada elemento XML vindo de Emprestimo.xml. Este ID será utilizado pelo evento de clique do botão Cancelar ou Fechar, que instanciará novamente a classe

Emprestimo com o ID como parâmetro, chamando o método Cancelar() ou Fechar(). No método Cancelar(), o Emprestimo.xml é carregado e o dado de registro cujo ID for o especificado será retornado, logo em seguida, os atributos Data Inicial e Data Final da classe Emprestimo pegará os valores destas mesmas tags neste registro retornado e AtribuirDisponibilidade() do Controller é chamado pra armazenar imediatamente a nova data livre (a data cancelada), Após isto, o método Excluir() é chamado. Já no caso do método Fechar(), o mesmo procedimento de Cancelar() é feito, com a exceção de não excluir o dado imediatamente após a atribuição de disponibilidade de data livre e passar não só as datas, mas também o nome e o equipamento, pois estes dados serão salvos para armazenar em Entregues.xml. Após armazenar neste arquivo usando Registrar(), o dado finalmente é excluído usando Excluir().

A partir destes requisitos, conseguimos compreender a essência do software, avaliando formas alternativas de se recorrer a um mesmo dado, usando polimorfismo e herança. As classes da camada de dados, relativos ao Model, como: Emprestimo, Professor e Equipamento; Herdam características da classe Cadastro por ter as mesmas funcionalidades principais: Registrar(), Excluir(), Verificar(), desta forma, sobrescrevendo estes métodos contendo suas próprias lógicas, que podem ser similares entre elas ou não. Da mesma forma, o método Verificar() é sobrecarregado, pode ter definições com parâmetros ou se parâmetros e similar a isto, os construtores de instanciação são sobrecarregados com maiores tipos de alternativas. É neste contexto que encontramos o conceito de polimorfismo para cadastro de dados. Não só os métodos herdados ou da camada de dados, mas Coletar() é um exemplo que não herda da classe Cadastro() sendo uma característica específica das classes, mas que é sobrecarregado com parâmetros, e seguindo esta lógica, a camada de negócios utiliza sobrecarga no método AtribuirDisponibilidade, podendo ter até duas formas de realizar esta atribuição. Em seguida, veremos sobre os requisitos não-funcionais do software.