

TÓPICOS DE ENGENHARIA DE SOFTWARE

|  |
| --- |
| **Avaliação Continuada**  Professora: Juliana Bueno  Karine A. Ramos Jardim 112332  Marilia Paula da Silva 132818  Romário Lima 14000244 |
| Pedro Augusto 137758  Cassiano 14007086  Breno Moreira 14001449 |
|  |
|  |
|  |
|  |
|  |

DOCUMENTO DE REQUISITOS

**QUESTIONÁRIO DE ENTREVISTA**

1. Qual o ramo do negócio?
2. Como funciona o sistema atual?
3. Como funciona a relação com o fornecedor, existe algum método para cadastro?
4. Qual o método utilizado para controle do estoque?
5. Como funciona a logistica de entrega dos produtos?
6. Controle das vendas, qual o método utilizado?
7. Como funciona o cadastro dos clientes?
8. Qual o método utilizado para o cadastro dos produtos?
9. Existe vinculo com laboratorios?
10. Como funciona o cadastro de convenios?
11. Como funciona os registros dos funcionários?
12. Qual o propósito do novo sistema?

**REQUISITOS FUNCIONAIS**

**RF001:** Cadastrar medicamentos informando: nome, princípio ativo, fabricante, data de fabricação, data de vencimento, indicações de uso, classificação (tarja vermelha, preta, etc.).

**RF002:** Registrar a entrada no estoque de mercadorias através do código de barras, informando o número da Nota Fiscal do fornecedor.

**RF003:** Atualizar o estoque de mercadorias a medida em que ocorrer saída no caixa (venda).

**RF004:** Registrar nível mínimo de estoque de mercadorias.

**RF005:** Emitir relatório com os prazos de validade e quantidade dos medicamentos em estoque diariamente.

**RF006:** Solicitar assinatura digital do gerente para venda de medicamentos controlados.

**RF007:** Cadastrar fornecedores incluindo: número do CNPJ, endereço físico e eletrônico, contato telefônico, nome dos representantes, telefone dos representantes.

**RF008:** Cadastrar clientes que utilizam medicamentos controlados informando: CPF, data de nascimento, endereço físico e eletrônico, contato telefônico, nome e CRM do médico que assinou a receita de medicamento controlado.

**RF009:** Cadastrar clientes incluindo: CPF, endereço físico e eletrônico, contato telefônico, convênio e data de nascimento.

**RF010:** Gerar código de pedido de clientes assim que solicitada a entrega, informando os medicamentos solicitados, o endereço de entrega, o número de cadastro do cliente, a forma de pagamento a data de efetivação do pagamento.

**RF011:** Registrar a entrega dos medicamentos informando o código do pedido e a data de entrega.

**RF012:** Cadastrar empresas conveniadas informando: nome, CNPJ, endereço físico e eletrônico, nome do representante, contatos telefônicos.

**RF013:** Cadastrar lista de medicamentos com desconto das empresas conveniadas, utilizando o cadastro de medicamentos.

**RF014:** Cadastrar funcionários, informando CPF, data de nascimento, endereço físico e eletrônico, contato telefônico, função.

**RF015:** Cadastrar código de acesso (login) para funcionários através do CPF já registrado no cadastro de funcionários.

**RF016:** Cadastrar funções/cargos dos funcionários através da inserção do CPF já previamente cadastrado, informando a data de inicio das atividades, qual a função/cargo, turno de trabalho, salário base, gratificações, descontos, se utiliza auxílio-transporte ou não.

**RF017:** Controlar horário de entrada e saída de funcionário através do código de acesso de cada um.

**RF018:** Emitir relatório mensal de carga horária dos funcionários.

**RF019:** Cadastrar convênio para funcionários, indicando o percentual de desconto para cada tipo de medicamento.

**RF020:** Emitir relatório mensal de medicamentos adquiridos por funcionário.

**RF021:** Permitir a consulta dos dados dos clientes.

**RF022:** Gerar relatório com o fluxo de caixa com base em filtros de datas (dia, semana, mês, ano...).

**RF023:** Gerar relatório de venda de medicamentos com desconto.

**RF024:** Controlar os boletos a serem pagos aos fornecedores, com a inclusão da data de vencimento, banco responsável, nome e CNPJ da empresa credora.

**RF025:** Emitir relatório de pagamentos a serem efetuados semanalmente.

**RF026:** Emitir relatório de percentual pago as empresas de cartão de crédito e débito mensalmente, conforme vendas efetuadas com este sistema de pagamento.

**RF027:** Emitir relatório de valores recebidos de acordo com os filtros: pecúnia, cheque, cartão e de acordo com filtros de data (dia, semana, mês).

**RF028:** Emitir relatório de medicamentos não vendidos por mais de quinze dias, informando o nome, princípio ativo e laboratório.

**RF029:** Emitir relatório dos dez medicamentos mais vendidos no mês, informando o nome, princípio ativo e laboratório.

**RF030:** Emitir relatório de vendas dos funcionários com filtros de datas (dia, semana, mês), considerando quantidade de medicamentos vendidos e total pecuniário de retorno por preço de venda, através do número de registros efetuados pelo código de acesso do funcionário ao sistema.

**REQUISITOS NÃO FUNCIONAIS**

**RNF001:** Infraestrutura da rede e computadores devem ser padrão cliente-servidor

**RNF002:** A base de dados deve ser protegida para acesso apenas de usuários autorizados.

**RNF003:** O tempo de resposta do sistema não deve ultrapassar 30 segundos.

**RNF004:** O software deve ser operacionalizado no sistema windows.

**RNF005:** O sistema deve ser implementado em C# com banco de dados MySQL.

**RNF006:** Servidor configurado localmente.

**RNF007:** Robustez do sistema.

**RNF008:** Tempo de processamento.

**RNF009:** Os dados devem ser protegidos por criptografia.

**RNF010:** Linguagem do código de barras deve ser programada em ZPL.

**REGRAS DE NEGÓCIOS**

**RNG001:** Não será aceito ocadastro de medicamentos com vencimento inferior a 01 ano a contar da data de entrega.

**RNG002:** A entrada de mercadorias em estoque ocorrerá apenas com Notas Fiscais de Fornecedores previamente cadastrados.

**RNG003:** A atualização de estoque estará vinculada ao fechamento efetivo da venda, após a emissão da Nota Fiscal de saída de mercadorias.

**RNG004:** O registro de nível mínimo de estoque de mercadorias deve estar vinculado ao princípio ativo do medicamento e seu respectivo laboratório.

**RNG005:** Emitir relatório após o fechamento de todos os sistemas de pagamentos da farmácia.

**RNG006:** Identificado o lançamento de código de medicamento controlado o sistema bloqueie o fechamento da venda, liberando apenas após a inserção da assinatura digital do gerente.

**RNG007:** O cadastro de fornecedores estará vinculado ao site governamental para validar o CNPJ, caso o número não seja encontrado nos registros o sistema não permite a inserção de nenhum dado.

**RNG008:** Limitar o desconto de medicamentos a 20%.

**RNG009:** O controle de entrada e saída de funcionários valida de 15 em 15 minutos. Ex: se registrou a entrada 07h05minh o sistema gera 07h15minh

**RNG010:** A senha de acesso de funcionários deve possuir no mínimo 08 dígitos e deve incluir um caractere especial

**DIAGRAMAS DE CASO DE USO**

**RF001:** Cadastrar medicamentos informando: nome, princípio ativo, fabricante, data de fabricação, data de vencimento, indicações de uso, classificação (tarja vermelha, preta, etc.).

**RF004:** Registrar nível mínimo de estoque de mercadorias.

**RF006:** Solicitar assinatura digital do gerente para venda de medicamentos controlados.

**RF013:** Cadastrar lista de medicamentos com desconto das empresas conveniadas, utilizando o cadastro de medicamentos.

**RF015:** Cadastrar código de acesso (login) para funcionários através do CPF já registrado no cadastro de funcionários.

**RF016:** Cadastrar funções/cargos dos funcionários através da inserção do CPF já previamente cadastrado, informando a data de inicio das atividades, qual a função/cargo, turno de trabalho, salário base, gratificações, descontos, se utiliza auxílio-transporte ou não.

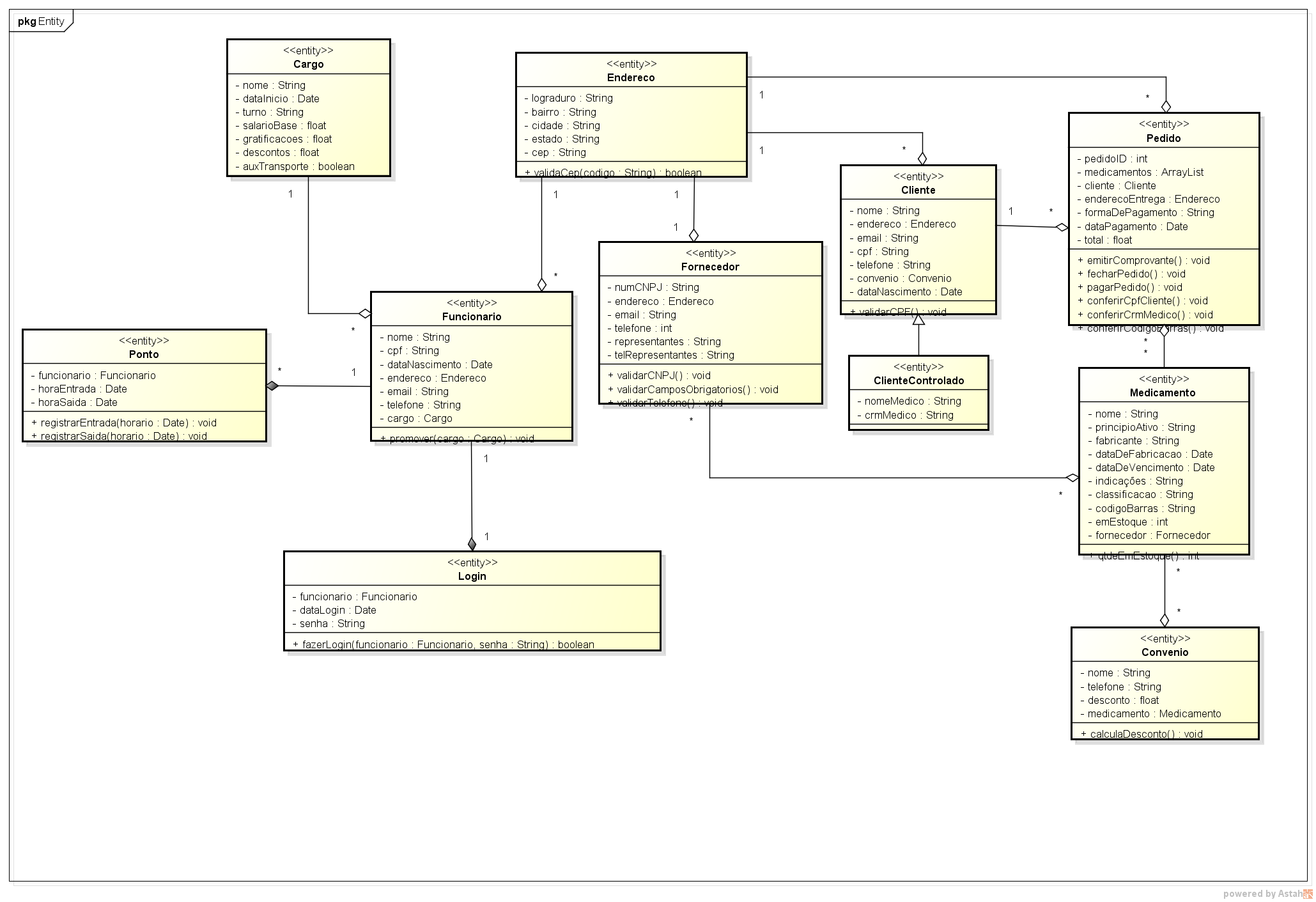
**RF017:** Controlar horário de entrada e saída de funcionário através do código de acesso de cada um.

**RF024:** Controlar os boletos a serem pagos aos fornecedores, com a inclusão da data de vencimento, banco responsável, nome e CNPJ da empresa credora.

**RF027:** Emitir relatório de valores recebidos de acordo com os filtros: pecúnia, cheque, cartão e de acordo com filtros de data (dia, semana, mês).

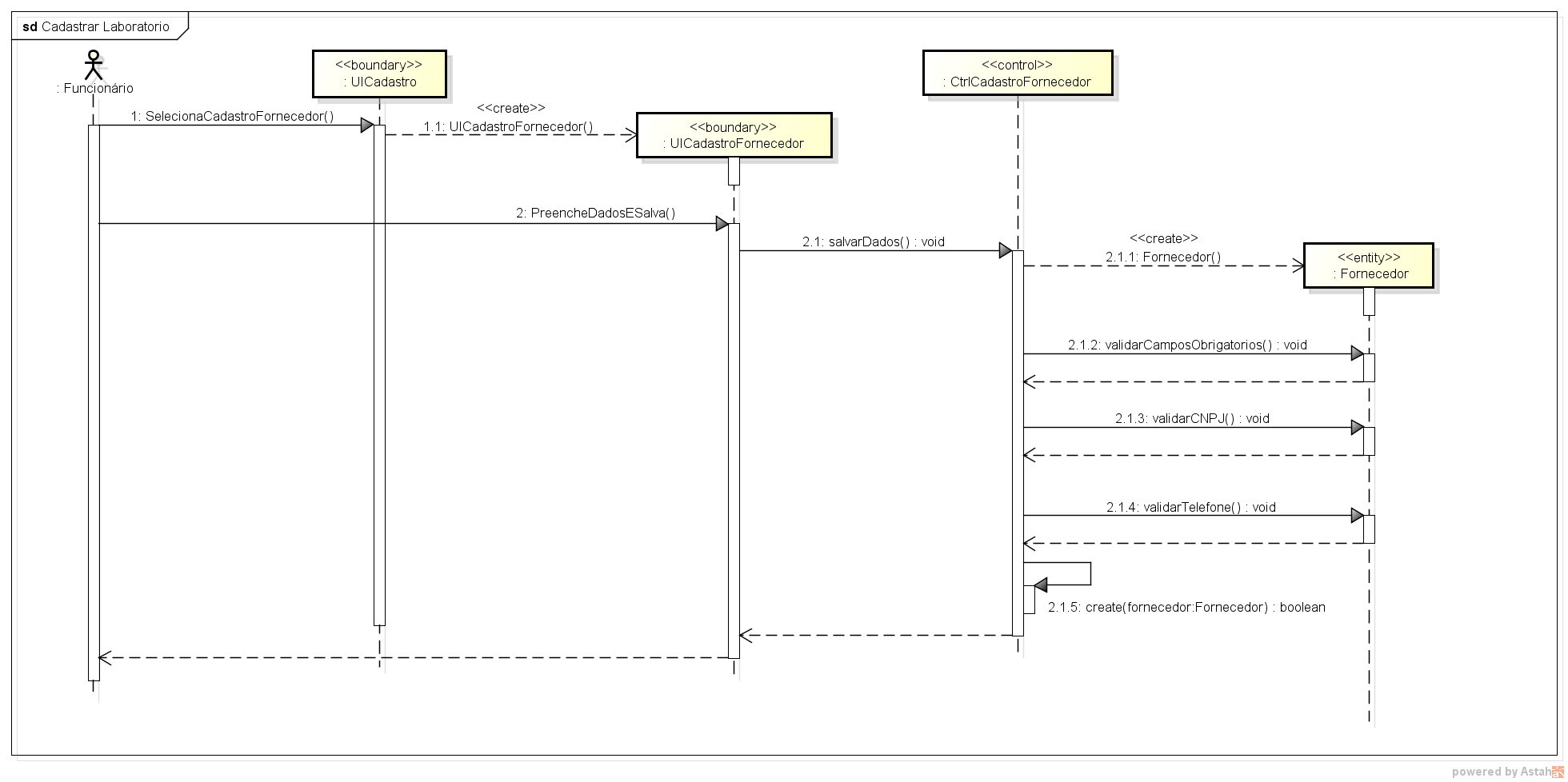
**RF029:** Emitir relatório dos dez medicamentos mais vendidos no mês, informando o nome, princípio ativo e laboratório.

**DIAGRAMAS DE CLASSES**

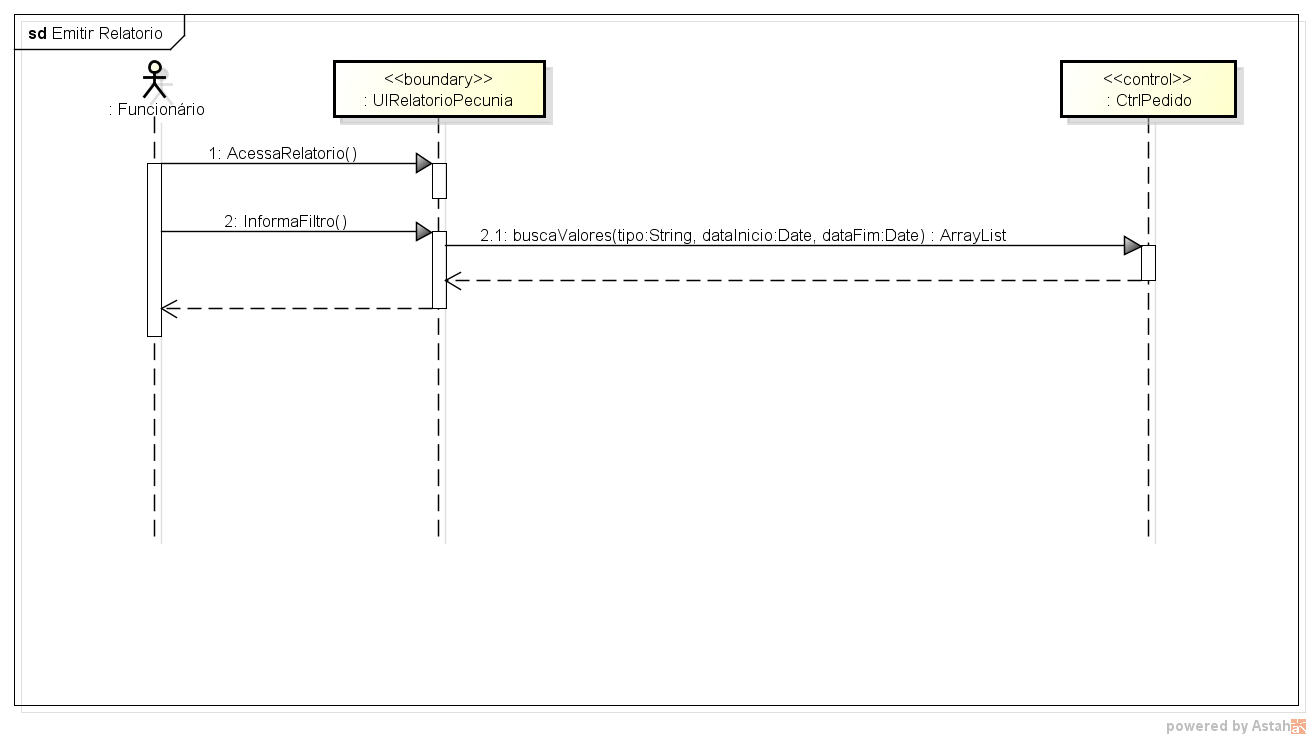
****

**DIAGRAMAS DE SEQUÊNCIA**

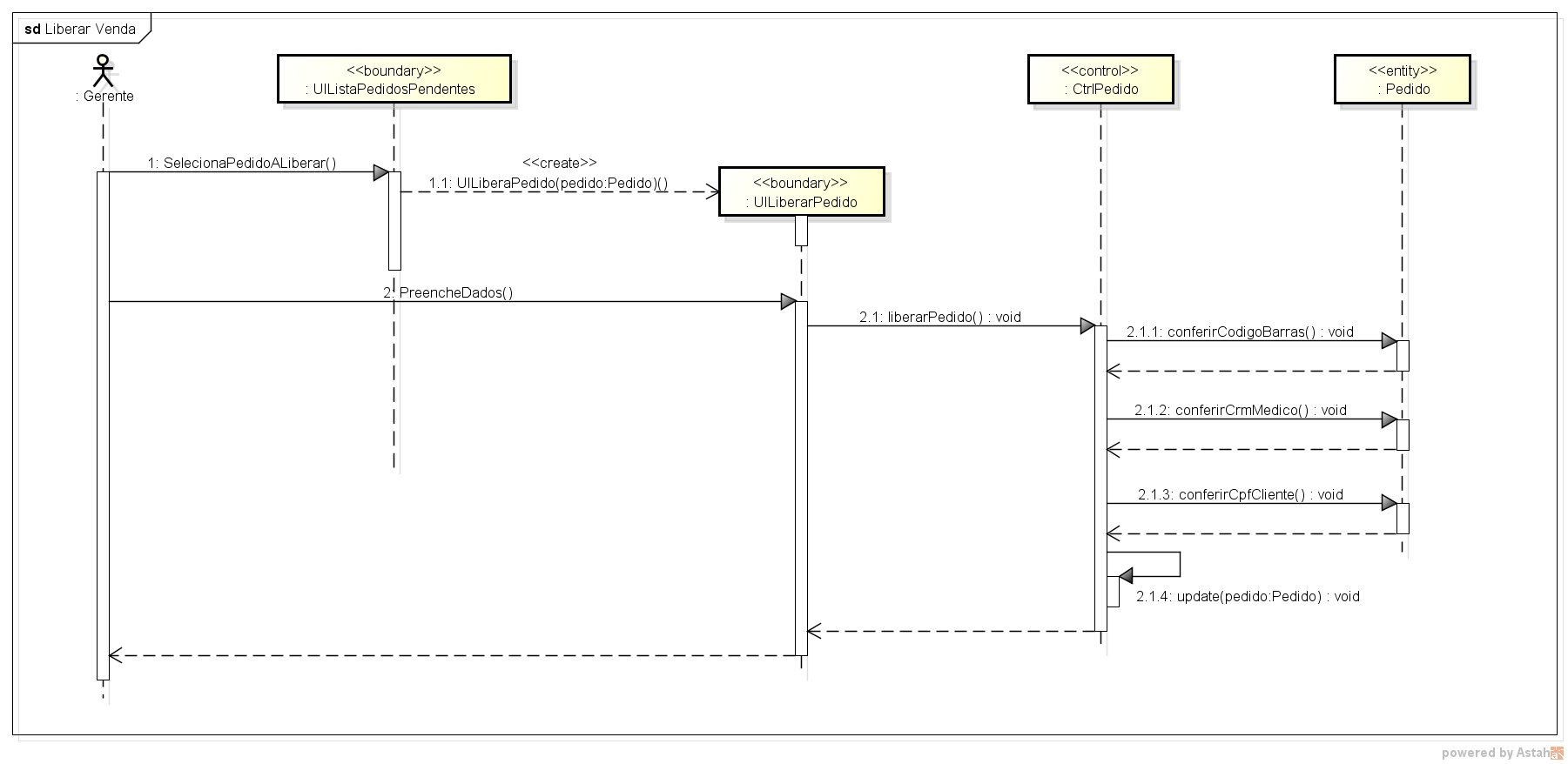
**Cadastrar Laboratório**

****

**Emitir Relatório Pecúnia**

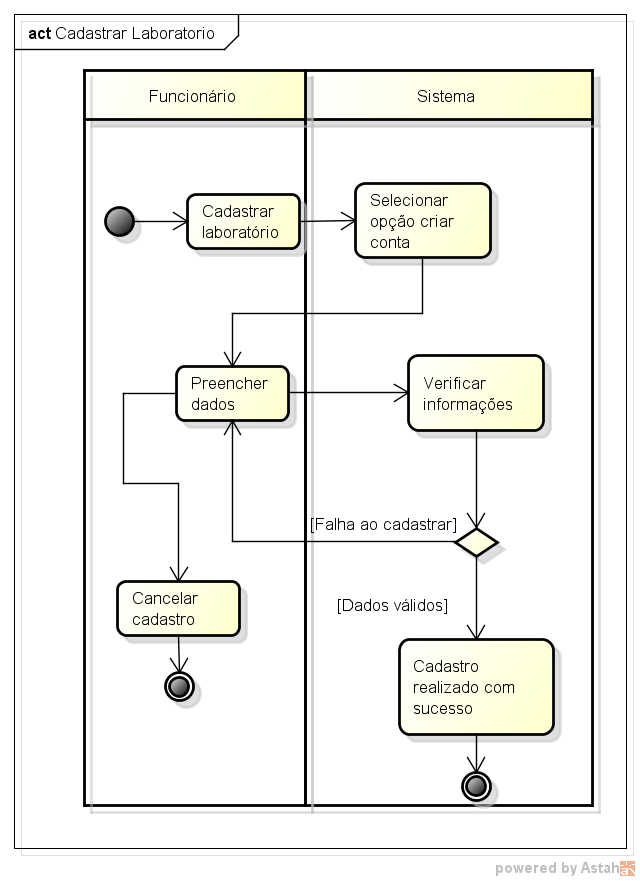
****

**Liberar Venda**

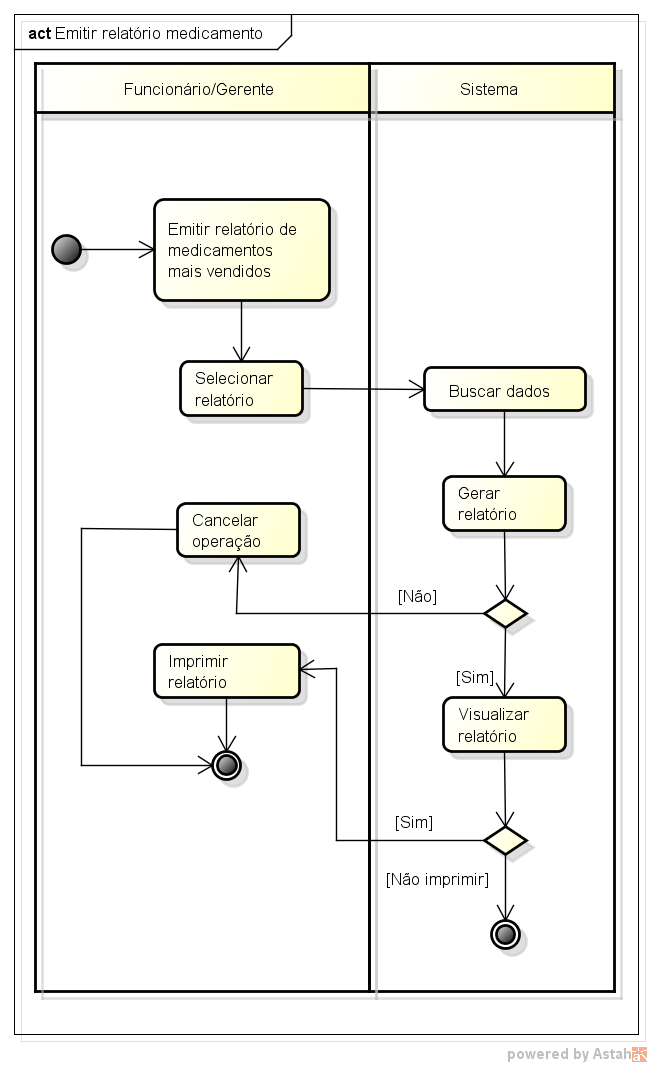
****

**DIAGRAMAS DE ATIVIDADES**

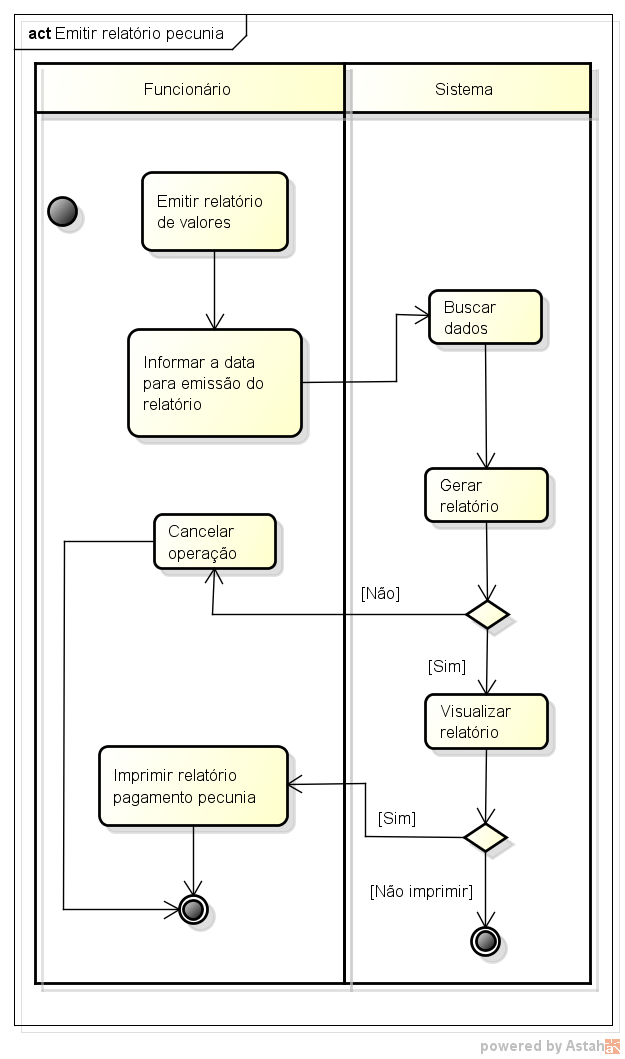
**Cadastrar Laboratório**

****

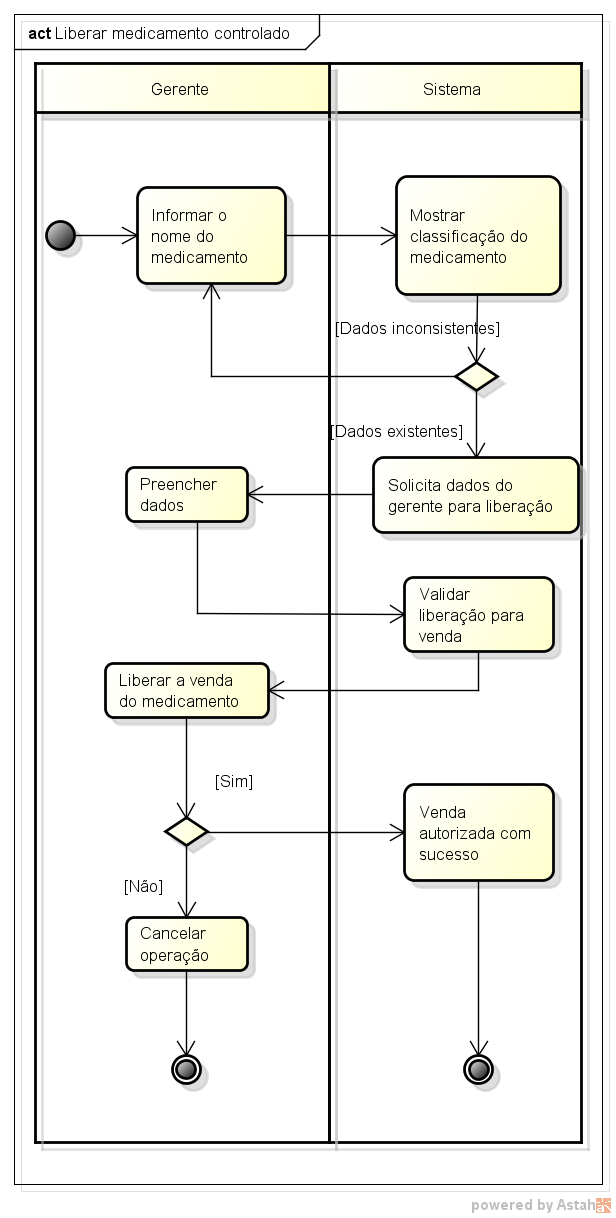
**Emitir Relatório Medicamento**

****

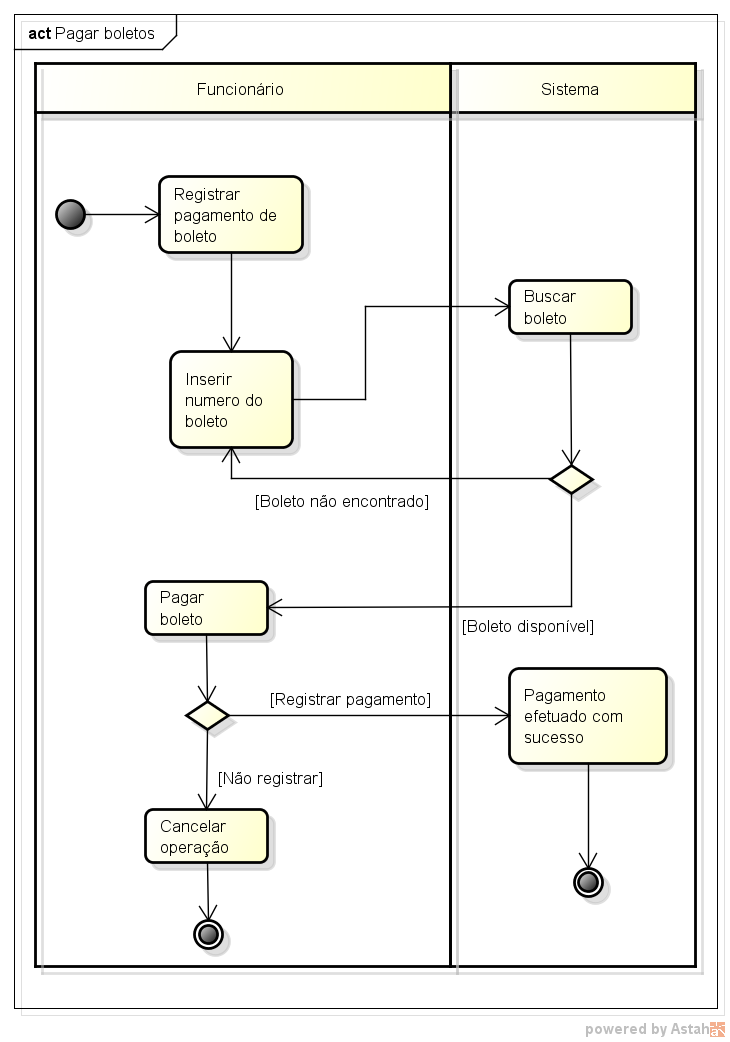
**Emitir Relatório Pecúnia**

****

**Liberar Medicamento Controlado**

****

**Pagar Boletos**

****

**LISTA DE INSPEÇÃO**

**LI01:** Todas variáveis foram inicializadas.

**LI02:** Todos loops tem condição de parada.

**LI03:** Todas variáveis de entrada são usadas.

**LI04:** Espaço alocado corretamente.

**LI05:** Todas condições de erro foram mapeadas.

**LI06:** Toda variável de saída recebe um valor antes da saída.

**LI07:** Tipos de parâmetros dos métodos – declaração e chamada.

**LI08:** Quantidade de parâmetros de métodos – declaração e chamada.

**LI09:** Todos case (switch) tem break**.**

**LI10:** Todo switch tem default.

**LI11:** Toda herânça é usada.

**LI12:** Todas as contantes foram denominadas.

**LI13:** Limite superior de vetores.

**LI14:** São usados strings de caracteres.

**LI15:** Overflow de buffer**.**

**LI16:** Cada loop termina corretamente.

**LI17:** Declarações compostas estão corretamente delimitada entre parênteses.

**LI17:** Todos os casos declarados em ‘case’ são levados em conta.

**LI19:** Entradas inesperadas podem fazer com que os dados sejam corrompidos.

**LI20:** Todas as chamadas de funções e de métodos tem o número correto de parâmetros.

**LI21:** Os parâmetros estão na ordem correta.

**LI22:** Componentes que acessam memória tem o mesmo modelo de estrutura da memória compartilhada.

**LI23:** Estruturas ligadas modificadas tem todas as suas ligações reatribuídas.

**LI24:** Armazenamento dinâmico usado tem espaço corretamente alocado.

**LI25:** Espaço de memória liberado depois de não ser mais necessário.

**LI26:** Limite de array não violado.

**LI27:** Código totalmente executável.

**LI28:** Variáveis atribuídas duas vezes utilizada entre as duas atribuições.

**LI29:** Resultados de funções utilizados.

**LI30:** Condição inacessível.

**LISTA DE PLANEJAMENTO DE TESTES**

**CAIXA PRETA**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lista de Planejamento de Testes | | | | | |
| **CAIXA PRETA** | | | | | |
| **TESTE** | **NOME** | **DESCRIÇÃO** | **ENTRADAS** | **RESULTADO ESPERADO** | **RESULTADO OBTIDO** |
| 1 | Acessar sistema | inserir senha funcionário | informar somente senha numérica | mensagem de erro informando senha inválida, devido a mesma não ser alfanumérica. |  |
| 2 | Cadastrar medicamento | cadastrar medicamento informando todos os campos obrigatórios | informar todos os campos obrigatórios | mensagem de cadastro realizado com sucesso |  |
| 3 | Cadastrar medicamento | cadastar medicamento sem informar o campo obrigatório categoria | informar todos os campos obrigatórios sem informar a categoria do medicamento | mensagem de erro informando que falta preencher campo obrigatório |  |
| 4 | Consultar medicamento | consultar medicamento sem informar nenhum tipo de filtro | N/A | mostrar toda relação de medicamentos contidos na base de dados |  |
| 5 | Venda de medicamento controlado | Solicitar assinatura digital do gerente | informar campo para liberação da venda sem autorização gerencial | mensagem informando que o funcionário não tem autorização para liberar a venda |  |
| 6 | Cadastrar laboratório | cadastrar laboratório sem informar campo obrigatório CNPJ | informar todos os campos sem informar o CNPJ do fornecedor | mensagem de ero informando que não foi possivel realizar o cadastro, devido a falta de preenchimento de campo obrigatório |  |
| 7 | Consultar estoque | consultar estoque sem informar qualquer tipo de filtro | N/A | mostrar toda relação de produtos e dados relacionados a quantidade em estoque |  |
| 8 | Acessar sistema | tentativa de acesso por usuário sem autorização | informar dados de login inválidos | mostrar mensagem de erro informando que usuário não tem autorização para acessar o sistema |  |
| 9 | Gerar relatório financeiro | emitir relatório de fluxo de caixa com base em filtros de dadas | informar data para emissão do relatório | mensagem informando que o relatório foi gerado com sucesso. |  |
| 10 | Tempo de resposta do sistema | o tempo de resposta de uma solicitação no sistema não deve ultrapassar 30 segundos | informar solicitação e aguardar resposta do sistema | solicitação exibida na tela dentro do prazo estabelecido |  |
| 11 | Tempo de processamento | executar o sistema com volume de informações anormais | solicitar várias atividades ao mesmo tempo ao sistema(consultar dados, emitir relatórios, cadastrar produtos) | evitar gerar várias interrupções por segundo |  |
| 12 | Registro ponto funcionário | Controlar horário de entrada e saída | informar código de acesso | dados validados individualmente por meio do código de acesso |  |
| 13 | Emitir relatório mensal ponto eletrônico | Relatório mensal de carga horária dos funcionários | informar nome completo funcionário e período | relatório gerado com sucesso, contendo as informações mensais de horas trabalhadas do funcionário |  |
| 14 | Leitura de código de barras | Registrar entrada no estoque de mercadorias através do código de barras | leitura do medicamento através do leitor optico | Fornecer dados do produto, tais como: informações do fabricante e código do produto. |  |
| 15 | Emitir relatório com os prazos de válidade e quantidades | Gerar relatório contendo os prazos de válidades e a quantidade em estoque dos medicamentos diariamente | N/A | retornar todos os medicamentos na base de dados |  |
| 16 | Cadastrar clientes | cadastrar clientes que utilizam medicamentos controlados | informar todos os campos obrigatórios, execeto o CRM do médico | mensagem informando que o cadastro não foi realizado, devido a omissão de dado obrigatório |  |
| 17 | Cadastrar clientes | cadastrar clientes informando todos os campos de dados pessoais | informar todos os campos obrigatórios: CPF, data de nascimento, endereço fisico e eletronico, contato telefonico e convênio | mensagem informando que o cadastro foi gerado com sucesso |  |
| 18 | Consultar clientes | informar dados dos clientes com um dado inconsistente(CPF) | informar nome completo e CPF | mensagem de erro informando que o CPF do cliente não existe na base de dados |  |
| 19 | Emitir relatório de vendas | emitir relatório de valores recebidos de acordo com os filtros de forma de pagamento e datas | informar data e o tipo de pagamento | mostrar relatório com todos os pagamentos recebidos no dia e o tipo de pagamento(dinheiro, cheque, cartão) |  |
| 20 | Medicamentos não vendidos | emitir relatório de medicamentos não vendidos por mais de 15 dias, sem informar o principio ativo | informar nome e laboratório, exceto o principio ativo | mensagem de erro informando a necessidade de preenchimento de campo obrigatório |  |
| 21 | Queda de energia | verificar se tempo recuperação dos dados ultrapassa o estabelecido(3 minutos) | N/A | tempo de resposta da recuperação do sistema deverá ser de no máximo 3 minutos |  |
| 22 | Período de uso do sistema | Verificar se o tempo de expiração da página de algum tipo de solicitação ao sistema ultrapassa 3 minutos | N/A | página da solicitação deverá fechar assim que atingir o limite de 3 minutos sem uso |  |
| 23 | Queda de rede | solicitação não carregada, devido a queda de rede, dados digitados anteriormente serão perdidos | N/A | mensagem de erro informando que página não pode ser carregada |  |
| 24 | Acesso externo ao sistema | Acessar o sistema de forma externa, sem a devida autorização | N/A | mensagem de erro informando acesso negado |  |
| 25 | Tempo de acesso ao sistema | tempo para logar não deve ultrapassar 60 segundos | informar dados de login válidos | sistema iniciado dentro do prazo estabelecido |  |
| 26 | Re-executar teste | novo módulo inserido no cadastrar medicamento | informar todos os dados obrigatórios, incluindo o novo campo | mensagem deverá mostrar que o cadastro foi realizado com sucesso |  |
| 27 | Cadastro de pedido | informar numero de pedido não cadastrado | informar o numero do pedido inválido | mensagem de erro informando que pedido não existe |  |
| 28 | Cadastro de pedido | informar número de pedido cadastrado | informar numero do pedido válido | relação do pedido é exibida na tela |  |
| 29 | Layout ambiente | análise do layout | N/A | layout de fácil utilização |  |
| 30 | Preenchimento dos campos obrigatórios | Todos os campos obrigatórios são checados antes da finalização da solicitação | N/A | mensagem informando sucesso na exibição de dada solicitação |  |

**IMPLEMENTAÇÃO CLASSES**

**Fornecedor**

|  |
| --- |
| package entity; |
|  | public class Fornecedor { |
|  |  |
|  | private String nome; |
|  | private String numCnpj; |
|  | private String endereco; |
|  | private String email; |
|  | private String telefone; |
|  | private String representantes; |
|  | private String telRepresentantes; |
|  |  |
|  | public Fornecedor() { |
|  |  |
|  | } |
|  |  |
|  | public Fornecedor(String nome, String numCnpj, String endereco, |
|  | String email, String telefone) { |
|  | this.nome = nome; |
|  | this.numCnpj = numCnpj; |
|  | this.endereco = endereco; |
|  | this.email = email; |
|  | this.telefone = telefone; |
|  | } |
|  |  |
|  | // 5 |
|  | private static final int[] pesoCNPJ = {6, 5, 4, 3, 2, 9, 8, 7, 6, 5, 4, 3, 2}; |
|  | public boolean validarCnpj(String cnpj) { |
|  | if ((cnpj==null)||(cnpj.length()!=14)) return false; |
|  |  |
|  | Integer digito1 = calcularDigito(cnpj.substring(0,12), pesoCNPJ); |
|  | Integer digito2 = calcularDigito(cnpj.substring(0,12) |
|  | + digito1, pesoCNPJ); |
|  | return cnpj.equals(cnpj.substring(0,12) + digito1.toString() |
|  | + digito2.toString()); |
|  | } |
|  |  |
|  | // 6 |
|  | public boolean validarCamposObrigatorios() { |
|  | return (nome.length() > 0 && |
|  | numCnpj.length() > 0 && |
|  | endereco.length() > 0 && |
|  | email.length() > 0 && |
|  | telefone.length() > 0 && |
|  | representantes.length() > 0 && |
|  | telRepresentantes.length() > 0); |
|  | } |
|  |  |
|  | // 7 |
|  | public boolean validarTelefone(String num) { |
|  | return num.matches(".((10)|([1-9][1-9]).)\\s9?[6-9][0-9]{3}-[0-9]{4}") |
|  | || num.matches(".((10)|([1-9][1-9]).)\\s[2-5][0-9]{3}-[0-9]{4}"); |
|  | } |
|  |  |
|  | //-------------------- |
|  | // Auxiliar functions |
|  | //-------------------- |
|  |  |
|  | private static int calcularDigito(String str, int[] peso) { |
|  | int soma = 0; |
|  | for (int indice=str.length()-1, digito; indice >= 0; indice-- ) { |
|  | digito = Integer.parseInt(str.substring(indice,indice+1)); |
|  | soma += digito\*peso[peso.length-str.length()+indice]; |
|  | } |
|  | soma = 11 - soma % 11; |
|  | return soma > 9 ? 0 : soma; |
|  | } |
|  |  |
|  | //--------------------- |
|  | // Getters ans Setters |
|  | //--------------------- |
|  |  |
|  | public String getNome() { |
|  | return nome; |
|  | } |
|  |  |
|  | public void setNome(String nome) { |
|  | this.nome = nome; |
|  | } |
|  |  |
|  | public String getNumCnpj() { |
|  | return numCnpj; |
|  | } |
|  |  |
|  | public void setNumCnpj(String numCnpj) { |
|  | this.numCnpj = numCnpj; |
|  | } |
|  |  |
|  | public String getEndereco() { |
|  | return endereco; |
|  | } |
|  |  |
|  | public void setEndereco(String endereco) { |
|  | this.endereco = endereco; |
|  | } |
|  |  |
|  | public String getEmail() { |
|  | return email; |
|  | } |
|  |  |
|  | public void setEmail(String email) { |
|  | this.email = email; |
|  | } |
|  |  |
|  | public String getTelefone() { |
|  | return telefone; |
|  | } |
|  |  |
|  | public void setTelefone(String telefone) { |
|  | this.telefone = telefone; |
|  | } |
|  |  |
|  | public String getRepresentantes() { |
|  | return representantes; |
|  | } |
|  |  |
|  | public void setRepresentantes(String representantes) { |
|  | this.representantes = representantes; |
|  | } |
|  |  |
|  | public String getTelRepresentantes() { |
|  | return telRepresentantes; |
|  | } |
|  |  |
|  | public void setTelRepresentantes(String telRepresentantes) { |
|  | this.telRepresentantes = telRepresentantes; |
|  | } |
|  |  |
|  |  |
|  | } |

**Pedido**

|  |
| --- |
| package entity; |
|  |  |
|  | import aux.Medicamento; |
|  | import aux.ClienteControlado; |
|  | import aux.Cliente; |
|  | import java.util.ArrayList; |
|  | import java.util.Date; |
|  |  |
|  | public class Pedido { |
|  |  |
|  | private int pedidoID; |
|  | private ArrayList<Medicamento> medicamentos; |
|  | private Cliente cliente; |
|  | private String enderecoEntrega; |
|  | private String formaPagamento; |
|  | private Date dataPagamento; |
|  | private float total = 0; |
|  | private String status = "ABERTO"; |
|  |  |
|  | // 1 |
|  | public void fecharPedido() { |
|  | if(medicamentos != null && medicamentos.size() > 0) { |
|  | for(Medicamento m : medicamentos) |
|  | { |
|  | this.total += m.getValor(); |
|  | } |
|  | this.status = "FECHADO"; |
|  | } |
|  | } |
|  |  |
|  | // 2 |
|  | public void pagarPedido() { |
|  | if(this.status.equals("FECHADO")) |
|  | { |
|  | dataPagamento = new Date(); |
|  | this.status = "PAGO"; |
|  | } |
|  | } |
|  |  |
|  | // 3 |
|  | public boolean conferirCpfCliente(String cpf) { |
|  | return cliente.getCpf().equals(cpf); |
|  | } |
|  |  |
|  | // 4 |
|  | public boolean conferirCrmMedico(String crm) { |
|  | if(cliente instanceof ClienteControlado) |
|  | { |
|  | return ((ClienteControlado)cliente).getCrmMedico().equals(crm); |
|  | } |
|  | else |
|  | { |
|  | return false; |
|  | } |
|  | } |
|  |  |
|  | //--------------------- |
|  | // Getters and Setters |
|  | //--------------------- |
|  |  |
|  | public int getPedidoID() { |
|  | return pedidoID; |
|  | } |
|  |  |
|  | public void setPedidoID(int pedidoID) { |
|  | this.pedidoID = pedidoID; |
|  | } |
|  |  |
|  | public ArrayList<Medicamento> getMedicamentos() { |
|  | return medicamentos; |
|  | } |
|  |  |
|  | public void setMedicamentos(ArrayList<Medicamento> medicamentos) { |
|  | this.medicamentos = medicamentos; |
|  | } |
|  |  |
|  | public Cliente getCliente() { |
|  | return cliente; |
|  | } |
|  |  |
|  | public void setCliente(Cliente cliente) { |
|  | this.cliente = cliente; |
|  | } |
|  |  |
|  | public String getEnderecoEntrega() { |
|  | return enderecoEntrega; |
|  | } |
|  |  |
|  | public void setEnderecoEntrega(String enderecoEntrega) { |
|  | this.enderecoEntrega = enderecoEntrega; |
|  | } |
|  |  |
|  | public String getFormaPagamento() { |
|  | return formaPagamento; |
|  | } |
|  |  |
|  | public void setFormaPagamento(String formaPagamento) { |
|  | this.formaPagamento = formaPagamento; |
|  | } |
|  |  |
|  | public Date getDataPagamento() { |
|  | return dataPagamento; |
|  | } |
|  |  |
|  | public void setDataPagamento(Date dataPagamento) { |
|  | this.dataPagamento = dataPagamento; |
|  | } |
|  |  |
|  | public float getTotal() { |
|  | return total; |
|  | } |
|  |  |
|  | public void setTotal(float total) { |
|  | this.total = total; |
|  | } |
|  |  |
|  | public String getStatus() { |
|  | return status; |
|  | } |
|  |  |
|  | public void setStatus(String status) { |
|  | this.status = status; |
|  | } |
|  |  |
|  |  |
|  | } |

**Ponto Eletrônico**

|  |
| --- |
| package entity; |
|  |  |
|  | import aux.Funcionario; |
|  | import java.util.Date; |
|  |  |
|  | public class Ponto { |
|  | private Funcionario funcionario; |
|  | private Date horaEntrada; |
|  | private Date horaSaida; |
|  |  |
|  | public Ponto(Funcionario funcionario) { |
|  | this.funcionario = funcionario; |
|  | } |
|  |  |
|  | // 8 |
|  | public boolean registrarEntrada(Date horario) { |
|  | if(horario == null || this.funcionario == null) return false; |
|  | if(horario.equals(this.horaSaida)) return false; |
|  | this.setHoraEntrada(horario); |
|  | return true; |
|  | } |
|  |  |
|  | // 9 |
|  | public boolean registrarSaida(Date horario) { |
|  | if(horario == null || this.funcionario == null) return false; |
|  | if(horario.equals(this.horaEntrada)) return false; |
|  | this.setHoraSaida(horario); |
|  | return true; |
|  | } |
|  |  |
|  | //--------------------- |
|  | // Getters and Setters |
|  | //--------------------- |
|  |  |
|  | public Funcionario getFuncionario() { |
|  | return funcionario; |
|  | } |
|  |  |
|  | public void setFuncionario(Funcionario funcionario) { |
|  | this.funcionario = funcionario; |
|  | } |
|  |  |
|  | public Date getHoraEntrada() { |
|  | return horaEntrada; |
|  | } |
|  |  |
|  | public void setHoraEntrada(Date horaEntrada) { |
|  | this.horaEntrada = horaEntrada; |
|  | } |
|  |  |
|  | public Date getHoraSaida() { |
|  | return horaSaida; |
|  | } |
|  |  |
|  | public void setHoraSaida(Date horaSaida) { |
|  | this.horaSaida = horaSaida; |
|  | } |
|  |  |
|  |  |
|  | } |

**TESTES UNITÁRIOS**

**Fornecedor**

|  |
| --- |
| import entity.Fornecedor; |
|  | import org.junit.Test; |
|  | import static org.junit.Assert.\*; |
|  | public class FornecedorJUnitTest { |
|  |  |
|  | public FornecedorJUnitTest() { |
|  | } |
|  |  |
|  | // validarCnpj |
|  | @Test |
|  | public void testValidarCnpjOk() { |
|  | Fornecedor f = new Fornecedor("fornecedor", |
|  | "", "", "", ""); |
|  | assertTrue(f.validarCnpj("00776574000741")); |
|  | } |
|  |  |
|  | @Test (expected = NullPointerException.class) |
|  | public void testValidarCnpjNull() { |
|  | Fornecedor f = null; |
|  | f.validarCnpj("00776574000741"); |
|  | } |
|  |  |
|  | @Test |
|  | public void testValidarCnpjNok() { |
|  | Fornecedor f = new Fornecedor("fornecedor", |
|  | "", "", "", ""); |
|  | assertFalse(f.validarCnpj("13642634756318")); |
|  | } |
|  |  |
|  | // validarCamposObrigatorios |
|  | @Test |
|  | public void testValidarCamposOk() { |
|  | Fornecedor f = new Fornecedor("fornecedor", |
|  | "test", "test", "test", "test"); |
|  | f.setRepresentantes("test"); |
|  | f.setTelRepresentantes("test"); |
|  | assertTrue(f.validarCamposObrigatorios()); |
|  | } |
|  |  |
|  | @Test (expected = NullPointerException.class) |
|  | public void testValidarCamposNull() { |
|  | Fornecedor f = null; |
|  | f.validarCamposObrigatorios(); |
|  | } |
|  |  |
|  | @Test |
|  | public void testValidarCamposNok() { |
|  | Fornecedor f = new Fornecedor("fornecedor", |
|  | "", "", "", ""); |
|  | assertFalse(f.validarCamposObrigatorios()); |
|  | } |
|  |  |
|  | // validarTelefone |
|  | @Test |
|  | public void testValidarTelOk() { |
|  | Fornecedor f = new Fornecedor(); |
|  | assertTrue(f.validarTelefone("(11) 98877-6655")); |
|  | } |
|  |  |
|  | @Test (expected = NullPointerException.class) |
|  | public void testValidarTelNull() { |
|  | Fornecedor f = null; |
|  | f.validarTelefone("(11) 98877-6655"); |
|  | } |
|  |  |
|  | @Test |
|  | public void testValidarTelNok() { |
|  | Fornecedor f = new Fornecedor(); |
|  | assertFalse(f.validarTelefone("(51)a98877-6655")); |
|  | } |
|  |  |
|  | } |

**Pedido**

|  |
| --- |
| import aux.Cliente; |
|  | import aux.ClienteControlado; |
|  | import aux.Medicamento; |
|  | import entity.Pedido; |
|  | import java.util.ArrayList; |
|  | import java.util.Date; |
|  | import org.junit.Test; |
|  | import static org.junit.Assert.\*; |
|  |  |
|  | public class PedidoJUnitTest { |
|  |  |
|  | public PedidoJUnitTest() { |
|  | } |
|  |  |
|  | // fecharPedido |
|  | @Test |
|  | public void testFecharPedidoOk() { |
|  | ArrayList<Medicamento> meds = new ArrayList<>(); |
|  | for(int i = 0; i < 10; i++) { |
|  | Medicamento med = new Medicamento(); |
|  | med.setValor(10); |
|  | meds.add(med); |
|  | } |
|  | Pedido ped = new Pedido(); |
|  | ped.setMedicamentos(meds); |
|  | ped.fecharPedido(); |
|  | assertTrue(ped.getTotal() == 100 && ped.getStatus().equals("FECHADO")); |
|  | } |
|  |  |
|  | @Test(expected = NullPointerException.class) |
|  | public void testFecharPedidoNull() { |
|  | ArrayList<Medicamento> meds = new ArrayList<>(); |
|  | for(int i = 0; i < 10; i++) { |
|  | Medicamento med = new Medicamento(); |
|  | med.setValor(10); |
|  | meds.add(med); |
|  | } |
|  | Pedido ped = null; |
|  | ped.setMedicamentos(meds); |
|  | ped.fecharPedido(); |
|  | } |
|  |  |
|  | @Test |
|  | public void testFecharPedidoNok() { |
|  | ArrayList<Medicamento> meds = new ArrayList<>(); |
|  | for(int i = 0; i < 9; i++) { |
|  | Medicamento med = new Medicamento(); |
|  | med.setValor(10); |
|  | meds.add(med); |
|  | } |
|  | Pedido ped = new Pedido(); |
|  | ped.setMedicamentos(meds); |
|  | ped.fecharPedido(); |
|  | assertFalse(ped.getTotal() == 100 && ped.getStatus().equals("FECHADO")); |
|  | } |
|  |  |
|  | // pagarPedido |
|  | @Test |
|  | public void testPagarPedidoOk() { |
|  | Pedido ped = new Pedido(); |
|  | Medicamento med = new Medicamento(); |
|  | med.setValor(10); |
|  | ArrayList<Medicamento> meds = new ArrayList<>(); |
|  | meds.add(med); |
|  | ped.setMedicamentos(meds); |
|  | ped.fecharPedido(); |
|  | ped.pagarPedido(); |
|  | assertTrue(ped.getStatus().equals("PAGO")); |
|  | } |
|  |  |
|  | @Test(expected = NullPointerException.class) |
|  | public void testPagarPedidoNull() { |
|  | Pedido ped = null; |
|  | ped.fecharPedido(); |
|  | ped.pagarPedido(); |
|  | } |
|  |  |
|  | @Test |
|  | public void testPagarPedidoNok() { |
|  | Pedido ped = new Pedido(); |
|  | ped.pagarPedido(); |
|  | assertTrue(ped.getStatus().equals("ABERTO")); |
|  | } |
|  |  |
|  | //conferirCpfCliente |
|  | @Test |
|  | public void testConferirCpfOk() { |
|  | Pedido ped = new Pedido(); |
|  | Cliente cli = new Cliente("Cliente", "End", "email", |
|  | "11122233344", "", "", new Date()); |
|  | ped.setCliente(cli); |
|  | assertTrue(ped.conferirCpfCliente("11122233344")); |
|  | } |
|  |  |
|  | @Test(expected = NullPointerException.class) |
|  | public void testConferirCpfNull() { |
|  | Pedido ped = new Pedido(); |
|  | Cliente cli = null; |
|  | ped.setCliente(cli); |
|  | ped.conferirCpfCliente("11122233344"); |
|  | } |
|  |  |
|  | @Test |
|  | public void testConferirCpfNok() { |
|  | Pedido ped = new Pedido(); |
|  | Cliente cli = new Cliente("Cliente", "End", "email", |
|  | "11122233344", "", "", new Date()); |
|  | ped.setCliente(cli); |
|  | assertFalse(ped.conferirCpfCliente("55566677788")); |
|  | } |
|  |  |
|  | //conferirCrmMedico |
|  | @Test |
|  | public void testConferirCrmOk() { |
|  | Pedido ped = new Pedido(); |
|  | ClienteControlado cli = new ClienteControlado("Cliente", "End", "email", |
|  | "", "", "", new Date(), "", "112233-4"); |
|  | ped.setCliente(cli); |
|  | assertTrue(ped.conferirCrmMedico("112233-4")); |
|  | } |
|  |  |
|  | @Test |
|  | public void testConferirCrmNull() { |
|  | Pedido ped = new Pedido(); |
|  | ClienteControlado cli = null; |
|  | ped.setCliente(cli); |
|  | assertFalse(ped.conferirCrmMedico("112233-4")); |
|  | } |
|  |  |
|  | @Test |
|  | public void testConferirCrmNok() { |
|  | Pedido ped = new Pedido(); |
|  | ClienteControlado cli = new ClienteControlado("Cliente", "End", "email", |
|  | "", "", "", new Date(), "", "112233-4"); |
|  | ped.setCliente(cli); |
|  | assertFalse(ped.conferirCrmMedico("556677-8")); |
|  | } |
|  | } |

**Ponto Eletrônico**

|  |
| --- |
| import aux.Funcionario; |
|  | import entity.Ponto; |
|  | import java.util.Date; |
|  | import org.junit.Test; |
|  | import static org.junit.Assert.\*; |
|  |  |
|  | public class PontoJUnitTest { |
|  |  |
|  | public PontoJUnitTest() { |
|  | } |
|  |  |
|  | // registrarEntrada |
|  | @Test |
|  | public void testRegEntradaOk() { |
|  | Funcionario f = new Funcionario(); |
|  | Ponto p = new Ponto(f); |
|  | assertTrue(p.registrarEntrada(new Date())); |
|  | } |
|  |  |
|  | @Test (expected = NullPointerException.class) |
|  | public void testRegEntradaNull() { |
|  | Funcionario f = new Funcionario(); |
|  | Ponto p = null; |
|  | p.registrarEntrada(new Date()); |
|  | } |
|  |  |
|  | @Test |
|  | public void testRegEntradaNok() { |
|  | Funcionario f = null; |
|  | Ponto p = new Ponto(f); |
|  | assertFalse(p.registrarEntrada(new Date())); |
|  | } |
|  |  |
|  | // registrarSaida |
|  | @Test |
|  | public void testRegSaidaOk() { |
|  | Funcionario f = new Funcionario(); |
|  | Ponto p = new Ponto(f); |
|  | assertTrue(p.registrarSaida(new Date())); |
|  | } |
|  |  |
|  | @Test (expected = NullPointerException.class) |
|  | public void testRegSaidaNull() { |
|  | Funcionario f = new Funcionario(); |
|  | Ponto p = null; |
|  | p.registrarSaida(new Date()); |
|  | } |
|  |  |
|  | @Test |
|  | public void testRegSaidaNok() { |
|  | Funcionario f = null; |
|  | Ponto p = new Ponto(f); |
|  | assertFalse(p.registrarSaida(new Date())); |
|  | } |
|  | } |