Avaliação da Disciplina: Desenvolvimento Web 2

Objetivo e Contexto

Desenvolver uma aplicação web completa para um sistema de controle de medicamentos, aplicando a arquitetura MVC (Model-View-Controller). O projeto deve utilizar Node.js/Express, MySQL (mysql2), EJS para renderização e a biblioteca Joi para validação robusta dos dados de entrada.

Esta prova é individual e vale 10 pontos. Duração: três aulas.

Stack Tecnológico Exigido

Categoria	**Tecnologia**	**Uso Obrigatório**
Arquitetura	MVC	Separação estrita em Model, View e Controller.
Backend	Node.js, Express	Servidor e lógica de negócio.
Banco de Dados	MySQL	Driver mysql2.
Templates	EJS	Renderização de todas as páginas e dados.
Validação	Joi	Validação de schemas e dados de entrada (POST).

Configuração do Banco de Dados (MySQL)

SQL
-- Criação do Banco de Dados
CREATE DATABASE IF NOT EXISTS saude;
USE saude;

-- Tabela de Medicamentos

```
CREATE TABLE medicamentos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome_medicamento VARCHAR(255) NOT NULL,
    quantidade INT NOT NULL DEFAULT 0
);

-- Tabela de Retiradas (Registra quem pegou qual medicamento)
CREATE TABLE retiradas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email_municipe VARCHAR(255) NOT NULL,
    medicamento_id INT NOT NULL,
    data_retirada DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (medicamento_id) REFERENCES medicamentos(id)
);

-- Inclusão de Dados Iniciais
INSERT INTO medicamentos (nome_medicamento, quantidade) VALUES
('Paracetamol 500mg', 15),
('Dipirona 1g', 5),
('Amoxicilina 500mg', 2),
('Ibuprofeno 600mg', 1);
```

Funcionalidades e Formato de Saída (View EJS)

Todas as interações devem ser renderizadas na view EJS principal (farmacia.ejs). O formato de saída é crucial para a correção automatizada.

1. Configuração da Arquitetura e Servidor
   - Arquitetura MVC (0,5 pt): Implementar separação lógica (Model, View, Controller).
   - Porta (0,5 pt): O servidor deve rodar na porta 4000

2. Listar Todos os Medicamentos
   - Método: GET
   - Rota: / (Página inicial)
   - Ação: Buscar todos os medicamentos e suas quantidades.
   - Saída EJS (Parâmetros de Renderização e Conteúdo):
     - O Controller deve chamar: res.render('farmacia', { medicamentos: [...], mensagem: null, erro: null, filtro: null })
     - Teste de Saída Esperado: A view renderizada deve conter um elemento HTML com o atributo id="total-medicamentos" cujo innerText seja o número total de medicamentos listados (Ex: 4).

3. Retirar Medicamento
   - Método: POST

• Rota: /retirada
• Corpo da Requisição (Exemplo):
medicamento_id=1&email_municipe=cidadao.teste@email.com

A. Validação de Dados de Entrada com JOI
Implementar schema Joi para medicamento_id (número, obrigatório, > 0) e email_municipe (e-mail válido, obrigatório).
B. Lógica de Retirada e Saída de Feedback
A rota deve, após a operação (ou falha), redirecionar ou renderizar a view principal com o feedback nos parâmetros mensagem ou erro.

Cenário	Status HTTP	Exemplo de Mensagem no EJS (Parâmetro)	Teste de Saída Esperado na View
Sucesso	200 OK	Retirada realizada com sucesso. Estoque atual: 14	Elemento com id="mensagem-sucesso" contendo o texto acima. Ex.: return res.status(200).redirect(`/?mensagem=${encodeURIComponent(sucessoMsg)}`);
Erro (Estoque Baixo)	400 Bad Request	O medicamento Amoxicilina 500mg não possui estoque disponível para retirada.	Elemento com id="mensagem-erro" contendo o texto acima. Ex.: return res.status(200).redirect(`/?mensagem=${encodeURIComponent(mensagem)}`);
Erro (Já Retirou)	409 Conflict	Este munícipe já retirou o medicamento Paracetamol 500mg e não pode retirar novamente.	Elemento com id="mensagem-erro" contendo o texto acima.

Erro (JOI/Input)	400 Bad Request	Dados inválidos: "email_municipe" deve ser um e-mail válido	Elemento com id="mensagem-erro" contendo a mensagem de erro do JOI.

4. Medicamentos com Estoque Baixo
   • Método: GET
   • Rota: /estoque-baixo
   • Ação: Listar medicamentos com menos de 3 unidades em estoque.
   • Saída EJS (Parâmetros de Renderização e Conteúdo):
      ◦ O Controller deve chamar: res.render('farmacia', { medicamentos: [{id: 3, ...}, {id: 4, ...}], mensagem: null, erro: null, filtro: 'ESTOQUE BAIXO' })
      ◦ Teste de Saída Esperado: A view deve conter um elemento HTML com o atributo id="filtro-ativo" cujo innerText seja o valor do parâmetro filtro (Ex: ESTOQUE BAIXO).

Roteiro de Desenvolvimento Sugerido (3 Aulas)

Este roteiro visa guiar o aluno dentro do tempo estipulado, garantindo que as funcionalidades principais sejam abordadas.

1. Abra o seu github e faça um fork do repositório https://github.com/wab030/votingtemplate
2. Será no novo repositório que você irá desenvolver a sua prova. Clone esse repositório para a sua máquina.
3. Entre no folder que você clonou.
4. No repositório já existe uma pasta src onde deve ser desenvolvida a avaliação. Já existem dois arquivos lá. o server.js e o app.js.
5. O seu arquivo principal para inicio do server é o server.js. Para iniciar o servidor digite nodemon server.js.
6. A sua solução deve estar toda dentro da pasta src para o processo funcionar. Isso é obrigatório.
7. Ao término da prova add seus arquivos à área de stage, commite e envie para o seu repo (fork)
8. Agora você precisa enviar o seu código para o repositório do professor.
    a. No seu repo encontre o botão contribute, clique sobre ele e abra um pull request.
    b. No título coloque o seu nome seguido do seu prontuário. Ex.: André Luís Bordignon CP209089
    c. (Opcional) Coloque uma mensagem na descrição.
    d. Clique no botão Criar Pull Request.
9. Depois do pull request o processo automatizado será iniciado. Quando terminar você poderá acessar o relatório na aba actions do repo do professor. Verifique se está tudo OK. Se não faça os ajustes necessários e submeta novamente o código. Na segunda vez você não precisa criar um pull request, basta subir os arquivos.

Tabela de Pontuação Consolidada (10,0 Pontos)

Requisito	Detalhes da Avaliação	Pontos
1. Configuração e Arquitetura	**(Total: 1,0 ponto)**	**1,0**
	a. Implementação da arquitetura **MVC** (Model, View, Controller, Rotas). (0,5 pt)	
	b. Servidor rodando na **Porta 4000**. (0,5 pt)	
2. Listar Todos (GET /)	**(Total: 1,0 ponto)**	**1,0**
	a. Rota `GET /` funcionando, exibindo **todos** os medicamentos.	
	b. Saída EJS correta, incluindo o elemento `id="total-medicamentos"`. (1,0 pt)	
3. Retirar Medicamento (POST /retirada)	**(Total: 5,0 pontos)**	**5,0**
	a. Sucesso e Estoque (2,0 pt): Processamento completo da retirada (transação/Model), redução do estoque e **feedback de sucesso** na View.	
	b. Validação JOI (1,0 pt): Implementação do *middleware* Joi para validar `medicamento_id` e `email_municipe`, e feedback de **erro na View** para inputs inválidos.	
	c. Erro: Duplicidade (1,0 pt): Lógica para impedir que o munícipe retire o **mesmo medicamento** mais de uma vez (erro 409).	
	d. Erro: Estoque Zero (1,0 pt): Verificação de estoque antes da retirada e feedback de erro para **estoque insuficiente**.	
4. Estoque Baixo (GET /estoque-baixo)	**(Total: 1,0 ponto)**	**1,0**
	a. Funcionalidade e Lógica (0,5 pt): Rota `GET /estoque-baixo` implementada, filtrando corretamente itens com **menos de 3** unidades.	

	b. Saída EJS (0,5 pt): Renderização correta da View com o elemento `id="filtro-ativo"` com o valor **`ESTOQUE BAIXO`**.	
5. Análise de Código com IA (Gemini)	**(Total: 2,0 pontos)**	**2,0**
	a. Pontuação baseada na qualidade do código, boas práticas (ex: transações), segurança (ex: SQL Injection) e clareza, conforme relatório da IA.	
TOTAL GERAL DA AVALIAÇÃO		**10,0**