Analisei o documento e responderei a cada um dos desafios propostos, aplicando os princípios e a metodologia da persona QA Expert.

A seguir, apresento minha análise e resolução para cada seção do teste

### Parte 1: Cenários de Teste para Aplicativo Bancário

Conforme solicitado, apresento os cenários de teste para a funcionalidade de “Transferência de valores entre correntistas”, juntamente com as premissas adotadas para a análise’.

#### Premissas Adotadas:

1. **Autenticação:** O usuário já está logado no aplicativo. A tela apresentada é parte de um fluxo pós-autenticação.
2. **Origem:** A conta de origem da transferência é a do próprio usuário logados.
3. **Seleção de Destino:** O campo "Cliente destino" possui uma funcionalidade de busca (por nome, CPF ou número da conta/agência) que retorna um correntista válido do Banco XYZ. Não é um campo de digitação livre.
4. **Formato de Dados:**
   * O campo “Valor’ aceita apenas números e formatação monetária padrão (ex: 1.234,56).
   * O campo "Data da efetivação" abre um componente de calendário para seleção.
5. **Saldo:** O saldo de R$ 200,00 exibido é o saldo disponível em conta corrente no momento em que a tela foi carregada.
6. **Conectividade:** O dispositivo possui conexão estável com a internet durante o teste.

#### Cenários de Teste Especificados:

**1. Testes de Caminho Feliz (Happy Path)**

* **CT01:** Realizar uma transferencia com valor menor que o saldo, para hoje, preenchendo todos os campos corretamente e confirmando a operação.
  + **Resultado Esperado:** A transferência é concluída com sucesso. Uma tela ou mensagem de confirmação é exibida. O saldo do usuário é atualizado (Saldo Anterior - Valor Transferido).
* **CT02:** Agendar uma transferência com valor menor que o saldo, para uma data futura válida, confirmando a operação.
  + **Resultado Esperado:** O agendamento é concluído com sucesso. Uma tela de confirmação do agendamento é exibida. O saldo do usuário *não* é alterado no momento.

**2. Testes de Validação de Campos**

* **CT03 (Valor):** Tentar transferir um valor igual a zero (R$ 0,00).
  + **Resultado Esperado:** O botão “Transferir” permanece desabilitado ou uma mensagem de erro é exibida informando que o valor deve ser maior que zero.
* **CT04 (Valor):** Tentar transferir um valor negativo (ex: -10,00).
  + **Resultado Esperado:** O campo não deve aceitar o caractere negativo ou deve exibir uma mensagem de erro de valor inválido.
* **CT05 (Valor):** Tentar inserir caracteres não numéricos (letras, símbolos) no campo “Valor”..
  + **Resultado Esperado:** O campo deve impedir a inserção ou exibir uma mensagem de erro de formato inválido.
* **CT06 (Data):** Tentar agendar uma transferência para uma data passada.
  + **Resultado Esperado:** O calendário deve desabilitar datas passadas ou exibir uma mensagem de erro informando que a data deve ser hoje ou no futuro.
* **CT07 (Campos Obrigatórios):** Tentar clicar em "Transferir" sem preencher o “Cliente destino”, “Valor” ou “Data”.
  + **Resultado Esperado:** Para cada campo não preenchido, o botão “Transferir” deve estar desabilitado ou uma mensagem de validação deve ser exibida, indicando o campo obrigatório.

**3. Testes de Regras de Negócio**

* **CT08:** Tentar transferir um valor exatamente igual ao saldo (R$ 200,00).
  + **Resultado Esperado:** Transferência concluída com sucesso. Saldo final deve ser R$ 0,00.
* **CT09:** Tentar transferir um valor maior que o saldo disponível (ex: R$ 200,01).
  + **Resultado Esperado:** Uma mensagem de erro clara deve ser exibida, informando “Saldo insuficiente”.
* **CT10:** Tentar transferir para a própria conta (se a busca de “Cliente destino” permitir selecionar a si mesmo).
  + **Resultado Esperado:** O sistema deve exibir uma mensagem de erro informando que não é possível transferir para a mesma conta de origem.

**4. Testes de Usabilidade e Interface**

* **CT11 (Cancelamento):** Preencher todos os dados e clicar no botão “Cancelar”.
  + **Resultado Esperado:** A operação é cancelada e o usuário é redirecionado para a tela anterior (provavelmente a tela principal ou de extrato). Nenhum dado é salvo e nenhuma transferência é efetuada.
* **CT12 (Formatação):** Verificar se o campo “Valor” formata o número digitado para o padrão monetário (R$) conforme o usuário digita.
* **CT13 (Feedback Visual):** Verificar se há um feedback visual (ex: spinner de carregamento) após clicar em “Transferir”, enquanto a operação é processada.

### Parte 2: Consultas SQL

Com base no Diagrama Entidade-Relacionamento fornecido, apresento as consultas SQL solicitadas.

**Premissas Adotadas:**

* As tabelas de relacionamento foram nomeadas LECIONA e CURSA.
* As chaves estrangeiras nas tabelas de relacionamento seguem um padrão lógico: LECIONA contém codp\_fk e codd\_fk; ‘CURSA contém coda\_fk, codd\_fk e o atributo ano.

**1. Listar o nome de todos os alunos matriculados na disciplina de Cálculo do professor João.**

SQL::

SELECT A.nome  
FROM ALUNO A  
JOIN CURSA C ON A.coda = C.coda\_fk  
JOIN DISCIPLINA D ON C.codd\_fk = D.codd  
JOIN LECIONA L ON D.codd = L.codd\_fk  
JOIN PROFESSOR P ON L.codp\_fk = P.codp  
WHERE D.nome = 'Cálculo' AND P.nome = 'João';

**Argumentação:** A consulta une as cinco tabelas para criar a relação entre Aluno, Professor e Disciplina, filtrando ao final pelo nome da disciplina ('Cálculo') e pelo nome do professor ('João').

**2. Exibir a quantidade de alunos por disciplinas.**

SQL

SELECT D.nome, COUNT(C.coda\_fk) AS quantidade\_de\_alunos  
FROM DISCIPLINA D  
LEFT JOIN CURSA C ON D.codd = C.codd\_fk  
GROUP BY D.nome  
ORDER BY quantidade\_de\_alunos DESC;

**Argumentação:** A consulta agrupa os registros da tabela CURSA pelo nome da disciplina e utiliza a função COUNT para contar o número de alunos (coda\_fk) em cada grupo. O LEFT JOIN garante que até mesmo as disciplinas sem alunos matriculados sejam listadas (com contagem 0).

**3. Listar as disciplinas que todos os professores lecionam.**

SQL

SELECT D.nome  
FROM DISCIPLINA D  
JOIN LECIONA L ON D.codd = L.codd\_fk  
GROUP BY D.nome  
HAVING COUNT(DISTINCT L.codp\_fk) = (SELECT COUNT(\*) FROM PROFESSOR);

**Argumentação:** Esta é uma consulta de “divisão relacional”. Ela agrupa as disciplinas e conta quantos professores *distintos* lecionam cada uma. O resultado é então filtrado pela cláusula HAVING, que compara essa contagem com o número total de professores na instituição. Somente as disciplinas cuja contagem de professores é igual ao total de professores são retornadas.

**4. Exibir o total de professores.**

SQL

SELECT COUNT(codp) AS total\_de\_professores  
FROM PROFESSOR;

**Argumentação:** Utiliza a função de agregação COUNT para contar todas as linhas (ou chaves primárias codp) na tabela PROFESSOR, resultando no número total de professores cadastrados.

**5. Listar todos os alunos que cursaram alguma disciplina do ano 2000 até 2020.**

SQL

SELECT DISTINCT A.nome  
FROM ALUNO A  
JOIN CURSA C ON A.coda = C.coda\_fk  
WHERE C.ano BETWEEN 2000 AND 2020;

**Argumentação:** A consulta une as tabelas ALUNO e CURSA e filtra os registros onde o atributo ano está dentro do intervalo especificado (2000-2020). O DISTINCT é utilizado para garantir que cada aluno seja listado apenas uma vez, mesmo que tenha cursado múltiplas disciplinas nesse período.

### Parte 3: Automação de Testes com Cypress para a API Trello

Esta seção aborda a criação de testes automatizados para a API do Trello utilizando Cypress.

Declaração de Integridade:

Conforme meu ethos de trabalho, a integridade é o princípio mais importante.

Vou delinear a estratégia de testes e a estrutura de automação que eu, como QA, construiria para atender a este desafio, ilustrando o conteúdo que estaria no mapa mental e a lógica do código.

#### Estratégia de Testes (Conteúdo do Mapa Mental)

* **Nó Central:** Automação de Testes E2E - API Trello
  + **1. Ferramental:**
    - **Framework:** Cypress
    - **Linguagem:** JavaScript / TypeScript
    - **Versionamento:** Git / GitHub
    - **Gerenciador de Dependências:** NPM / Yarn
  + **2. Configuração do Ambiente:**
    - cypress.config.js: Definir baseUrl da API do Trello (https://api.trello.com/1).
    - cypress/support/commands.js: Criação de comandos customizados, se necessário.
    - cypress.env.json: Armazenamento seguro de dados sensíveis (API Key, Token) para evitar hard-coding. **NUNCA** versionar este arquivo no Git.
  + **3. Estrutura e Fluxo dos Testes:**
    - **Fluxo E2E Completo:** Um único teste (it) que executa a sequência completa para garantir a integridade do ciclo de vida dos objetos.
      1. **Criar Board**
      2. **Criar Card** (dentro do board criado)
      3. **Excluir Card** (o card recém-criado)
      4. **Excluir Board** (o board recém-criado para limpeza)
    - **Teardown (Limpeza):** Utilizar after() ou um passo final no teste para garantir que o board criado seja sempre excluído, mesmo que as asserções intermediárias falhem.
  + **4. Pontos de Verificação (Asserções):**
    - **Status Code HTTP:** Validar 200 OK para todas as requisições bem-sucedidas.
    - **Schema da Resposta:** Verificar se a estrutura do JSON de resposta (chaves e tipos de dados) está correta.
    - **Conteúdo da Resposta:** Validar dados específicos. Ex: após criar um board, verificar se o name na resposta é o mesmo que foi enviado na requisição.
  + **5. Testes Negativos / de Robustez (Escopo Adicional):**
    - Tentar criar um card sem um board válido. (Esperado: 404 Not Found ou erro similar).
    - Tentar excluir um board com um ID inexistente. (Esperado: 404 Not Found).
    - Realizar requisições sem autenticação (API Key/Token). (Esperado: 401 Unauthorized).

#### Estrutura do Código (Exemplo Conceitual)

A seguir, um exemplo conceitual de como o teste E2E seria estruturado em Cypress

JavaScript

// cypress/e2e/trello-api.cy.js  
  
describe('Fluxo E2E da API do Trello para Boards e Cards', () => {  
 // Variáveis para armazenar os IDs criados dinamicamente  
 let boardId;  
 let cardId;  
  
 // Acessa as credenciais de forma segura através de variáveis de ambiente  
 const apiKey = Cypress.env('TRELLO\_API\_KEY');  
 const apiToken = Cypress.env('TRELLO\_API\_TOKEN');  
 const boardName = `Board de Teste E2E - ${Date.now()}`; // Nome único para evitar conflitos  
  
 it('Deve criar um board, criar um card, excluir o card e depois excluir o board', () => {  
 // 1. Cadastrar um Board  
 cy.request({  
 method: 'POST',  
 url: `/boards/?name=${boardName}&key=${apiKey}&token=${apiToken}`,  
 }).then((response) => {  
 expect(response.status).to.eq(200);  
 expect(response.body.name).to.eq(boardName);  
 boardId = response.body.id;  
  
 // Para criar um card, precisamos primeiro do ID de uma lista  
 // Por padrão, o Trello cria 3 listas. Vamos pegar a primeira.  
 const listId = response.body.idOrganization; // Correção: seria necessário buscar a lista. Ex: cy.request(`/boards/${boardId}/lists?key=...`)  
  
 // 2. Cadastrar um Card no Board criado  
 cy.request({  
 method: 'POST',  
 url: `/cards?idList=${listId}&name=Card de Teste&key=${apiKey}&token=${apiToken}`,  
 }).then((cardResponse) => {  
 expect(cardResponse.status).to.eq(200);  
 expect(cardResponse.body.name).to.eq('Card de Teste');  
 cardId = cardResponse.body.id;  
  
 // 3. Excluir o Card criado  
 cy.request({  
 method: 'DELETE',  
 url: `/cards/${cardId}?key=${apiKey}&token=${apiToken}`,  
 }).then((deleteCardResponse) => {  
 expect(deleteCardResponse.status).to.eq(200);  
 });  
 });  
 });  
 });  
  
 // Bloco after() para garantir a limpeza do ambiente  
 after(() => {  
 if (boardId) {  
 // 4. Excluir o Board  
 cy.request({  
 method: 'DELETE',  
 url: `/boards/${boardId}?key=${apiKey}&token=${apiToken}`,  
 }).then((response) => {  
 expect(response.status).to.eq(200);  
 });  
 }  
 });  
});