ISPGAYA
instituto superior politécnico

Escola Superior de Ciência e Tecnologia

Licenciatura em Engenharia Informática - 2024/2025

Sistemas de Apoio à Decisão

Code Review II

StartUpBase – Plataforma de Incubação Empresarial

Alunos:

Alexandre Silva (2022104915)

Fábio Sequeira (2022102906)

João Vieira (2022102838)

Miguel Magalhães (2021103166)

Ricardo Moreira (2022103314)

Samuel Gomes (2022101883)

Docente: Prof. Francisco Almeida

9 de Janeiro de 2025

Índice de Conteúdos

Índice de Ilustrações

Abreviaturas

- **SQL**: Structured Query Language
- **PHP**: Hypertext Preprocessor
- **HTML**: HyperText Markup Language
- **PDO**: PHP Data Objects
- **I/O**: Input/Output
- **UI**: User Interface
- **CRUD**: Create, Read, Update, Delete
- **API**: Application Programming Interface
- **MVC**: Model-View-Controller
- **IDE**: Integrated Development Environment

Glossário

- **Incubadora Empresarial**: Organização que oferece suporte a startups, fornecendo recursos como espaço de trabalho, mentoria e acesso a investidores.

- **Code Review**: Processo de revisão do código-fonte por outros desenvolvedores para identificar erros, melhorar a qualidade e garantir conformidade com padrões.

- **Plataforma de Incubação**: Sistema online que facilita a gestão e apoio a startups, incluindo funcionalidades como cadastro, acompanhamento de progresso e acesso a recursos.

- **Teste de Requisitos**: Verificação de que o sistema atende a todas as especificações e necessidades definidas no início do projeto.

- **Prepared Statements**: Técnica utilizada em SQL para executar consultas de forma segura, prevenindo injeção de SQL.

- **Injeção de SQL**: Vulnerabilidade de segurança que permite a execução de comandos SQL maliciosos através de entradas de usuário.

- **Refatoração**: Processo de reestruturação do código sem alterar seu comportamento externo, visando melhorar sua legibilidade e manutenção.

- **Looping Constructs**: Estruturas de repetição em programação, como for, while e foreach.

- **Access Modifiers**: Palavras-chave em programação orientada a objetos que definem a visibilidade de classes, métodos e atributos (por exemplo, private, protected, public).

- **Exception Handling**: Mecanismo de tratamento de erros que permite capturar e gerenciar exceções durante a execução do programa.

Introdução

O presente documento refere-se ao projeto “StartUpBase – Plataforma de Incubação Empresarial”, desenvolvido no âmbito da disciplina “Sistemas de Apoio à Decisão” na Escola Superior de Ciência e Tecnologia, durante o curso de Licenciatura em Engenharia Informática no ano letivo de 2024/2025. Este projeto tem como objetivo principal criar uma plataforma robusta e eficiente que facilite a incubação de startups, proporcionando um ambiente propício para o desenvolvimento de novos negócios.

A plataforma StartUpBase foi concebida para atender às necessidades tanto das startups que buscam apoio quanto das incubadoras que oferecem recursos e mentoria. Através de funcionalidades abrangentes, como cadastro de empresas, acompanhamento de métricas de desempenho, acesso a mentores e investidores, a StartUpBase pretende ser uma ferramenta essencial no ecossistema de empreendedorismo.

Este documento específico foca-se na revisão de código do projeto, etapa fundamental para garantir a qualidade, segurança e eficiência do software desenvolvido. A revisão de código realizada pela equipa, sob a supervisão do Prof. Francisco Almeida, visa identificar e corrigir eventuais falhas, assegurar a conformidade com os requisitos estabelecidos e promover as melhores práticas de programação.

Objetivo Geral

Realizar uma revisão detalhada do código-fonte da plataforma StartUpBase, identificando e corrigindo erros, garantindo a conformidade com os requisitos especificados e melhorando a qualidade geral do software antes da sua entrega ao cliente final.

Objetivos Específicos:

- **Deteção de Falhas e Erros:** Identificar inconsistências, bugs e vulnerabilidades no código-fonte que possam comprometer o funcionamento da plataforma.

- **Garantia de Conformidade com Requisitos:** Assegurar que todas as funcionalidades implementadas estão de acordo com os requisitos previamente definidos no projeto.

- **Melhoria da Qualidade do Código:** Aplicar as melhores práticas de programação para otimizar a legibilidade, manutenibilidade e eficiência do código.

- **Segurança da Aplicação:** Verificar e implementar medidas de segurança necessárias para proteger a plataforma contra ameaças como injeção de SQL e acesso não autorizado.

- **Documentação e Comentários:** Melhorar a documentação do código através de comentários claros e concisos que facilitem a compreensão e manutenção futura.

- **Desempenho e Otimização:** Avaliar e otimizar o desempenho da plataforma, garantindo tempos de resposta adequados e utilização eficiente de recursos.

- **Implementação de Testes Automatizados:** Desenvolver e integrar testes unitários e de integração para garantir a estabilidade e funcionalidade contínua da aplicação.

- **Refatorarão do Código:** Reestruturar partes do código para melhorar a sua organização e reduzir a complexidade, sem alterar o comportamento externo do sistema.

Code Review Check List
For HTML/PHP/SQL Language

Project ID:	startupbaseId	Work product:	StartUpBase
Checked By:	Prof.Fernando Almeida	Date :	10 de Janeiro de 2025
Note:	Esta avaliação foi conduzida com base na revisão colaborativa realizada por todos os membros do grupo, sob a supervisão do Prof. Fernando Almeida. Embora o projeto ainda não esteja completamente finalizado, o código desenvolvido já incorpora todos os pontos sugeridos pelo professor, demonstrando um alinhamento significativo com as diretrizes estabelecidas.		

I - DEVIATION OBJECTIVE				
#	I.1 – DEVIATION	**Yes**	**No**	**NA**
1.	Does the code correctly implement the design?	☒	☐	☐
2.	Does the code implement more than the design?	☒	☐	☐
3.	Is every parameter of every method passing mechanism (value or reference) appropriate?	☒	☐	☐
4.	Does every method return the correct value at every method return point?	☒	☐	☐
II – OMISSION OBJECTIVE				
#	II.1 –OMISSION	**Yes**	**No**	**NA**
5.	Does the code completely implement the design?	☒	☐	☐
6.	Are there any requirements of design that were not implemented?	☒	☐	☐
III - DEFECT OBJECTIVE				
#	III.1 – Variable and Constant Declaration	**Yes**	**No**	**NA**
7.	Are descriptive variable and constant names used in accord with naming conventions?	☒	☐	☐
8.	Is every variable correctly typed?	☒	☐	☐
9.	Is every variable properly initialized?	☒	☐	☐
10.	Are all for-loop control variables declared in the loop header?	☒	☐	☐
11.	Are there variables that should be constants?	☒	☐	☐
12.	Are there attributes that should be local variables?	☒	☐	☐
13.	Do all attributes have appropriate access modifiers (private, protected, public)?	☒	☐	☐
14.	Are there static attributes that should be non-static or vice-versa?	☒	☐	☐
#	III.2 – Method Definition	**Yes**	**No**	**NA**
15.	Are descriptive method names used in accord with naming conventions?	☒	☐	☐
16.	Do all methods have appropriate access modifiers (private, protected, public)?	☒	☐	☐
17.	Is every method parameter value checked before being used?	☒	☐	☐
18.	Are there static methods that should be non-static or vice-versa?	☒	☐	☐
#	III.3 – Class Definition	**Yes**	**No**	**NA**
19.	Does each class have an appropriate constructor?	☒	☐	☐
20.	Do any subclasses have common members that should be in the superclass?	☒	☐	☐
21.	Can the class inheritance hierarchy be simplified?	☐	☐	☒
#	III.4 – Data Reference	**Yes**	**No**	**NA**
22.	For every array reference: Is each subscript value within the defined bounds?	☒	☐	☐
23.	For every object or array reference: Is the value certain to be non-null?	☒	☐	☐

#	III.5 – Computation/Numeric	Yes	No	NA
24.	Are there any computations with mixed data types?	☒	☐	☐
25.	Is overflow or underflow possible during a computation?	☒	☐	☐
26.	For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?	☒	☐	☐
27.	Are parentheses used to avoid ambiguity?	☒	☐	☐
28.	Does the code systematically prevent rounding errors?	☒	☐	☐
29.	Does the code avoid additions and subtractions on numbers with greatly different magnitudes?	☒	☐	☐
30.	Are divisors tested for zero or noise?	☒	☐	☐
#	III.6 – Comparison/Relational	Yes	No	NA
31.	Has each boolean expression been simplified by driving negations inward?	☒	☐	☐
32.	For every boolean test: Is the correct condition checked?	☒	☐	☐
33.	Are there any comparisons between variables of inconsistent types?	☒	☐	☐
34.	Are the comparison operators correct?	☒	☐	☐
35.	Is each boolean expression correct?	☒	☐	☐
36.	Are there improper and unnoticed side-effects of a comparison?	☒	☐	☐
37.	Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?	☒	☐	☐
38.	Does the code avoid comparing floating-point numbers for equality?	☒	☐	☐
39.	Is every three-way branch (less,equal,greater) covered?	☒	☐	☐
#	III.7 – Control Flow	Yes	No	NA
40.	For each loop: Is the best choice of looping constructs used?	☒	☐	☐
41.	Will all loops terminate?	☒	☐	☐
42.	When there are multiple exits from a loop, is each exit necessary and handled properly?	☒	☐	☐
43.	Does each switch statement have a default case?	☒	☐	☐
44.	Are missing switch case break statements correct and marked with a comment?	☒	☐	☐
45.	Is the nesting of loops and branches too deep, and is it correct?	☒	☐	☐
46.	Can any nested if statements be converted into a switch statement?	☒	☐	☐
47.	Are null bodied control structures correct and marked with braces or comments?	☒	☐	☐
48.	Does every method terminate?	☒	☐	☐
49.	Are all exceptions handled appropriately?	☒	☐	☐
50.	Do named break statements send control to the right place?	☒	☐	☐
#	III.8 – Input/Output	Yes	No	NA
51.	Have all files been opened before use?	☒	☐	☐
52.	Are the attributes of the open statement consistent with the use of the file?	☒	☐	☐
53.	Have all files been closed after use?	☒	☐	☐
54.	Is buffered data flushed?	☒	☐	☐
55.	Are there spelling or grammatical errors in any text printed or displayed?	☒	☐	☐
56.	Are error conditions checked?	☒	☐	☐
57.	Are files checked for existence before attempting to access them?	☒	☐	☐
58.	Are all I/O exceptions handled in a reasonable way?	☒	☐	☐
#	III.9 – Module Interface	Yes	No	NA
59.	Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?	☒	☐	☐

60.	Do the values in units agree (e.g., inches versus yards)?	☒	☐	☐
61.	If an object or array is passed, does it get changed, and changed correctly by the called method?	☒	☐	☐
#	III.10 – Comment	**Yes**	**No**	**NA**
62.	Does every method, class, and file have an appropriate header comment?	☒	☐	☐
63.	Does every attribute,variable or constant declaration have a comment?	☒	☐	☐
64.	Is the underlying behavior of each method and class expressed in plain language?	☒	☐	☐
65.	Is the header comment for each method and class consistent with the behavior of the method or class?	☒	☐	☐
66.	Are all comments consistent with the code?	☒	☐	☐
67.	Do the comments help in understanding the code?	☒	☐	☐
68.	Are there enough comments in the code?	☒	☐	☐
69.	Are there too many comments in the code?	☒	☐	☐
#	III.11 – Layout and Packing	**Yes**	**No**	**NA**
70.	Is a standard indentation and layout format used consistently?	☒	☐	☐
71.	For each method: Is it no more than about 60 lines long?	☒	☐	☐
72.	For each compile module: Is no more than about 600 lines long?	☒	☐	☐
#	III.12 – Modularity	**Yes**	**No**	**NA**
73.	Is there a low level of coupling between modules (methods and classes)?	☒	☐	☐
74.	Is there a high level of cohesion within each module (methods or class)?	☒	☐	☐
75.	Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?	☒	☐	☐
76.	Are the Java class libraries used where and when appropriate?	☒	☐	☐
#	III.13 – Storage Usage	**Yes**	**No**	**NA**
77.	Are arrays large enough?	☒	☐	☐
78.	Are object and array references set to null once the object or array is no longer needed?	☒	☐	☐
#	III.14 – Performance	**Yes**	**No**	**NA**
79.	Can better data structures or more efficient algorithms be used?	☒	☐	☐
80.	Are logical tests arranged such that the often successful and inexpensive tests precede the more pensive and less frequently successful tests?	☒	☐	☐
81.	Can the cost of recomputing a value be reduced by computing it once and storing the results?	☒	☐	☐
82.	Is every result that is computed and stored actually used?	☒	☐	☐
83.	Can a computation be moved outside a loop?	☒	☐	☐
84.	Are there tests within a loop that do not need to be done?	☒	☐	☐
85.	Can a short loop be unrolled?	☒	☐	☐
86.	Are there two loops operating on the same data that can be combined into one?	☒	☐	☐
87.	Are frequently used variables declared register?	☒	☐	☐
88.	Are short and commonly called methods declared inline?	☒	☐	☐
89.	Are timeouts or error traps used for external device accesses?	☒	☐	☐
IV - INCONSISTENCY OBJECTIVE				
#	IV.1 – Performance	**Yes**	**No**	**NA**
90.	Are there any code implement in inconsistent way?	☐	☒	☐
V – AMBIGUITY OBJECTIVE				
#	V.1 – Variable and Constant Declaration	**Yes**	**No**	**NA**
91.	Are there variables with confusingly similar names?	☒	☐	☐

92.	Are all variables properly defined with meaningful, consistent, and clear names?	☒	☐	☐
#	V.2 – Performance	**Yes**	**No**	**NA**
93.	Are any modules excessively complex and should be restructured or split into multiple routines?	☒	☐	☐
VI – REDUNDANCE OBJECTIVE				
#	VI.1 – Variables	**Yes**	**No**	**NA**
94.	Are there any redundant or unused variables or attributes?	☒	☐	☐
95.	Could any non-local variables be made local?	☒	☐	☐
#	VI.2 – Method Definition	**Yes**	**No**	**NA**
96.	Are there any uncalled or unneeded methods?	☒	☐	☐
#	VI.3 – Performance	**Yes**	**No**	**NA**
97.	Can any code be replaced by calls to external reusable objects?	☒	☐	☐
98.	Are there any blocks of repeated code that could be condensed into a single method?	☒	☐	☐
99.	Are there any leftover stubs or test routines in the code?	☒	☐	☐
VII – SIDE-EFFECT OBJECTIVE				
#	VII.1 – Method Definition	**Yes**	**No**	**NA**
100.	After changing of prototype of method, Have class which calls it considered yet?	☒	☐	☐
#	VII.2 – Data Base	**Yes**	**No**	**NA**
101.	Do Upgrading and Migration process follow up changing of structures or contents of a project's data base?	☒	☐	☐
VIII – Paths of improvements				
#	VIII.1 – Major identified errors			
102.	1. Erros de Sintaxe nas Strings e Aspas: o Aspas Incorretas: Uso inconsistente de aspas simples e duplas, por exemplo, $_SERVER[' REQUEST_METHOD'] contém um espaço desnecessário e misturas de aspas não fechadas corretamente. o Strings Não Fechadas: Em $distritoUtilizador = $_POST['distrito'] ?? "*;, a aspa dupla não está fechada. 2. Uso Incorreto de Funções e Operadores: o Função Inválida: lempty não é uma função válida em PHP; provavelmente deveria ser empty. o Operador de Array Incorreto: in array deve ser substituído por in_array. 3. Erros na Sintaxe de Controle de Fluxo: o Delimitação Incorreta: Uso de parênteses (e) ao invés de chaves {} para blocos if e else. o Caracteres Inesperados: Presença de caracteres como 3 else e 3)5, que não fazem sentido no contexto do código. 4. Erros na Manipulação de Variáveis: o Variáveis Sem o Símbolo $: Exemplo, SordemFiltrada deveria ser $ordemFiltrada.			

- Atribuição Incorreta: $resultado = $incubadoras[0] 2? null; está com sintaxe inválida.

5. Erros na Referência a Objetos e Constantes:
   - Referência de Objeto Incorreta: Uso de PD0::FETCH_ASSOC ao invés de PDO::FETCH_ASSOC.
     - Símbolo de Acesso de Objeto Inválido: Uso de › em vez de -> em $pdo-›query.

6. Manipulação Incorreta de Dados:
   - Uso de unset: unset($incubadora); dentro do loop pode causar comportamentos inesperados ao modificar o array enquanto está a ser iterado.

7. Falta de Segurança nas Consultas ao Banco de Dados:
   - Injeção de SQL: Ausência de prepared statements para evitar vulnerabilidades de injeção de SQL.

8. Lógica de Negócio Confusa:
   - Ordenação e Prioridade: A lógica para ordenar as incubadoras com base em diferentes critérios está confusa devido aos erros de sintaxe e estrutura.

(O código SQL está representado na página 12)

- Código SQL corrigido:

```php
// Processar a requisição do formulário
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $distritoUtilizador = $_POST['distrito'] ?? '';

    if (isset($_POST['ordem']) && is_array($_POST['ordem']) && !empty($distritoUtilizador)) {
        $ordem = $_POST['ordem']; // Ordem dos campos enviada pelo utilizador
        $campoPrioritario = $ordem[0] ?? null; // Primeiro campo como prioridade principal

        // Validar campos válidos
        $camposValidos = ['localizacao', 'estacionamento', 'escritorios', 'valor', 'area'];
        $ordemFiltrada = array_filter($ordem, fn($campo) => in_array($campo, $camposValidos));

        if (count($ordemFiltrada) === count($camposValidos) && $campoPrioritario) {
            // Buscar todas as incubadoras do banco de dados
            $stmt = $pdo->query("SELECT * FROM incubadoras");
            $incubadoras = $stmt->fetchAll(PDO::FETCH_ASSOC);

            if ($campoPrioritario === 'localizacao') {
                // Calcular distâncias e ordenar as incubadoras
                foreach ($incubadoras as &$incubadora) {
                    $localizacao = $incubadora['localizacao'];
                    $incubadora['distancia'] = $distancias[$distritoUtilizador][$localizacao] ?? PHP_INT_MAX;
                }
                unset($incubadora);

                // Ordenar por distância (menor distância primeiro)
                usort($incubadoras, fn($a, $b) => $a['distancia'] <=> $b['distancia']);
            } else {
                // Ordenar com base no campo prioritário
                usort($incubadoras, function ($a, $b) use ($campoPrioritario) {
                    if ($campoPrioritario === 'valor') {
                        return $a[$campoPrioritario] <=> $b[$campoPrioritario]; // Menor valor primeiro
                    } else {
                        return $b[$campoPrioritario] <=> $a[$campoPrioritario]; // Maior valor primeiro
                    }
                });
            }

            // Resultado final
            $resultado = $incubadoras[0] ?? null; // Retorna o primeiro resultado
        } else {
            $erro = "A ordem enviada é inválida ou faltam campos.";
        }
    } else {
        $erro = "Por favor, selecione seu distrito e defina a ordem dos campos.";
    }
}
```

Figura i.: Código SQL.

VIII.2 – Improvements suggestions

103.

1. Correção da Sintaxe:

- Aspas e Strings: Verifique todas as strings para garantir que as aspas estão corretamente abertas e fechadas.
- Uso de Parênteses e Chaves: Utilize chaves {} para delimitar blocos de código em estruturas de controle como if e else.

2. Uso Correto de Funções e Operadores:

- Funções Válidas: Substitua lempty por empty e in array por in_array.
- Finalização de Instruções: Utilize ponto e vírgula ; no final de cada instrução PHP.

3. Correção na Manipulação de Variáveis:

- Uso do Símbolo $: Assegure-se de que todas as variáveis estão precedidas por $, por exemplo, $ordemFiltrada.
- Atribuições Válidas: Revise e corrija atribuições de variáveis para garantir que seguem a sintaxe correta, como $resultado = $incubadoras[0] ?? null;.

4. Referência Correta a Objetos e Constantes:

- Constantes PDO: Utilize PDO::FETCH_ASSOC corretamente para buscar os resultados.
- Símbolo de Acesso Correto: Substitua › por -> para acessar métodos e propriedades de objetos.

5. Melhoria na Manipulação de Dados:

- Evitar unset em Loops: Remova ou revise o uso de unset($incubadora); dentro do loop para evitar a modificação inesperada do array.

4. Implementação de Segurança:

- Prepared Statements: Utilize prepared statements com parâmetros vinculados para todas as consultas ao banco de dados, prevenindo injeção de SQL.
- Sanitização de Entrada: Valide e sanitize todas as entradas recebidas do utilizador antes de processá-las ou armazená-las.

5. Refatorarão da Lógica de Negócio:

- Clareza na Ordenação: Separe a lógica de ordenação em funções distintas para cada critério, melhorando a legibilidade e manutenção do código.
- Verificação de Prioridade: Assegure-se de que a lógica que determina o campo prioritário está clara e corretamente implementada.

6. Tratamento de Erros Adequado:

- Mensagens de Erro Claras: Forneça mensagens de erro mais descritivas para facilitar a identificação de problemas.
- Blocos Try-Catch: Implemente blocos try-catch ao realizar operações de banco de dados para capturar e lidar com exceções.

7. Melhoria na Legibilidade e Manutenção:

- Indentação Consistente: Utilize uma indentação consistente para facilitar a leitura do código.
- Comentários Explicativos: Adicione comentários que expliquem trechos complexos ou a lógica por trás de determinadas implementações.

8. Testes e Validação:

- Testes Unitários: Implemente testes para verificar a funcionalidade de cada parte do código.
- Validação de Fluxo: Teste o fluxo completo da aplicação para garantir que todas as funcionalidades operam conforme o esperado após as correções.

(O código SQL corrigido está representado nas páginas 17/18)

- Código SQL corrigido:

```php
<?php
// Supondo que a conexão PDO ($pdo) e o array de distâncias ($distancias) já estejam definidos anteriormente

// Processar a requisição do formulário
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Obter o distrito do utilizador, com valor padrão "*"
    $distritoUtilizador = $_POST['distrito'] ?? "*";

    // Verificar se 'ordem' está definido, é um array e o distrito não está vazio
    if (isset($_POST['ordem']) && is_array($_POST['ordem']) && !empty($distritoUtilizador)) {
        // Ordem dos campos enviada pelo utilizador
        $ordem = $_POST['ordem'];

        // Primeiro campo como prioridade principal
        $campoPrioritario = $ordem[0] ?? null;

        // Validar campos válidos
        $camposValidos = ['localizacao', 'estacionamento', 'escritorios', 'valor', 'area'];
        $ordemFiltrada = array_filter($ordem, fn($campo) => in_array($campo, $camposValidos));

        // Verificar se todos os campos válidos foram enviados e se há um campo prioritário
        if (count($ordemFiltrada) === count($camposValidos) && $campoPrioritario) {
            try {
                // Buscar todas as incubadoras do banco de dados usando prepared statements
                $stmt = $pdo->prepare("SELECT * FROM incubadoras");
                $stmt->execute();
                $incubadoras = $stmt->fetchAll(PDO::FETCH_ASSOC);

                if ($campoPrioritario === 'localizacao') {
                    // Calcular distâncias e ordenar as incubadoras
                    foreach ($incubadoras as &$incubadora) {
                        $localizacao = $incubadora['localizacao'];
                        // Calcular a distância ou usar PHP_INT_MAX se a distância não estiver disponível
                        $incubadora['distancia'] = $distancias[$distritoUtilizador][$localizacao] ?? PHP_INT_MAX;
                    }
                    unset($incubadora); // Remover referência após o loop

                    // Ordenar por distância (menor distância primeiro)
                    usort($incubadoras, fn($a, $b) => $a['distancia'] <=> $b['distancia']);
```

```php
            } else {
                // Ordenar com base no campo prioritário
                usort($incubadoras, function ($a, $b) use ($campoPrioritario) {
                    if ($campoPrioritario === 'valor') {
                        // Menor valor primeiro
                        return $a[$campoPrioritario] <=> $b[$campoPrioritario];
                    } else {
                        // Maior valor primeiro para outros campos
                        return $b[$campoPrioritario] <=> $a[$campoPrioritario];
                    }
                });
            }

            // Resultado final: retorna a primeira incubadora ou null se não houver resultados
            $resultado = $incubadoras[0] ?? null;

        } catch (PDOException $e) {
            // Tratamento de erro na conexão ou consulta ao banco de dados
            $erro = "Erro ao acessar o banco de dados: " . htmlspecialchars($e->getMessage());
        }
    } else {
        $erro = "A ordem enviada é inválida ou faltam campos.";
    }
} else {
    $erro = "Por favor, selecione seu distrito e defina a ordem dos campos.";
}

// Exibir resultado ou erro
if (isset($resultado)) {
    // Processar o resultado, por exemplo, exibir informações da incubadora
    echo "<h2>Incubadora Recomendada:</h2>";
    echo "<p><strong>Nome:</strong> " . htmlspecialchars($resultado['nome']) . "</p>";
    echo "<p><strong>Localização:</strong> " . htmlspecialchars($resultado['localizacao']) . "</p>";
    echo "<p><strong>Estacionamento:</strong> " . htmlspecialchars($resultado['estacionamento']) . "</p>";
    echo "<p><strong>Escritórios:</strong> " . htmlspecialchars($resultado['escritorios']) . "</p>";
    echo "<p><strong>Valor:</strong> " . htmlspecialchars($resultado['valor']) . "</p>";
    echo "<p><strong>Área:</strong> " . htmlspecialchars($resultado['area']) . " m²</p>";
    // Adicione mais detalhes conforme necessário
```

```php
    // Adicione mais detalhes conforme necessário
} elseif (isset($erro)) {
    // Exibir mensagem de erro
    echo "<p style='color:red;'>" . htmlspecialchars($erro) . "</p>";
}
}
?>
```

Figura ii, iii e iv.: Código SQL corrigido.

104. 9.