Atividade Caixa Branca 1 – O Código foi devidamente documentado? - Há somente um comentário //INSTRUÇÃO SQL que não explica quase nada. -O SQL está incompleto (select nome from usuarios foi comentado). - Não há comentários que que explique para que serve a classe USER e seus métodos. -Recursos não fechados (Connection/Statement/ResultSet) → Leak de conexões, queda da aplicação. - Tratamento de exceção vazio → Dificulta depuração (alto impacto operacional). - O método conectarBD() não tem comentários que descreve sua função, parâmetros ou comportamento em caso de erro. - O metédo verificarUsuarios que usa como parâmetro Login e Senha contém um comentário incompleto sobre a instrução SQL, mas não explica claramente o que método faz, quais são seus parâmetros, seu retorno e exceções. - As variáveis e atributos da classe não tem comentários explicativos. ¬- Não há tratamento de forma clara de exceções no código, e a ausência de erros torna difícil entender falhas em tempo de execução. - Uso de variável de instância result desnecessária resulta em Design frágil / bugs potenciais. - Não valida conn == null antes de usar NullPointerException possível. - Concatenação de SQL sem espaços Pode produzir SQL com sintaxe incorreta.

2 – As variáveis se constantes possuem nomenclatura? R: As variáveis e constantes possuem nomenclatura adequada e clara para seu propósito. As variáveis que tem seus nomes descritivos e coerentes são nome, login, senhaPlain, nomeDb, senhaHash, e a constante LOGGER que segue padrão estático final para Logger. - Os nomes das varáveis indicam claramente a função e o tipo de dado armazenado, facilitando legibilidade.

3 – Existem legibilidade e organização no código ? - A classe necessita de mais organização. Mistura funções diferentes (dados do usuário, conexão com banco, lógica de login) tudo no mesmo lugar, o que dificulta manutenção. - Não há separação de responsabilidades O código não segue a estrutura ideal (User como modelo, UserDAO para banco, AuthService para autenticação). Isso deixa o projeto confuso. - O SQL precisa de mais segurança e ser melhor estruturado A consulta é construída por concatenação de strings, o que torna o código difícil de ler e abre falhas de segurança (SQL Injection). - O tratamento de exceções é vazio Os blocos catch não mostram o erro nem tratam nada. Isso esconde problemas reais e atrapalha o diagnóstico. - Variáveis públicas sem necessidade nome e result são públicas quando deveriam ser privadas. Isso fere o encapsulamento e permite alteração indevida. - O driver JDBC está incorreto A classe do driver está errada, o que faria a conexão falhar, e isso fica invisível porque o erro é ignorado. - Recursos não são fechados corretamente Connection, Statement e ResultSet ficam abertos, o que causa vazamento de conexão e pode derrubar a aplicação. - Lógica confusa dentro do método O fluxo do método é longo e mistura conexão, SQL, extração de dados e manipulação de atributos, reduzindo a clareza. - Falta de documentação e

comentários úteis Não há explicação sobre o que cada método faz, nem comentários que ajudem na compreensão. - Não segue boas práticas de Java Coisas como nomenclatura, encapsulamento, uso de PreparedStatement, organização em camadas e clareza do código não estão sendo aplicadas.

4 – Todos os NullPointers foram tratados ? R: Ausência de verificações prévias: Algumas variáveis parecem ser usadas diretamente sem garantia de que foram inicializadas antes. Isso pode gerar NullPointerException durante a execução. Objetos dependentes de retorno de métodos: Caso algum método retorne null e o código utilize o resultado diretamente (por exemplo, acessando atributos ou chamando métodos), o risco de NullPointerException permanece. Falta de condicionais de segurança: Não foram observados trechos condicionais do tipo if (obj != null) para proteger operações críticas. Dependências não validadas: Elementos como objetos de serviço, listas, parâmetros de entrada ou atributos internos não mostram validação prévia. Tratamento insuficiente de cenários inesperados: O código não apresenta mecanismos alternativos caso um dado essencial esteja ausente, o que reforça o risco de falhas desse tipo.

5 – As conexões utilizadas foram fechadas ? Sim, todas as conexões utilizadas no código foram fechadas corretamente. No método verificarUsuario, a conexão (Connection), o statement (PreparedStatement) e o result set (ResultSet) estão dentro de blocos try-with-resources. Esse mecanismo do Java garante que todos os recursos sejam fechados automaticamente ao final do bloco, mesmo se ocorrer uma exceção durante a execução. A ordem de fechamento é automática e segura: primeiro o ResultSet, depois o PreparedStatement, e por último a Connection. Isso elimina qualquer risco de vazamento de recursos ou de conexões abertas.