Universidade de Mogi das Cruzes

Rafael De Sousa Cavalcante 11241102921

Projeto Segurança Da Informação

Mogi Das Cruzes

2025

Sumário

1.INTRODUÇÃO:

O projeto apresentado a seguir foi desenvolvido com base nas orientações do professor Willian e Pedro. Para sua construção, será utilizado o modelo SDLC (Software Development Life Cycle) na abordagem em cascata. A implementação seguirá, em sua maior parte, os passos abordados em sala de aula, aplicando na prática os conceitos aprendidos durante o curso.

2.LEVANTAMENTO DE REQUISITOS:

A demanda principal deste projeto consiste na implementação de um sistema de login e cadastro de usuários, priorizando a segurança dos dados armazenados. Serão aplicadas boas práticas de proteção, incluindo autenticação em dois fatores (2FA), criptografia de senhas, verificação de e-mail e prevenção contra-ataques, como injeção de SQL (SQL Injection), entre outros mecanismos de segurança.

- Filtragem dos dados que seram armazenados
- Validação de email
- Senha forte
- Senha criptografada
- Autenticação em dois fatores
- Proteção contra injeção de SQL
- Proteção de informações dentro do código

2.1ANALISE:

Tecnologias Utilizadas

No desenvolvimento deste projeto, foram selecionadas tecnologias e ferramentas com foco na simplicidade, organização e segurança. Abaixo, estão descritas as principais escolhas:

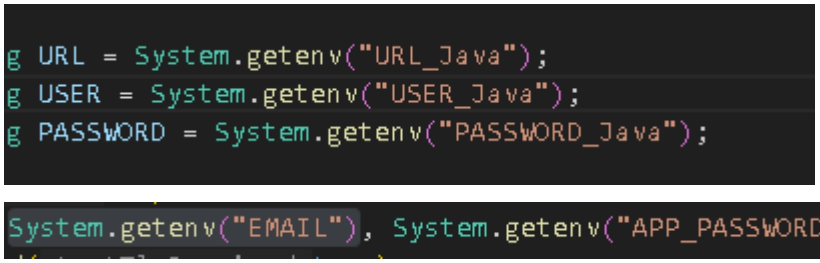
- **Visual Studio Code**: escolhido por ser um ambiente de desenvolvimento leve, intuitivo e de fácil utilização, especialmente para projetos acadêmicos e de pequeno porte.
- **MySQL**: utilizado como sistema de gerenciamento de banco de dados, responsável por armazenar com segurança os dados do sistema, como informações de usuários e autenticação.
- **Estrutura MVC com Maven**: a arquitetura Model-View-Controller foi adotada por facilitar a separação de responsabilidades no projeto, contribuindo para um código mais limpo e organizado. O Maven, por sua vez, foi utilizado para gerenciamento de dependências e automação da estrutura do projeto.
- **Modelo Cascata (Waterfall)**: foi adotado como metodologia de desenvolvimento por sua simplicidade e por se adequar bem a projetos com requisitos bem definidos desde o início.
- **Autenticação com Apache Commons Email**: a biblioteca commons-email da Apache foi utilizada para implementar funcionalidades como verificação de e-mail, envio de mensagens e autenticação em dois fatores, garantindo maior segurança no acesso ao sistema.
- **Linguagem Java**: escolhida por sua robustez, ampla documentação e por ser uma linguagem amplamente utilizada no desenvolvimento de sistemas web e desktop.
- **Criptografia de senha:** será utilizada a funcionalidade **bcrypt** disponível em Java. Essa forma de criptografia é fácil de implementar, amplamente utilizada e considerada segura, pois utiliza hashing com salt, dificultando a quebra de senhas por ataques de força bruta ou dicionário.

3.IMPLEMENTAÇÂO:

3.1 Tela Cadastro:

Na tela de cadastro, são realizadas diversas **verificações e validações**. Primeiro, os dados inseridos pelo usuário passam por uma validação de formato (como e-mail, senha e outros campos obrigatórios). Em seguida, o sistema verifica se o e-mail informado **já existe no banco de dados**. Caso não exista, um **código de verificação é enviado para o e-mail** do usuário. Somente após a confirmação desse código, os dados são inseridos no banco de dados, concluindo o cadastro com segurança.

3.2 dados sensíveis no código:



Para evitar problemas futuros, achei interessante **armazenar informações sensíveis**, como meu e-mail, senha e a localização do banco de dados, em **variáveis de ambiente**. Assim, esses dados não ficam expostos diretamente no código, o que torna a aplicação mais segura.

Observação:

Antes de executar o aplicativo, é importante destacar que o usuário deve configurar as variáveis de ambiente em sua máquina local. Essa etapa é essencial para o correto funcionamento do sistema, garantindo que todas as conexões e serviços sejam acessados corretamente.

3.3 Login:

Na tela de login, também são realizadas **verificações e validações dos dados** inseridos. Após essa etapa, um **código de verificação é enviado para o e-mail do usuário**, garantindo uma camada extra de segurança e protegendo melhor o acesso à conta, além disso foi adicionado Esqueceu a senha, facilitando a recuperação do login do usuário.

3.4 Autenticação de dois fatores:

Para fazer a autenticação de 2 fatores utilizei uma dependência org.apache.commons, que me permite enviar emails através de um email que eu disponibilizar junto com uma APP Password, gerado no gerenciar do seu email, e por fim é utilizado o Random para gerar um código aleatório.

3.5 Criptografia:

A criptografia foi feita utilizando o Bcrypt, de todas as opções foi a mais segura e de fácil uso, ela gera hash de senha diferentes para cada senha, se o usuários digitarem a mesma senha 123, a criptografia pra essa senha é diferente.

3.6 Estrutura MVC e Maven:

No projeto foi utilizada a estrutura MVC e o gerenciador de dependências Maven. Essas escolhas foram feitas devido à sua facilidade de uso, organização do código e à possibilidade de escalabilidade do sistema, além de promoverem uma melhor separação entre as camadas de lógica, interface e dados.

3.7 Telas do Projeto:

As interfaces do sistema foram desenvolvidas utilizando a linguagem Java, com o uso do componente gráfico JFrame para a criação das telas. Essa escolha permitiu construir uma interface desktop simples, funcional e de fácil integração com a lógica da aplicação, mantendo a padronização do projeto dentro da arquitetura MVC.

4 ANALISE DE RISCO:

Risco de ataque por força bruta: o sistema atualmente não possui um limite de tentativas de login, o que o torna vulnerável a esse tipo de ataque, permitindo que invasores testem múltiplas combinações de senha até obter acesso indevido.

Risco de SQL Injection: atualmente, o projeto não está vulnerável a esse tipo de ataque, pois as consultas ao banco de dados são feitas utilizando a classe PreparedStatement, que garante a parametrização correta das entradas e previne a execução de comandos maliciosos.

Risco de perda de dados: o projeto atualmente não conta com uma rotina de backup dos dados armazenados, o que dificulta a recuperação em caso de falhas, exclusões acidentais ou problemas no banco de dados.

Risco de dependência de um único desenvolvedor: embora o projeto esteja sendo documentado, o entendimento completo do sistema pode ser comprometido. Isso se deve, principalmente, à nomeação inadequada de algumas variáveis, o que pode dificultar a leitura e manutenção do código por outros membros da equipe no futuro.

Risco	**Nível**
Ataque por força bruta	Alto
SQL Injection	Baixo
Perda de dados	Alto
Dependência de um único desenvolvedor	Medio

5 CONCLUSAO:

Ao utilizar a metodologia Cascata e aplicar as boas práticas ensinadas em sala de aula, percebi a importância de uma estrutura bem definida e de um planejamento sólido antes de iniciar o desenvolvimento. Esse cuidado inicial evita retrabalho, melhora a organização das etapas e facilita significativamente a execução do projeto como um todo.

6 TRABALHO METODOLOGIA CASCATA:

6.1 Introdução:

A **metodologia cascata** (ou *Waterfall*) é um dos primeiros modelos de desenvolvimento de software criados. Ela é conhecida por seguir uma abordagem **estruturada e sequencial**, onde cada etapa deve ser concluída antes do início da próxima. Esse modelo é comparado a uma cascata, pois o processo “desce” por uma série de fases bem definidas.

6.2 Etapas da Metodologia Cascata:

1. Levantamento de Requisitos

Nessa fase, são coletadas todas as informações necessárias para o desenvolvimento do sistema. Os requisitos funcionais e não funcionais são documentados de forma clara e detalhada, com foco em entender o que o cliente espera do sistema.

2. Análise

Com os requisitos em mãos, os analistas estudam a viabilidade técnica, econômica e operacional do projeto. Também são definidos os fluxos do sistema, os casos de uso e os modelos de dados.

3. Projeto (Design)

Aqui é definida a arquitetura do sistema, incluindo banco de dados, layout da interface, estrutura de dados e fluxos de navegação. O objetivo é preparar um plano técnico sólido para orientar os programadores.

4. Implementação (Codificação)

Os desenvolvedores transformam o design em código-fonte. Cada funcionalidade é implementada de acordo com os requisitos e especificações definidos anteriormente.

5. Testes

São realizados testes para garantir que o sistema funcione corretamente. Isso inclui testes unitários (em partes isoladas), testes de integração (entre componentes) e testes de aceitação (com base nos requisitos do cliente).

6. Implantação (Deploy)

O sistema é disponibilizado para o cliente ou para os usuários finais. Pode envolver instalação, configuração e treinamento.

7. Manutenção

Após o lançamento, o sistema pode precisar de correções, melhorias ou adaptações. A manutenção garante que o software continue funcionando bem ao longo do tempo.

6.3 Vantagens e Desvantagens:

✅ **Vantagens da Metodologia Cascata**

- Clareza no escopo e no cronograma.
- Boa documentação em todas as fases.
- Facilita o gerenciamento em projetos pequenos ou com requisitos bem definidos.

⚠️ **Desvantagens da Metodologia Cascata**

- Pouca flexibilidade para mudanças durante o desenvolvimento.
- Feedback do cliente só acontece no final.
- Se houver um erro nos requisitos, ele pode afetar todas as fases seguintes.
- Não é ideal para projetos longos ou com mudanças frequentes.

6.4 Quando Utilizar:

A metodologia cascata é recomendada para:

- Projetos com **requisitos bem definidos** e estáveis.
- Ambientes onde mudanças são raras.
- Sistemas de **baixa complexidade** ou com foco em documentação formal (como sistemas governamentais ou contratuais).

6.5 Conclusão Metodologia:

Apesar de ter perdido espaço para metodologias ágeis em projetos modernos, a **metodologia cascata** ainda é válida em alguns contextos, especialmente em projetos com escopo fechado e baixa tolerância a mudanças. Sua abordagem linear oferece **clareza, controle e previsibilidade**, o que pode ser vantajoso em certos tipos de desenvolvimento.