ProjetoFinalPSSI — Guia detalhado de funcionalidades e como implementar Autor: Assistente (ChatGPT) Repositório consultado: https://github.com/Jesseh78/ProjetoFinalPSSI (Observação: o acesso direto ao conteúdo dos PDFs no repositório apresentou erro no visualizador do GitHub; este guia foi gerado a partir dos nomes de arquivos encontrados no repositório e boas práticas para um "Sistema de Denúncias" em Java. Ver referências no final.)

Sumário 1) O que eu encontrei no repositório 2) Visão geral das funcionalidades esperadas 3) Passo a passo: como rodar/inspecionar o repositório localmente 4) Implementação detalhada (por funcionalidade) - Modelo de dados (SQL) - Backend (exemplo com Spring Boot / JPA) - Upload de anexos - Autenticação e autorização - Painel administrativo e status das denúncias - Relatórios e exportação (PDF / CSV) - Notificações por e-mail - Segurança básica a aplicar 5) Como você mesmo pode implementar mudanças (ex.: adicionar "prioridade") 6) Funcionalidades recomendadas para continuar e alinhar ao modelo dos PDFs 7) Boas práticas de versão e deploy 8) Referências (páginas consultadas)

1) O que eu encontrei no repositório - Nome do repositório: ProjetoFinalPSSI — "Sistema de denuncias em JAVA". - Arquivos/documentos listados na raiz (observados na página do repositório): * Projeto_Sistema_Denuncias.pdf (documentação do projeto) * Roadmap_Desenvolvimento_Sistema.pdf (roadmap/cronograma) * Modelo_Banco_Sistema_Denuncias.pdf (modelo do banco de dados) - Existe uma pasta indicada como: sistema-denuncias (provavelmente com o código fonte). (Estas informações foram obtidas na página do repositório do GitHub.)

2) Visão geral das funcionalidades esperadas Com base nos nomes de arquivos e no tipo comum para um sistema de denúncias, funcionalidades típicas: - Cadastro de usuários (citizens e administradores) com login. - Formulário para abrir uma denúncia (título, descrição, categoria, local, anexos). - Lista/painel de denúncias com filtros (por status, categoria, data). - Fluxo de tratamento: status (Aberta → Em análise → Resolvida/Recusada). - Upload de anexos (imagens, documentos) associados à denúncia. - Painel administrativo com possibilidade de alterar status, atribuir responsáveis. - Relatórios exportáveis e geração de PDFs. - Roadmap e modelo do banco para normalização dos dados.

3) Passo a passo: como rodar/inspecionar o repositório localmente (a) clonar o repositório git clone https://github.com/Jesseh78/ProjetoFinalPSSI.git cd ProjetoFinalPSSI

(b) listar o conteúdo e localizar o código ls -la # procure por pastas como "sistema-denuncias", "src", "pom.xml" ou "build.gradle" find . -maxdepth 2 -type f -name "pom.xml" -o -name "build.gradle" -o -name "*.java"

(c) se for Maven (pom.xml presente): mvn clean install mvn spring-boot:run # se for Spring Boot # ou executar o JAR java -jar target/seu-app.jar

(d) se for projeto Java sem build tool (pelo menos 1 classe com main): # compile find src -name "*.java" > sources.txt javac @sources.txt -d out java -cp out com.example.MainClass

(e) verificar o banco de dados - Se houver arquivos SQL (ex.: schema.sql) ou instruções nos PDFs, importe para MySQL/Postgres: mysql -u root -p seu_banco < modelo_banco.sql

4) Implementação detalhada (por funcionalidade) ---------- Abaixo há exemplos concretos e comandos para implementar/estender cada funcionalidade. Use-os como referência e copie/cole trecho por trecho no seu projeto.

A) Modelo de dados (SQL) — Exemplo (MySQL) ========== -- tabel usuários: CREATE TABLE usuarios (id INT AUTO_INCREMENT PRIMARY KEY, nome VARCHAR(150) NOT NULL, VARCHAR(200) NOT NULL UNIQUE, senha_hash VARCHAR(255) NOT NULL, papel VARCHAR(20) NOT NULL, -- E 'CIDADAO', 'ADMIN' criado_em TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

-- tabela de denúncias: CREATE TABLE denuncias (id INT AUTO_INCREMENT PRIMARY KEY, titulo VARCHAR(255 NOT NULL, descricao TEXT NOT NULL, categoria VARCHAR(100), localizacao VARCHAR(255), status VARCHAR(30) DEFAULT 'ABERTA', -- ABERTA, EM_ANALISE, RESOLVIDA, RECUSADA usuario_id INT, criado_em TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (usuario_id) REFERENCES usuarios(id));

-- tabela de anexos: CREATE TABLE anexos (id INT AUTO_INCREMENT PRIMARY KEY, denuncia_id INT NOT NUL nome_arquivo VARCHAR(255) NOT NULL, caminho VARCHAR(500) NOT NULL, enviado_em TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (denuncia_id) REFERENCES denuncias(id) ON DELETE CASCADE);

-- tabela de comentários/registro de ações: CREATE TABLE comentarios (id INT AUTO_INCREMENT PRIMARY KEY,

denuncia_id INT NOT NULL, usuario_id INT, texto TEXT, criado_em TIMESTAMP DEFAULT CURRENT_TIMESTAM
FOREIGN KEY (denuncia_id) REFERENCES denuncias(id));

B) Backend (exemplo em Spring Boot + JPA) ========== # 1) Entidade Denunc
(exemplo simplificado) package com.seuprojeto.model;

import javax.persistence.*; import java.time.LocalDateTime;

@Entity @Table(name = "denuncias") public class Denuncia { @Id @GeneratedValue(strategy =
GenerationType.IDENTITY) private Long id;

private String titulo; @Column(columnDefinition = "TEXT") private String descricao; private
String categoria; private String localizacao; private String status = "ABERTA";

private LocalDateTime criadoEm = LocalDateTime.now();

@ManyToOne @JoinColumn(name = "usuario_id") private Usuario autor;

// getters e setters }

2) Repository package com.seuprojeto.repository; import
org.springframework.data.jpa.repository.JpaRepository; public interface DenunciaRepository extends
JpaRepository<Denuncia, Long> { List<Denuncia> findByStatus(String status); List<Denuncia>
findByCategoria(String categoria); }

3) Service (ex.: criação) @Service public class DenunciaService { @Autowired private DenunciaRepository
repo;

public Denuncia criar(Denuncia d) { // validações básicas if (d.getTitulo()==null ||
d.getTitulo().isBlank()) throw new IllegalArgumentException("Título obrigatório"); return repo.save(d);
} }

4) Controller (REST) @RestController @RequestMapping("/api/denuncias") public class DenunciaController {
@Autowired private DenunciaService service;

@PostMapping public ResponseEntity<Denuncia> criar(@RequestBody Denuncia d) { Denuncia criado =
service.criar(d); return ResponseEntity.status(HttpStatus.CREATED).body(criado); }

@GetMapping public List<Denuncia> listar(@RequestParam(required=false) String status) { if (status !=
null) return service.buscarPorStatus(status); return service.listarTodas(); } }

C) Upload de anexos (exemplo Spring Boot) ==========
@PostMapping("/{id}/anexos") public ResponseEntity<?> upload(@PathVariable Long id, @RequestParam("file")
MultipartFile file) throws IOException { // validar tamanho e tipo String folder =
"/var/app/uploads/denuncias/" + id; Files.createDirectories(Paths.get(folder)); String filename =
UUID.randomUUID().toString() + "_" + file.getOriginalFilename(); Path path = Paths.get(folder, filename);
Files.copy(file.getInputStream(), path); // salvar registro em tabela 'anexos' }

D) Autenticação e senha segura ========== - Nunca salvar senhas em texto puro. Use
BCrypt. Exemplo (Spring Security): @Bean public PasswordEncoder passwordEncoder() { return new
BCryptPasswordEncoder(); }

Ao registrar: String hash = passwordEncoder.encode(plainPassword);

Ao autenticar: passwordEncoder.matches(rawPassword, storedHash);

E) Painel administrativo ========== - Rotas REST protegidas por roles (ROLE_ADMIN). - Endpoints
para: * alterar status (PATCH /api/denuncias/{id}/status) * atribuir responsável * listar denúncias
filtradas e com paginação (Pageable)

Exemplo de alteração de status: @PatchMapping("/{id}/status") public ResponseEntity<?>
mudarStatus(@PathVariable Long id, @RequestBody Map<String,String> body) { String novo =
body.get("status"); service.atualizarStatus(id, novo, usuarioAtuante); return
ResponseEntity.ok().build(); }

F) Relatórios e exportação ========== - Para exportar em CSV: monte linhas via StringBuilder e responda com content-type text/csv. - Para exportar em PDF: use bibliotecas como iText (ATENÇÃO: licenças) ou OpenPDF. Alternativa: gerar HTML e usar wkhtmltopdf. Exemplo CSV: String csv = "id;titulo;status;criado_em "; for (Denuncia d: lista) csv += d.getId()+";"+escape(d.getTitulo())+";"+d.getStatus()+";"+d.getCriadoEm()+" ";

G) Notificações por e-mail ========== - Configure JavaMailSender no Spring Boot. - Em eventos (ex.: mudança de status) envie um e-mail ao autor: SimpleMailMessage msg = new SimpleMailMessage(); msg.setTo(autor.getEmail()); msg.setSubject("Atualização da sua denúncia"); msg.setText("Sua denúncia "+d.getTitulo()+" mudou para: "+d.getStatus()); mailSender.send(msg);

H) Segurança (mínimos essenciais) ========== - Use PreparedStatements/JPA para evita SQL injection. - Faça validação de entrada (server-side). - Proteja endpoints com autenticação e autorização. - Evite expor caminhos de arquivos; gere nomes aleatórios para anexos. - Ative HTTPS (em produção use certificado). - Logs de auditoria (quem fez o quê e quando).

5) Como você mesmo pode implementar mudanças (exemplo prático: adicionar campo "prioridade") ---------- 1. Abra seu projeto local e identifique a entidade que representa "Denuncia". 2. Adicione o atributo: private String prioridade = "NORMAL"; // NORMAL, ALTA, BAIXA 3. Atualize a tabela (SQL) com: ALTER TABLE denuncias ADD COLUMN prioridade VARCHAR(10) DEFAULT 'NORMAL'; 4. Atualize DTOs, formulários e endpoints onde for necessário 5. Teste: - criar denúncia com prioridade - listar e filtrar por prioridade (repository: findByPrioridade) 6. Commit e push: git checkout -b feat/prioridade-denuncias git add . git commit -m "Adiciona campo prioridade em Denuncia" git push origin feat/prioridade-denuncias # depois abra PR no GitHub

6) Funcionalidades recomendadas para continuar e alinhar ao modelo dos PDFs ---------- Baseado nos nomes "Roadmap" e "Modelo_Banco", sugiro: - Completar o modelo de banco com índices, chaves estrangeiras e normalização. - Criar o roadmap de entregas em issues/kanban no GitHub (Projects). - Implementar testes automatizados (unit + integração). - Gerar documentação da API (Swagger/OpenAPI). - Implementar fila de processamento (ex.: para enviar e-mails via RabbitMQ se volume grande). - Camada de auditoria (LOG e histórico): manter histórico de mudanças de status. - Importação/Exportação de dados (csv, excel). - Dashboard com métricas (denúncias por categoria, tempo médio de resolução). - Segurança avançada: MFA, rate limiting, proteção contra upload de arquivos maliciosos. - UI responsiva e acessível (WCAG). - Preparar deployment em container com Docker + docker-compose (app + db).

7) Boas práticas de versão e deploy ---------- - Use branches por feature e commit pequeno. - Mantenha README com instruções de build/run. - Integração contínua (GitHub Actions): rodar testes e build. - Deploy: criar Dockerfile; exemplo simples: FROM eclipse-temurin:17-jre ARG JAR_FILE=target/*.jar COPY ${JAR_FILE} app.jar ENTRYPOINT ["java","-jar","/app.jar"]

8) Referências - Página do repositório (lista de arquivos observada). ■cite■turn0view0■ - Visualizador do GitHub apresentou erro ao carregar algumas visualizações (impossibilidade de baixar direto via visualizador). ■cite■turn1view0■turn3view2■

---------- Fim do guia ----------