

DOCUMENTAÇÃO JHEICANAMA

Matutino – ANAD2BM1



(61) 3035-3900



SIGA Área Especial para Indústria nº 02
Setor Leste - Gama - DF
CEP: 72445-020

JHEICAFREE SISTEMA WEB

Professor: Geovane da Costa Oliveira

ADS: 2 semestre

Matéria: Projeto Integrado de Certificação Scrum

Integrantes: Andreza Cassandra

Icaro Lins

Jheffiny Gervazio

Julia Ferreira

Maria Clara

Natália Rodrigues

Matutino – ANAD2BM1



(61) 3035-3900



SIGA Área Especial para Indústria nº 02
Setor Leste - Gama - DF
CEP: 72445-020

Resumo

O presente projeto tem como objetivo o desenvolvimento de uma solução tecnológica voltada à mediação de serviços temporários, abrangendo trabalhos freelance, por diárias e de curta duração. A plataforma funcionará como intermediadora entre contratantes e prestadores de serviços, permitindo a publicação de vagas com informações detalhadas sobre requisitos, condições e valores. Paralelamente, os profissionais terão acesso a mecanismos de busca e filtragem que possibilitarão a identificação de oportunidades compatíveis com suas habilidades e disponibilidade. A proposta busca unir usabilidade, praticidade e eficiência, garantindo que a interação entre contratante e contratado ocorra em um ambiente seguro, centralizado e transparente. Para o desenvolvimento da aplicação, serão empregadas as linguagens HTML, CSS e JavaScript, com foco em responsividade, navegabilidade e acessibilidade.



Sumário

| | |
|--------------------------------------------------|----|
| 1. Introdução | 5 |
| 2. Objetivo Específico | 6 |
| 2.1. Objetivo Geral | 7 |
| 3. Justificativa | 8 |
| 4. Metodologia | 9 |
| 5. Desenvolvimento..... | 10 |
| 5.1. ContractorService | 11 |
| 5.2. SecurityConfig..... | 12 |
| 5.3. ContractorRepository | 13 |
| 5.4. Candidate..... | 14 |
| 5.5. ContractorController e AuthController | 15 |
| 5.6. Desenvolvimento final | 16 |
| 6. Conclusão | 17 |
| 7. Referências bibliográficas..... | 18 |



1. Introdução

Com o avanço da tecnologia e a crescente digitalização dos processos de trabalho, novas formas de contratação vêm ganhando espaço no mercado, especialmente no que se refere a serviços temporários e trabalhos autônomos. Nesse cenário, este projeto propõe o desenvolvimento de uma plataforma digital destinada à intermediação de serviços freelance e por diárias, voltada tanto para indivíduos que buscam oportunidades de renda complementar quanto para aqueles que necessitam contratar profissionais para demandas pontuais. A proposta consiste em oferecer um ambiente centralizado, acessível e confiável, que permita a divulgação e a busca de oportunidades de forma prática e eficiente. Dessa maneira, objetiva-se reduzir barreiras de comunicação, otimizar o processo de contratação e ampliar a visibilidade de prestadores de serviços, contribuindo para o fortalecimento do mercado de trabalho informal e para a geração de novas oportunidades.



2. Objetivo Específico

Desenvolver um website responsivo e intuitivo utilizando as linguagens HTML, CSS e Java Script;

Disponibilizar funcionalidades que permitam ao contratante cadastrar e divulgar vagas com informações detalhadas sobre requisitos, valores e condições;

Implementar mecanismos de busca e filtragem que possibilitem ao prestador de serviços localizar oportunidades compatíveis com suas habilidades e disponibilidade;

Garantir a usabilidade e a navegabilidade do sistema, assegurando uma experiência satisfatória para os usuários;

Promover maior visibilidade para profissionais autônomos, ampliando suas possibilidades de inserção no mercado de trabalho;

Estimular a formalização e a organização dos processos de contratação, reduzindo a informalidade nas negociações atua.



2.1. Objetivo Geral

Desenvolver uma plataforma digital inovadora voltada para a intermediação de serviços temporários, abrangendo tanto trabalhos de natureza freelance quanto atividades realizadas por diárias, de forma a criar um ambiente virtual seguro, acessível e eficiente. O projeto busca atender às necessidades de dois públicos principais: os contratantes, que demandam profissionais qualificados e disponíveis de forma ágil, e os prestadores de serviços, que necessitam de uma ferramenta confiável para divulgar suas habilidades e conquistar novas oportunidades de renda.

Além de promover a aproximação entre essas partes, a plataforma pretende se diferenciar ao oferecer mecanismos de transparência e confiança, como avaliações mútuas, verificação de perfis e histórico de serviços prestados, possibilitando relações mais seguras e justas. Outro aspecto central é a preocupação com a inclusão e a democratização do acesso, garantindo que o sistema seja intuitivo, fácil de usar e adaptado a diferentes perfis de usuários, desde os mais experientes em tecnologia até aqueles com menor familiaridade digital.

A iniciativa também se propõe a fomentar o empreendedorismo individual e a valorização do trabalho temporário como alternativa flexível no mercado, contribuindo para o fortalecimento da economia colaborativa. Para isso, serão incorporados recursos tecnológicos modernos, que facilitem a comunicação, o gerenciamento de propostas e pagamentos, bem como ferramentas de filtragem personalizadas, permitindo que cada usuário encontre oportunidades ou prestadores de serviços que atendam de forma precisa às suas necessidades.

Portanto, o objetivo geral é criar uma solução digital que não apenas simplifique a intermediação entre contratantes e prestadores, mas que também se torne um espaço confiável, dinâmico e sustentável, capaz de gerar impacto positivo tanto no âmbito profissional quanto social



3. Justificativa

A crescente procura por trabalhos temporários e oportunidades freelance reflete mudanças significativas no mercado de trabalho contemporâneo, caracterizado pela flexibilidade, pela busca por autonomia e pela necessidade de complementação de renda. Contudo, ainda se observam dificuldades no processo de divulgação e de acesso a essas oportunidades, uma vez que muitos profissionais dependem de redes sociais, grupos informais ou indicações pessoais, o que limita a visibilidade e a confiabilidade das ofertas. Nesse contexto, a criação de uma plataforma digital unificada justifica-se pela necessidade de proporcionar um espaço seguro, centralizado e eficiente para a intermediação entre contratantes e prestadores de serviços. Além de contribuir para a organização e acessibilidade das informações, a solução proposta amplia as chances de empregabilidade de trabalhadores informais, ao mesmo tempo em que facilita a contratação para pessoas físicas e jurídicas que necessitam de serviços pontuais.



4. Metodologia

A empresa Jheicanama optou pela linguagem de programação Java, escolhida por nossos desenvolvedores devido à sua robustez, portabilidade e ampla aceitação no mercado. Essa linguagem permite a utilização da programação orientada a objetos, além de oferecer recursos como classes, métodos e interfaces. É importante destacar que, por ser executada em uma máquina virtual, garante maior acessibilidade e flexibilidade. Tendo em vista esses fatores, utilizamos o Java para o desenvolvimento do backend (sistema que o usuário não vê) do nosso sistema web de freelance. Já no frontend (o que o usuário vê e com o que interage), utilizamos um conjunto de tecnologias: o HTML para a estrutura textual, o CSS para a parte estética e, por fim, o JavaScript para trazer interatividade, além de realizar a integração com o código do backend. Dessa forma, construímos um sistema web que oferece serviços de freelance, proporcionando maior acessibilidade ao nosso público-alvo.



5. Desenvolvimento

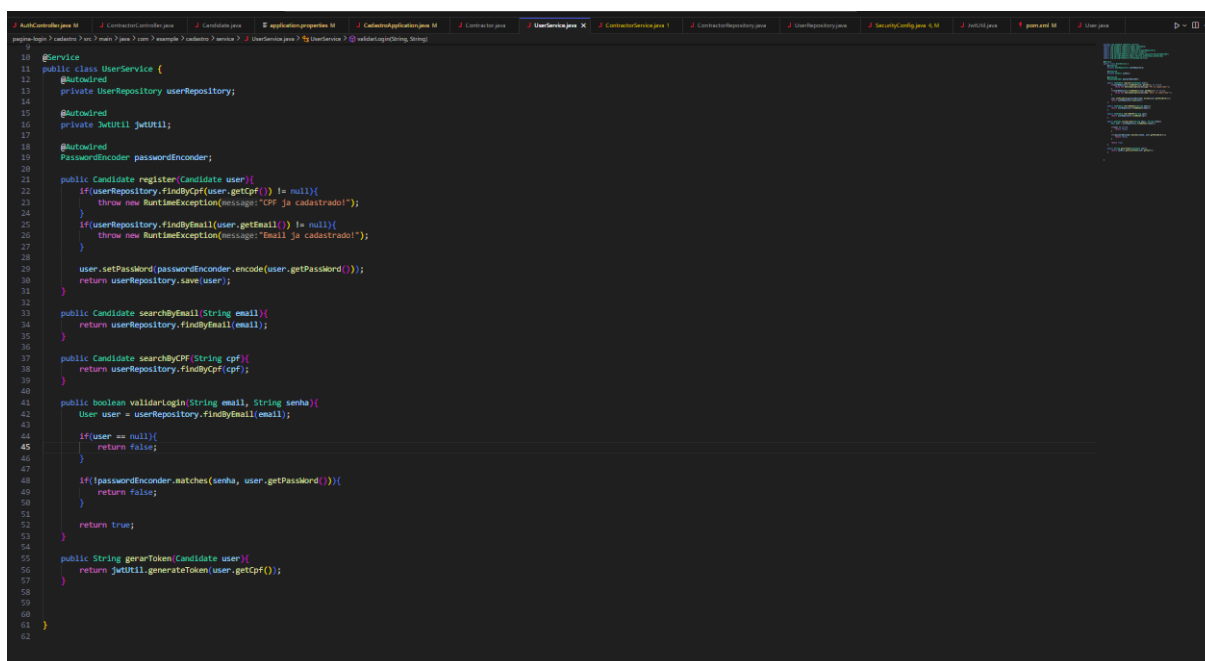
O projeto foi desenvolvido em Java utilizando o framework Spring Boot, que é bastante usado no meio corporativo por simplificar a criação de sistemas. Esse framework já traz configurações automáticas, ferramentas de segurança, conexão com banco de dados e até um servidor embutido, o que permite que a aplicação funcione de forma independente e sem complicações extras. O coração da aplicação está na classe principal, chamada `CadastroApplication`, que funciona como o botão de Iniciar do sistema. Quando essa classe é executada, o Spring Boot inicia todo o ciclo de vida da aplicação, carregando os componentes necessários, como entidades, serviços e repositórios, além de disponibilizar um servidor pronto para receber requisições.



5.1. ContractorService

As regras de funcionamento do sistema ficam concentradas nas chamadas “camadas de serviço”. Nelas temos, por exemplo, a UserService, responsável pela lógica de negócios relacionada aos candidatos, e a ContractorService, que faz o mesmo trabalho, mas para os contratantes. Essas classes cuidam de operações como registro, login e validação de dados, além de garantir a segurança das informações através da criptografia de senhas e da geração de tokens de autenticação. Assim, elas funcionam como o cérebro do sistema, organizando e controlando a forma como os usuários interagem com a aplicação.

Figura 1 - ContractorService



```
10 @Service
11 public class UserService {
12     @Autowired
13     private UserRepository userRepository;
14
15     @Autowired
16     private JwtUtil jwtUtil;
17
18     @Autowired
19     PasswordEncoder passwordEncoder;
20
21     public Candidate register(Candidate user){
22         if(userRepository.findByCpf(user.getCpf()) != null){
23             throw new RuntimeException(message:"CPF ja cadastrado!");
24         }
25         if(userRepository.findByEmail(user.getEmail()) != null){
26             throw new RuntimeException(message:"Email ja cadastrado!");
27         }
28         user.setPassword(passwordEncoder.encode(user.getPassword()));
29         return userRepository.save(user);
30     }
31
32     public Candidate searchByEmail(String email){
33         return userRepository.findByEmail(email);
34     }
35
36     public Candidate searchByCPF(String cpf){
37         return userRepository.findByCpf(cpf);
38     }
39
40     public boolean validateLogin(String email, String senha){
41         User user = userRepository.findByEmail(email);
42
43         if(user == null){
44             return false;
45         }
46
47         if(!passwordEncoder.matches(senha, user.getPassword())){
48             return false;
49         }
50
51         return true;
52     }
53
54     public String gerarToken(Candidate user){
55         return jwtUtil.generateToken(user.getCpf());
56     }
57 }
58
59
60
61
62 }
```

5.2. SecurityConfig

Outro ponto central é a segurança, configurada pela classe SecurityConfig. É nela que se define quais partes do sistema podem ser acessadas livremente e quais exigem login. Rotas como as de cadastro e login ficam abertas para qualquer pessoa, enquanto todas as demais só podem ser acessadas após autenticação. Além disso, essa configuração assegura que as senhas nunca sejam guardadas em texto simples no banco de dados, aumentando a proteção contra invasões e vazamentos de informação.

Figura 2 - SecurityConfig

```
pagina-login > cadastro > src > main > java > com > example > cadastro > security > SecurityConfig.java > ...
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
7 import org.springframework.security.crypto.password.PasswordEncoder;
8 import org.springframework.security.web.SecurityFilterChain;
9
10 @Configuration
11 public class SecurityConfig {
12     @Bean
13     public PasswordEncoder passwordEncoder(){
14         return new BCryptPasswordEncoder();
15     }
16
17     @Bean
18     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
19         http
20             .csrf().disable()
21             .authorizeHttpRequests()
22                 .requestMatchers(...patterns:"/usuarios/register", "/usuarios/login").permitAll()
23                 .anyRequest().authenticated()
24             .and()
25             .httpBasic(); // só para testes, depois você troca por JWT
26         return http.build();
27     }
28 }
29
30
31
```

5.3. ContractorRepository

O acesso aos dados é feito pelos repositórios, que são interfaces que conversam diretamente com o banco de dados. O UserRepository é responsável pelos candidatos e o ContractorRepository pelos contratantes. Graças ao Spring Data JPA, esses repositórios já vêm com métodos prontos para operações comuns, como salvar, atualizar, buscar ou deletar informações, além de permitir a criação de consultas personalizadas, como procurar usuários pelo e-mail, CPF ou CNPJ.

Figura 3 - ContractorRepository

```
pagina-login > cadastro > src > main > java > com > example > cadastro > repository > ContractorRepository.java > *O ContractorRepository
1 package com.example.cadastro.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.example.cadastro.model.Contractor;
7
8 @Repository
9 public interface ContractorRepository extends JpaRepository<Contractor, Long>{
10     Contractor findByEmail(String email);
11     Contractor findByCnpj(String cnpj);
12 }
13
```

5.4. Candidate

A forma como os dados são organizados está definida nos modelos da aplicação. A classe User funciona como uma base, reunindo atributos comuns, como nome, e-mail e senha, e evitando repetição de código. A partir dela, nascem classes mais específicas: Contractor, que adiciona informações de contratantes, como CNPJ, empresa e telefone, e Candidate, que acrescenta CPF e data de nascimento. Esses modelos também possuem regras de validação para garantir que os dados sejam corretos antes mesmo de serem salvos no banco, como verificar se o CPF tem 11 dígitos ou se a data de nascimento é realmente no passado.

Figura 4 - Candidate

```
pagina-login > cadastro > src > main > java > com > example > cadastro > model > Candidate.java > Candidate
1 package com.example.cadastro.model;
2 import java.time.LocalDate;
3
4 import jakarta.persistence.Entity;
5 import jakarta.validation.constraints.NotBlank;
6 import jakarta.validation.constraints.Past;
7 import jakarta.validation.constraints.Size;
8
9 @Entity
10 public class Candidate extends User {
11     @NotBlank(message = "CPF é obrigatório")
12     @Size(min = 11, max = 11, message = "CPF deve ter 11 caracteres")
13     private String cpf;
14
15     @Past(message = "Data de nascimento deve ser no passado")
16     private LocalDate dataNascimento;
17
18     public String getCpf() {
19         return cpf;
20     }
21
22     public void setCpf(String cpf) {
23         this.cpf = cpf;
24     }
25
26     public LocalDate getDataNascimento() {
27         return dataNascimento;
28     }
29
30     public void setDataNascimento(LocalDate dataNascimento) {
31         this.dataNascimento = dataNascimento;
32     }
33
34 }
```



5.5. ContractorController e AuthController

Por fim, existem os controladores, que são a porta de entrada da aplicação. O ContractorController e o AuthController recebem as requisições dos usuários, como pedidos de cadastro ou login, e encaminham essas informações para as classes de serviço. Eles também cuidam do retorno das respostas, incluindo mensagens de erro ou a geração de tokens de autenticação quando o login é bem-sucedido. Em alguns casos, ainda permitem que o usuário recupere seus próprios dados de forma segura, usando o token como chave de acesso.

Figura 5 - ContractorController

```

15
16 @RestController
17 @RequestMapping("/contratantes")
18 @CrossOrigin(origins="*")
19 public class ContractorController {
20
21     @Autowired
22     private ContractorService contractorService;
23
24     @Autowired
25     private JwtUtil jwtUtil;
26
27     @PostMapping("/register")
28     public ResponseEntity<> cadastrar(@RequestBody Contractor user){
29         try{
30             Contractor novo = contractorService.register(user);
31             return ResponseEntity.ok(novo);
32         } catch (RuntimeException e){
33             return ResponseEntity.badRequest().body(e.getMessage());
34         }
35     }
36
37     @PostMapping("/login")
38     public ResponseEntity<> login(@RequestParam String email, @RequestParam String senha){
39         boolean valido = contractorService.validarlogin(email, senha);
40
41         if(valido){
42             Contractor user = contractorService.searchByEmail(email);
43             String token = jwtUtil.generateToken(user.getEmail());
44             return ResponseEntity.ok(token);
45         }
46         return ResponseEntity.status(status:401).body(body:"Email/Cnpj ou senha incorreta");
47     }
48 }
49
50

```

Figura 6 - AuthController

```

19 @RestController
20 @RequestMapping("/usuarios")
21 @CrossOrigin(origins="*")
22 public class AuthController {
23
24     @Autowired
25     private UserService userService;
26
27     @Autowired
28     private JwtUtil jwtUtil;
29
30     @PostMapping("/register")
31     public ResponseEntity<> cadastrar(@RequestBody Candidate usuario){
32         try{
33             Candidate novo = userService.register(usuario);
34             return ResponseEntity.ok(novo);
35         } catch (RuntimeException e){
36             return ResponseEntity.badRequest().body(e.getMessage());
37         }
38     }
39
40     @PostMapping("/login")
41     public ResponseEntity<> login(@RequestParam String email, @RequestParam String senha){
42         boolean valido = userService.validarlogin(email, senha);
43
44         if (valido) {
45             Candidate user = userService.searchByEmail(email);
46             String token = jwtUtil.generateToken(user.getEmail());
47             return ResponseEntity.ok(token);
48         }
49         return ResponseEntity.status(status:401).body(body:"Email ou senha inválidos");
50     }
51
52     @PostMapping("/cpf")
53     public ResponseEntity<> searchByCpf(@PathVariable String cpf){
54         Candidate user = userService.searchByCPF(cpf);
55         if(user != null){
56             return ResponseEntity.ok(user);
57         }
58         return ResponseEntity.notFound().build();
59     }
60
61     @PostMapping("/me")
62     public ResponseEntity<> getUserData(@RequestHeader("Authorization") String token){
63         if(token.startsWith("Bearer ")){
64             token = token.substring(7);
65         }
66         if(jwtUtil.validateToken(token)){
67             return ResponseEntity.status(status:401).body(body:"Token inválido ou expirado");
68         }
69         String cpf = jwtUtil.extractCpf(token);
70         User user = userService.searchByCPF(cpf);
71         return ResponseEntity.ok(user);
72     }
73 }
74

```

5.6. Desenvolvimento final

O sistema foi construído seguindo uma arquitetura em camadas bem definida: a classe principal inicia tudo, os serviços aplicam as regras de negócio, os repositórios cuidam do banco de dados, os modelos estruturam as informações e os controladores fazem a ligação entre o usuário e o restante do sistema. Essa organização torna a aplicação mais clara, segura e fácil de manter, além de garantir que cada parte tenha uma responsabilidade.



6. Conclusão

O desenvolvimento do sistema web "Jheicafree" atingiu com sucesso o objetivo de criar uma plataforma robusta e funcional para a intermediação de serviços temporários. Ao longo do projeto, foi construída uma solução que atende diretamente à crescente demanda por um ambiente digital seguro e centralizado, capaz de conectar contratantes e prestadores de serviços de forma eficiente.

A metodologia adotada, que combinou o uso de Java com o framework Spring Boot para o backend e tecnologias consolidadas como HTML, CSS e JavaScript para o frontend, provou-se uma escolha acertada. Essa abordagem permitiu a criação de uma arquitetura em camadas bem definida, onde cada componente possui uma responsabilidade clara, desde os controladores que gerenciam as requisições até os repositórios que fazem a interface com o banco de dados. O resultado é um sistema organizado, seguro e de fácil manutenção.

O projeto não apenas entrega as funcionalidades essenciais propostas, como o cadastro de usuários e a publicação de vagas, mas também estabelece uma base sólida para futuras expansões. A implementação de recursos de segurança, como a criptografia de senhas e a autenticação via tokens, garante a proteção dos dados e a confiabilidade da plataforma.

Dessa forma, este trabalho representa uma contribuição significativa para o mercado de trabalho autônomo, oferecendo uma ferramenta que otimiza o processo de contratação, amplia a visibilidade de profissionais e fomenta a economia colaborativa. O sistema "Jheicafree" está, portanto, preparado para se tornar um ponto de encontro valioso entre a oferta e a demanda por serviços freelance.



7. Referências bibliográficas

GLOBADEV. *Curso de JavaScript para iniciantes*. [recurso eletrônico]. YouTube, 2021. Disponível em: <https://youtu.be/nPEpaft1y1k?si=oowiVYgnNvms9Fzw>.

FILIFE DESCHAMPS. *Curso de HTML e CSS: estrutura básica e boas práticas*. [recurso eletrônico]. YouTube, 2020. Disponível em: <https://youtu.be/McKNP3g6VBA?si=SQ9zMRZ8l1ycwmsh>.

PROGRAMADOR BR. *Curso de JavaScript: conceitos e fundamentos essenciais*. [recurso eletrônico]. YouTube, 2019. Disponível em: <https://youtu.be/w1J6gY40yMo?si=Qo27XGQyKaF5RftL>.

ORACLE. *Java Documentation*. Oracle, 2025. Disponível em: <https://docs.oracle.com/en/java/>

DIO – Digital Innovation One. *Bootcamps e Cursos de Java*. Disponível em: <https://www.dio.me/>. Acesso em: 17 set. 2025.

DEITEL, H. M.; DEITEL, P. J. *Java: como programar*. 10. ed. São Paulo: Pearson, 2016.

