

**FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS**  
**FATEC PROFESSOR JESSEN VIDAL**

**LEANDRO LOPES BUENO**

**ORIENTADOR: FABRICIO GALANDE MARQUES DE CARVALHO**

**IMAGES FOR TESTING**  
**SISTEMA WEB PARA GERAÇÃO DE CASOS DE TESTE DE**  
**IMAGENS DIGITAIS**

São José dos Campos

2020

## **SUMÁRIO**

### **1 Introdução**

#### **1.1 Definição do problema**

#### **1.2 Objetivo**

### **2 Desenvolvimento**

#### **2.1 Arquitetura**

#### **2.2 Modelo de Dados**

#### **2.3 Detalhes de desenvolvimento**

##### **2.3.1 Conexão com o banco de dados**

##### **2.3.2 Integração de Model e View**

##### **2.3.3 Manipulação das imagens**

##### **2.3.4 Download dos resultados**

##### **2.3.5 Interface do sistema**

### **3 Resultados e Discussão**

#### **3.1 Exemplo de geração de imagem para teste**

#### **3.2 Considerações finais**

# **1 INTRODUÇÃO**

Muitos sistemas utilizados atualmente são baseados em algoritmos de processamento de imagens. Muitos destes estão no dia-a-dia das pessoas, como, por exemplo, softwares de reconhecimento facial, presentes em diversos smartphones ou alguns aplicativos com funcionalidades capazes de reconhecer texto em fotos, muito útil para documentos.

Testar sistemas dessa natureza requer, com frequência, a geração de imagens cujas alterações (ex. rotação, deslocamento, etc...) são conhecidas de modo que a eficácia do sistema desenvolvido possa ser avaliada.

Dessa forma, é interessante gerar um sistema capaz de gerar dados de imagem para teste.

## **1.1 Definição do Problema**

Conforme foi mencionado a geração de imagens para testes é essencial para o processo de V&V (verificação e validação) de sistemas que trabalham com processamento de imagens. Entretanto, a geração de tais imagens para teste não é trivial e requer, muitas vezes, o conhecimento não só de programação, mas também de toda a matemática subjacente.

O problema tratado nesse trabalho é justamente o de geração automatizada de imagens para testes.

## **1.2 Objetivo**

Nesse trabalho serão desenvolvidos e avaliados:

- Lógica para geração de imagens;
- Backend em conjunto com uma interface web para abstrair o uso para o usuário;
- Geração de arquivos respectivos no caso de interesse por parte do usuário para download.

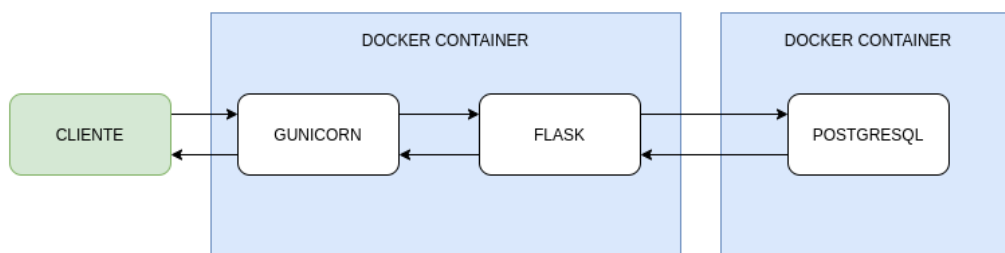
## 2 DESENVOLVIMENTO

Este capítulo apresenta detalhes sobre o desenvolvimento do sistema web para geração de casos de teste para imagens digitais.

### 2.1 Arquitetura

A arquitetura geral do sistema pode ser observada na imagem a seguir, sendo que:

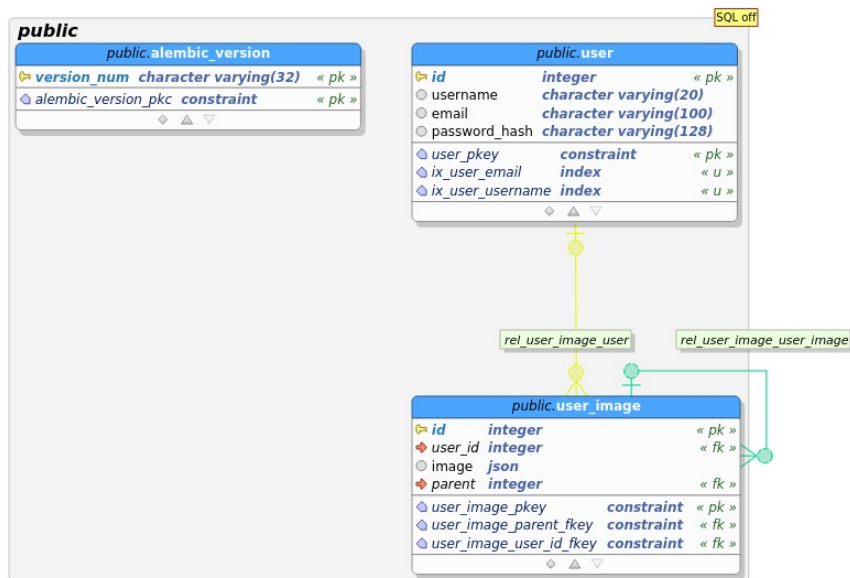
- **Cliente** - usuário que acessa a aplicação.
- **Gunicorn (Green Unicorn)** - Servidor de HTTP WSGI para Python usado para controlar o tráfego de requisições para a aplicação, gerando uma fila de processos conforme a demanda.
- **Flask** - Microframework Python para aplicações WEB.
- **PostgreSQL** - base de dados relacional.



### 2.2 Modelo de dados

A próxima figura apresenta o modelo de dados do sistema, uma representação ilustrativa de como as coisas conversam dentro do banco de dados PostgreSQL, citado na arquitetura, contendo as seguintes tabelas e seus campos:

- **alembic\_version** - Tabela usada no controle das migrações do banco de dados, que são a representação das modificações feitas, mantendo uma espécie de histórico.
  - version\_num - guarda a numeração da migração aplicada.
- **user** - Armazena os dados do usuário do sistema.
  - password\_hash - senha do usuário de forma encriptada.
- **user\_image** - Armazena a relação de cada usuário com as imagens e a relação das imagens entre si.
  - user\_id - chave estrangeira para o usuário dono da imagem.
  - image - json contendo informações da imagem salvas para futura recuperação da mesma.
  - parent - chave estrangeira com outro registro de "user\_image", representa a imagem original na qual esse registro foi gerado.



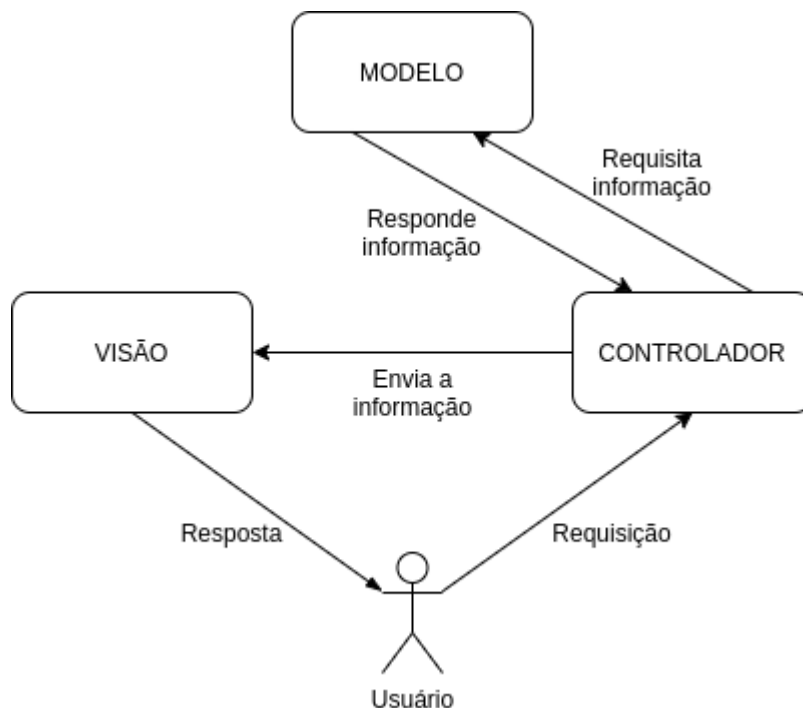
## 2.3 Detalhes do desenvolvimento

O sistema web foi desenvolvido utilizando o Microframework Flask junto ao ORM SQLAlchemy que abstrai a comunicação entre a aplicação e o Banco de Dados, seguindo uma arquitetura MVC(Modelo, Visão, Controlador), onde:

**Modelo:** Cria uma abstração do banco de dados, fazendo com que cada tabela do banco tenha uma classe equivalente no código.

**Visão:** Templates que nesse caso são estruturados em arquivos HTML com aprimoramentos feitos por CSS, que auxilia na estilização, e de JavaScript que ajuda na dinamicidade de alguns comportamentos nesse caso .

**Controlador:** Métodos que controlam as rotas e as interações entre o Modelo e a Visão.



Quanto a escolha do banco de dados PostgreSQL ela se deve pela facilidade de integração com o Flask e o SQL Alchemy, esse último abstraindo bastante a comunicação.

### 2.3.1 Conexão com o banco de dados

Como foi dito, toda a comunicação e estruturação do banco de dados acontece por meio da biblioteca SQLAlchemy, um ORM Python, com ele é possível representar toda a estrutura de dados por meio de classes, como é possível observar no modelo de User, por exemplo:

```
10 class User(UserMixin, db.Model):
11     id = db.Column(db.Integer, primary_key=True)
12     username = db.Column(db.String(20), index=True, unique=True)
13     email = db.Column(db.String(100), index=True, unique=True)
14     password_hash = db.Column(db.String(128))
15     images = db.relationship('UserImage', backref='user', lazy=True)
16
17     def __repr__(self):
18         return '<User {}>'.format(self.username)
19
20     def set_password(self, password):
21         self.password_hash = generate_password_hash(password)
22
23     def check_password(self, password):
24         return check_password_hash(self.password_hash, password)
```

Outra vantagem é a criação de métodos capazes de guardar regras importantes, como o `set_password`, presente na linha 20, ele é responsável por receber a senha do usuário e salvar no Modelo de forma encriptada através da função `generate_pass_hash`, garantindo a segurança desse dado, por exemplo.

### 2.3.2 Integração de Model e View

As vantagens de se utilizar um ORM ficam mais claras ao observar sua utilização dentro do Controlador, como no exemplo a seguir para o caso de registro de usuário:

```
55 @app.route('/registro', methods=['GET', 'POST'])
56 def registro():
57     if current_user.is_authenticated:
58         return redirect(url_for('home'))
59     form = RegistrationForm()
60     if form.validate_on_submit():
61         user = User(username=form.username.data, email=form.email.data)
62         user.set_password(form.password.data)
63         db.session.add(user)
64         db.session.commit()
65         flash('Parabéns, usuário cadastrado!')
66         return redirect(url_for('login'))
67     return render_template('registro.html', title='registro', form=form)
```

Nesse exemplo já podemos destacar uma ótima vantagem do Microframework Flask, ele cuida de toda a parte de resolução de URLs, bastando apenas utilizar, nesse caso, um decorator com o path e os métodos HTTP permitidos.

Seguindo no código primeiro é feita a verificação do usuário já não estar autenticado (linha 55), caso não e caso o formulário recebido seja válido (linha 60), o ORM entra em prática, é gerada uma instância de User (linha 61), atribuída uma senha com o método declarado no Modelo (linha 62) e salvo no banco (linhas 63 e 64), tudo feito rápido e facilmente com código Python.

A última parte do fluxo do Controlador é geralmente usada para retornar algum elemento visual ao usuário como um template HTML ou uma resposta HTTP que será tratada pelo navegador.

### 2.3.3 Manipulação das imagens

Já na parte central do sistema, a manipulação das imagens, entra o OpenCV (Open Source Computer Vision), uma biblioteca escrita em C/C++, mas com suporte a linguagem Python. O OpenCV é uma biblioteca muito bem estabelecida na área de visão computacional principalmente em relação ao Python e por ser Open Source, é constantemente atualizada pela comunidade altamente ativa.

Sendo assim, no contexto desse projeto, o OpenCV cuida de toda a parte de transformação nas imagens, enquanto o Flask cuida do Upload e Download dos resultados. Observe no código a seguir:

```

80     form = ImageForm()
81     if form.validate_on_submit():
82         file = form.image.data
83         user_image = UserImage(user=current_user, image=file)
84         db.session.add(user_image)
85         db.session.commit()
86         flash('Imagem base salva!')
87         return redirect(url_for('imagem'))

```

Essa parte do código cuida primeiramente do upload de uma imagem base que será utilizada para realizar as modificações. Pegamos a imagem enviada no formulário (linha 82), salvamos sua referência no banco vinculando ao usuário (linhas 83 a 85).

```

97     if form.validate_on_submit():
98         nw_image = img_transform(
99             imagem.image.url,
100             rotacao=form.data['rotacionar'],
101             zoom=form.data['zoom'],
102             force_gray=form.data['preto_branco'],
103             desfoque=form.data['desfoque'],
104         )
105         user_image = UserImage(user=current_user, image=nw_image, parent=imagem.id)
106         db.session.add(user_image)
107         db.session.commit()
108         flash('Nova imagem gerada com sucesso!')

```

Acima, o path de uma imagem e algumas informações obtidas do formulário são passadas para função `img_transform`, ela e retornará a imagem modificada (linhas 98 a 104) que é salva no modelo `UserImage`, vinculada ao usuário e a imagem base utilizada (linhas 105 a 107).

```

7 def img_transform(path_img, rotacao=0, zoom=1.0, force_gray=False, desfoque=False):
8     imagem = cv2.imread(path_img)
9     altura, largura = imagem.shape[:2]
10    ponto = (largura / 2, altura / 2)
11
12    # Rotaciona a imagem
13    img_rotacao = cv2.getRotationMatrix2D(ponto, rotacao, zoom)
14    result = cv2.warpAffine(imagem, img_rotacao, (largura, altura))
15
16    # Converte imagem para escala de cinza
17    if force_gray:
18        result = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
19
20    # Aplica um desfoque na imagem
21    if desfoque:
22        result = cv2.GaussianBlur(result, (35, 35), 0)
23
24    folder = tempfile.mkdtemp()
25    cv2.imwrite(os.path.join(folder, 'image.png'), result)
26
27    f = open(os.path.join(folder, 'image.png'), 'rb')
28    file = FileStorage(f, 'image.png', name='file', content_type='image/png')
29    return file

```

Nessa função primeiro carregamos a imagem com o path informado utilizando o método `imread` do OpenCV (linha 8), isso transformará a imagem em um Array do Numpy, uma biblioteca altamente otimizada para operações numéricas, outra informação que é gerada e utilizamos é a altura e largura, com ela calculamos o centro da imagem (linhas 9 e 10). Após, realizamos as modificações com base nos parâmetros informados, onde:

- É feita a rotação em graus da imagem baseada no ponto central calculado e com o zoom informado (linhas 13 e 14);



- Caso solicitado, é feita a conversão em tons de cinza (linhas 17 e 18);
- Também é feito o desfoque, caso solicitado (linhas 21 e 22).

Por último, para não perder o resultado, é criado um arquivo temporário, onde é salvo o resultado das modificações e como nosso modelo espera uma instância de FileStorage da biblioteca werkzeug, criamos e retornamos uma (linhas 27 e 28), que depois é salva de maneira efetiva como foi descrito anteriormente.

### 2.3.4 Download dos resultados

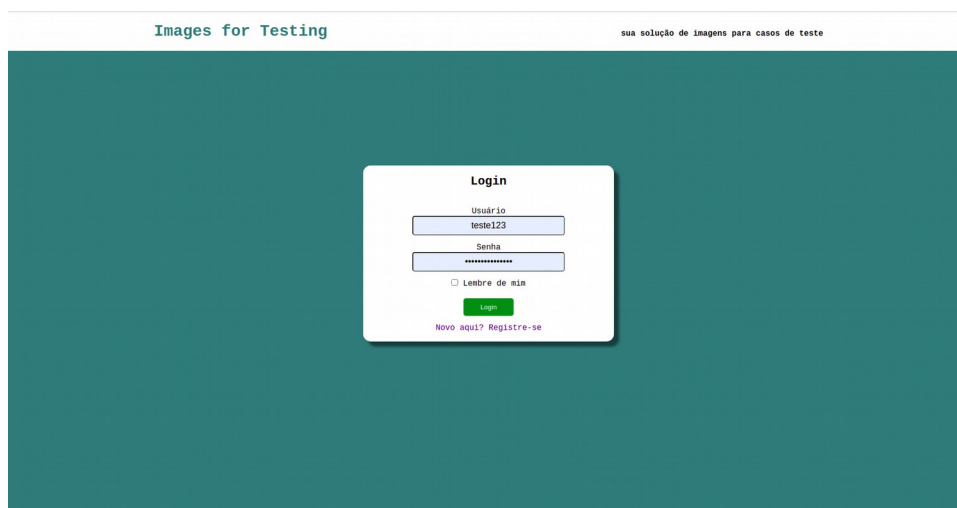
Para dar acesso aos arquivos gerados pelo usuário na sessão anterior, existe o endpoint a seguir:

```
16 @app.route('/imagem/download-zip/<int:pk>')
17 def download_zip(pk):
18     memory_file = BytesIO()
19     zf = zipfile.ZipFile(memory_file, 'w', zipfile.ZIP_STORED)
20     files = [f.image.url for f in UserImage.query.filter_by(parent=pk)]
21     for file in files:
22         zf.write(file, os.path.basename(file))
23     zf.close()
24     memory_file.seek(0)
25     return send_file(memory_file, mimetype='application/zip', as_attachment=True, attachment_filename='images.zip')
```

Nele primeiro é gerado um arquivo ZIP em memória com permissão de escrita (linhas 18 e 19), após é gerada a lista de imagens do usuário baseadas na chave primária da imagem principal (linha 20), em seguida ocorre uma iteração nessa lista e por último o arquivo é concluído, selecionado e enviado através da função `send_file` do Flask (linhas 23 a 25), encerrando toda a rotina principal pensada para o sistema.

### 2.3.5 Interface do sistema

Tela de Login

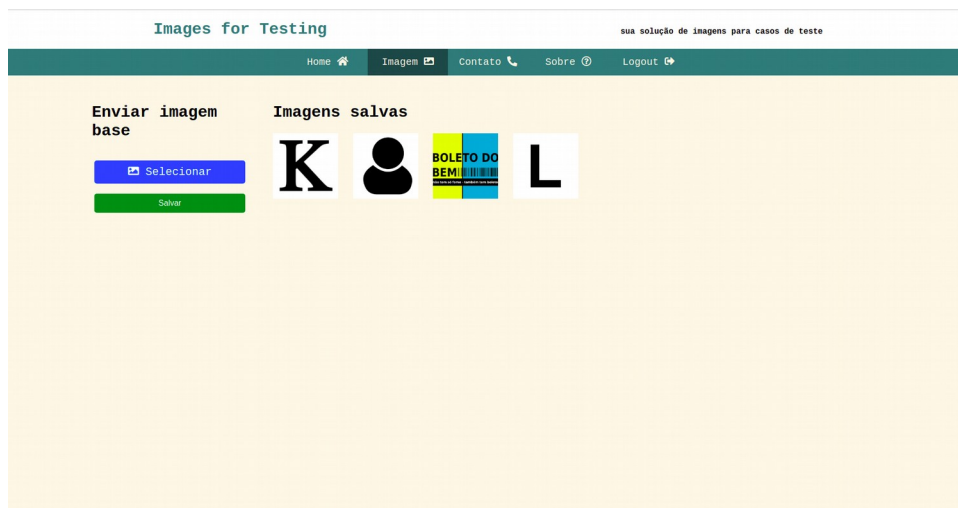


The screenshot shows a web application interface for 'Images for Testing'. At the top, there is a header with the text 'Images for Testing' on the left and 'sua solução de imagens para casos de teste' on the right. The main content area has a dark teal background. In the center, there is a white login form titled 'Login'. The form contains two input fields: 'Usuário' with the value 'teste123' and 'Senha' with masked characters '\*\*\*\*\*'. Below these fields is a checkbox labeled 'Lembre de mim' which is unchecked. At the bottom of the form is a green 'Login' button. Below the button, there is a link that says 'Novo aqui? Registre-se'.

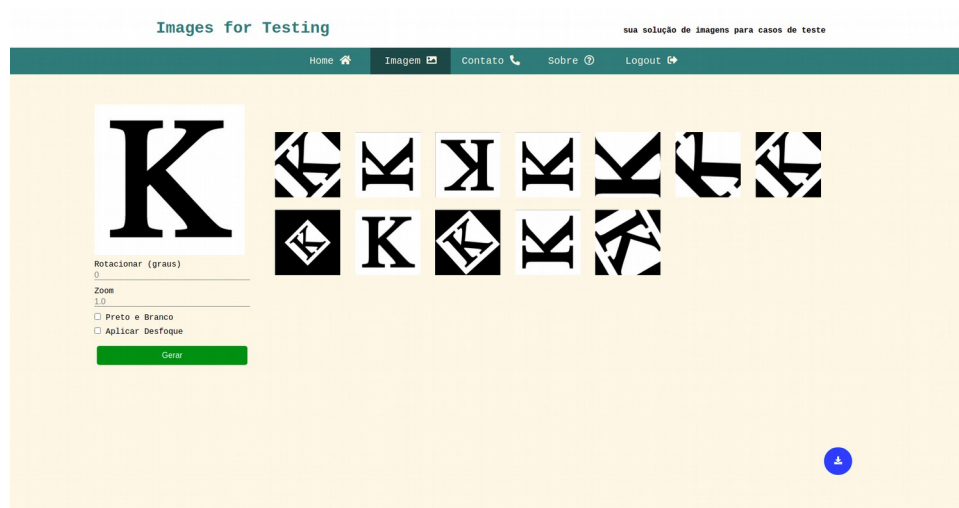
## Home com instruções de uso



## Tela para upload e seleção da imagem que será modificada



Tela com formulário para modificar imagem salva e botão para baixar os resultados

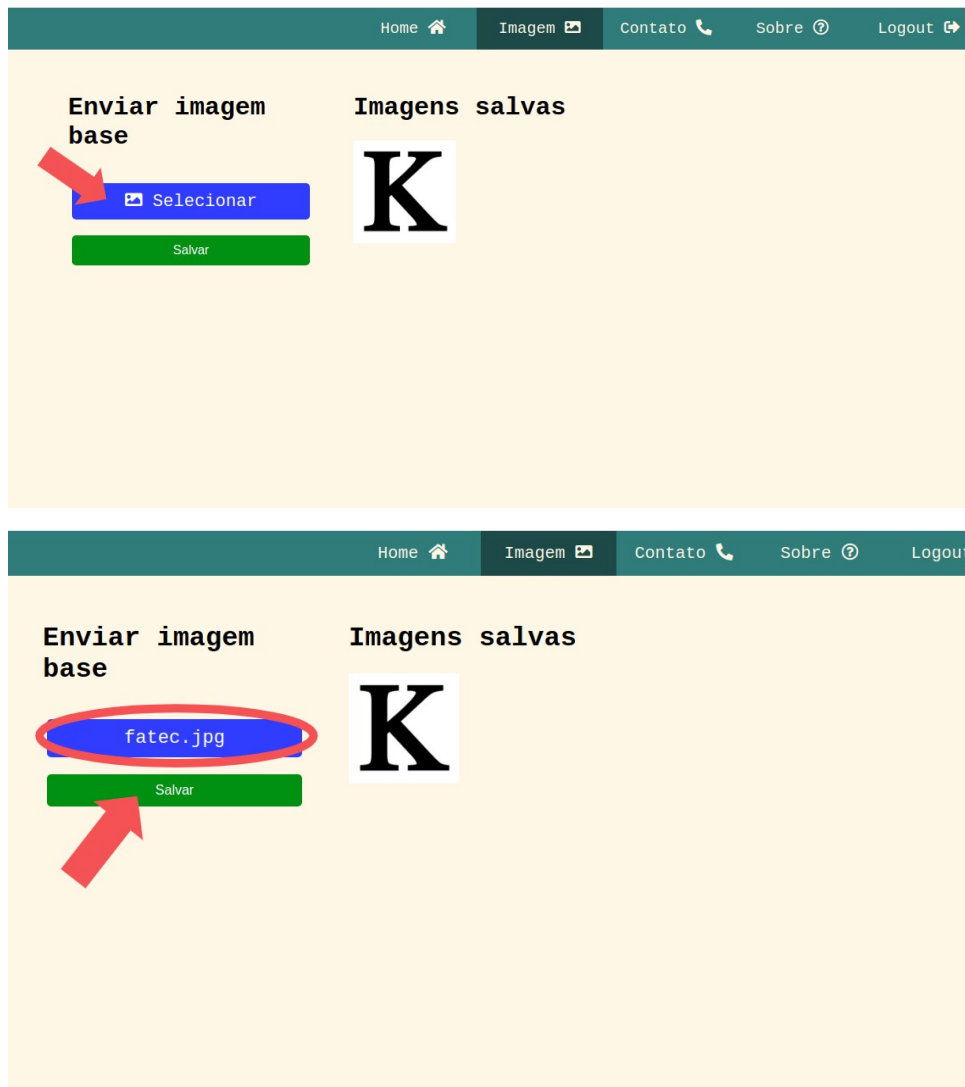


### 3 RESULTADOS E DISCUSSÃO

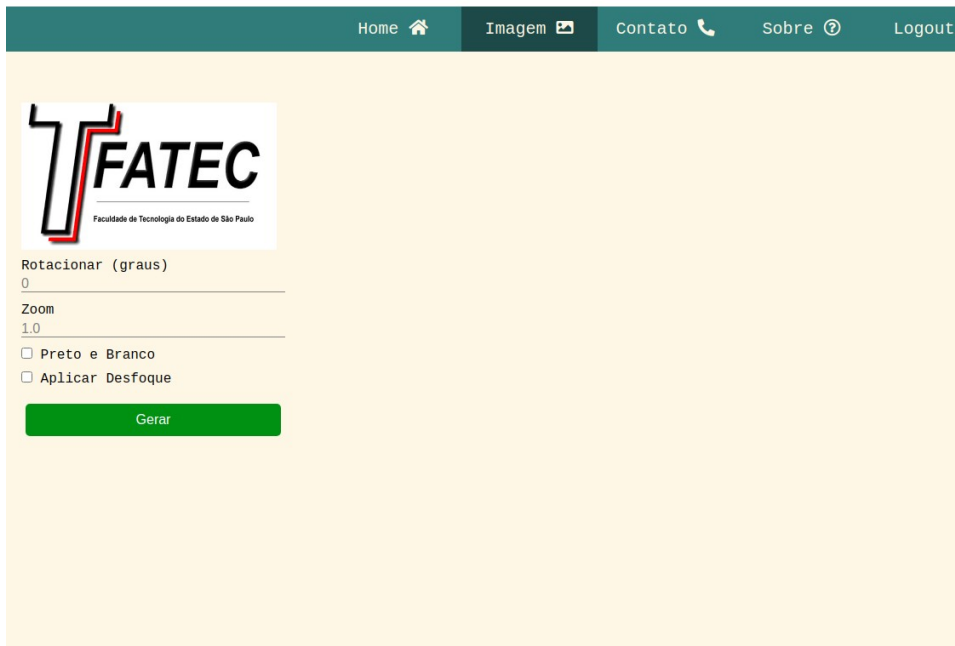
Este capítulo tem por objetivo apresentar os principais resultados e conclusões obtidas no decorrer de todo o processo de desenvolvimento.

#### 3.1 Exemplo de geração de imagem para teste

- I. Deve ser feito o upload da imagem que será utilizada para gerar os casos de teste, clicando em selecionar, na seção de "Imagem" do sistema, escolhendo um arquivo da sua máquina e depois clicando em salvar:



- II. Logo após, clicando sobre a imagem será exibido formulário para geração dos casos de teste a partir dela.



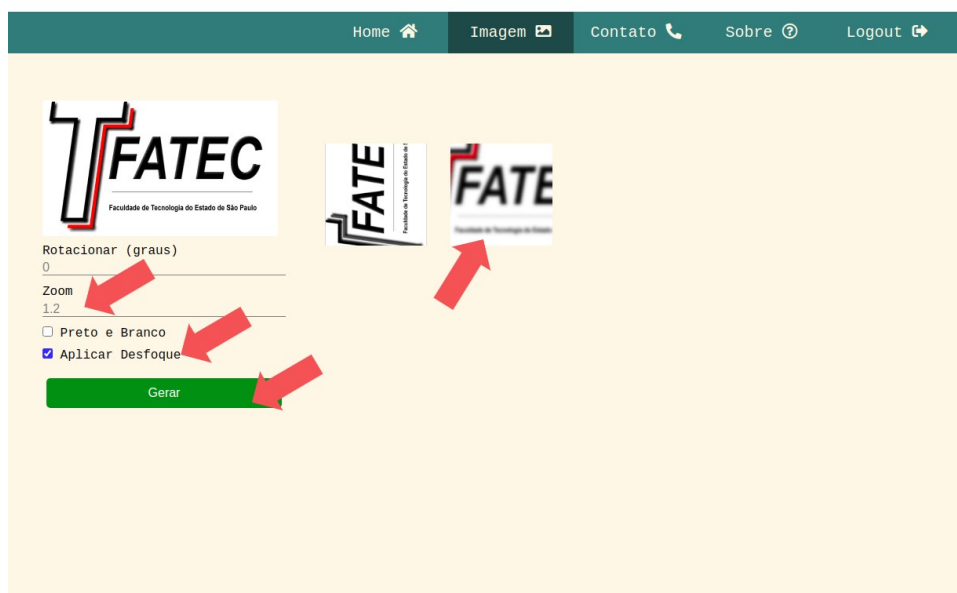
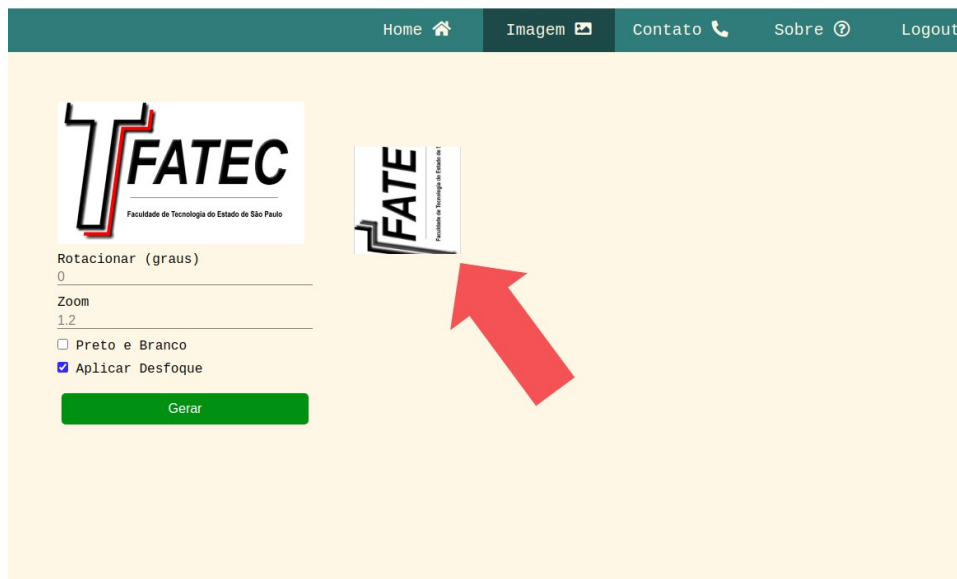
The screenshot shows a web interface with a dark teal header containing navigation links: Home, Imagem, Contato, Sobre, and Logout. The main content area has a light yellow background. At the top left, there is a logo for FATEC (Faculdade de Tecnologia do Estado de São Paulo). Below the logo, there is a form with the following elements: a text input for 'Rotacionar (graus)' with the value '0', a text input for 'Zoom' with the value '1.0', two checkboxes labeled 'Preto e Branco' and 'Aplicar Desfoque', and a green button labeled 'Gerar'.

- III. Existem várias possibilidades de transformação, para esse caso a imagem será rotacionada 90 graus e as cores mudadas para escala de cinza.

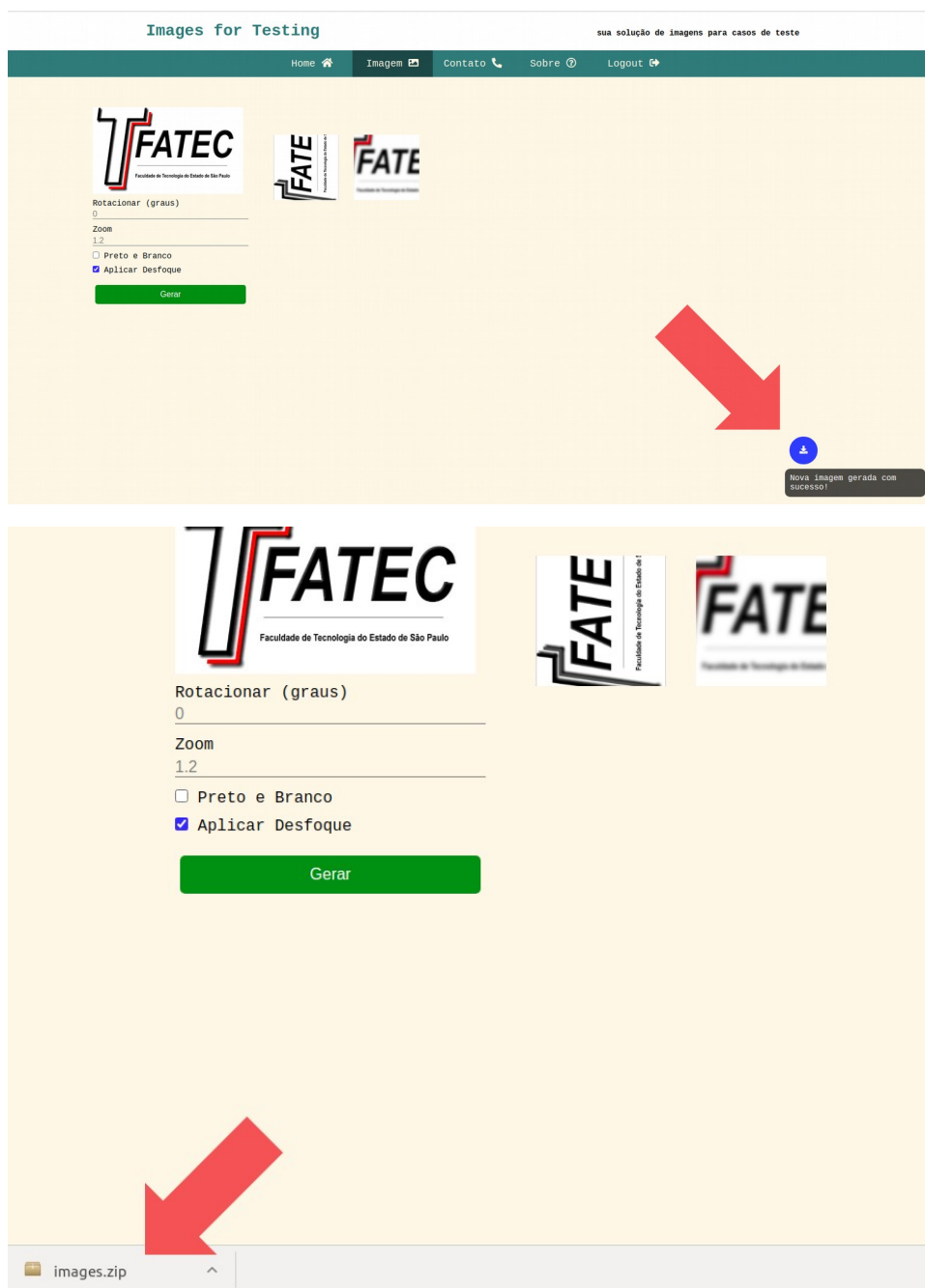


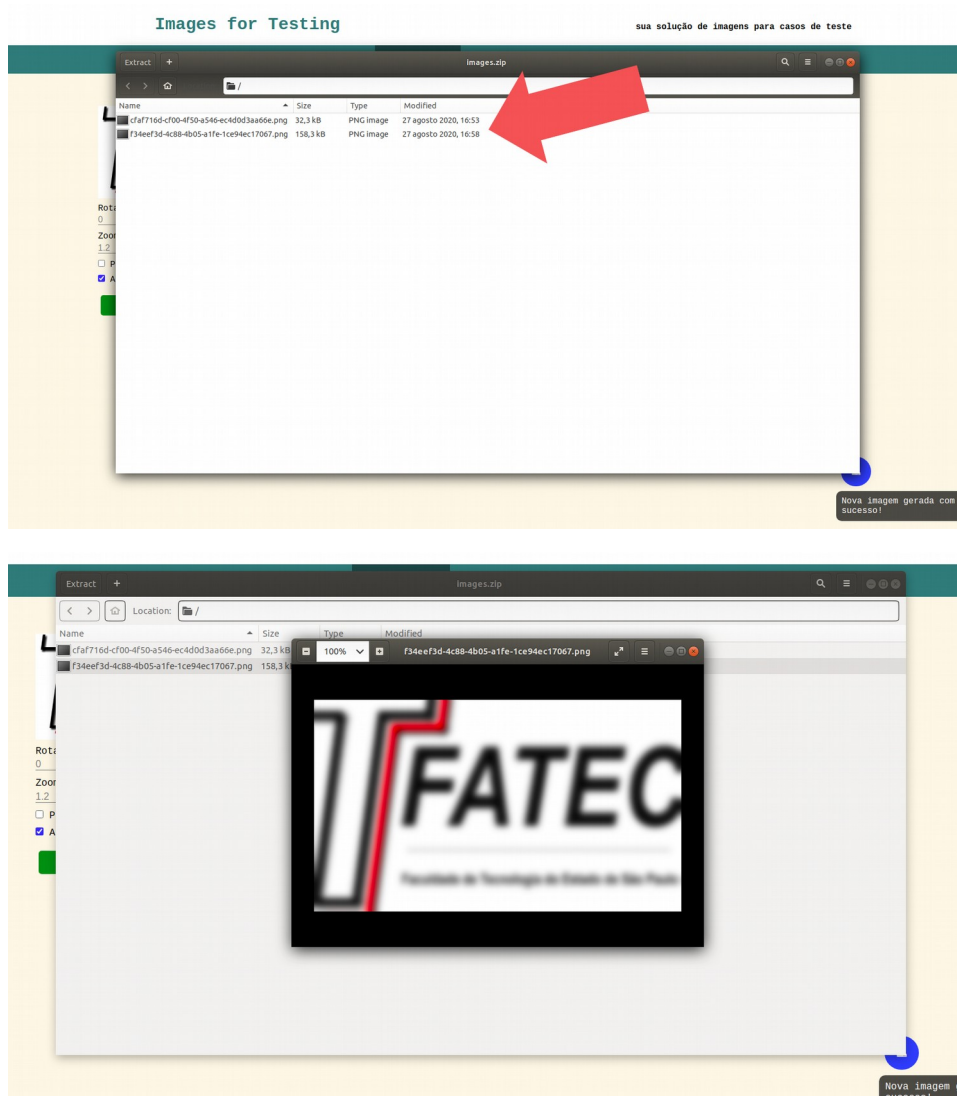
This screenshot shows the same FATEC logo generation interface as the previous one, but with different settings. The 'Rotacionar (graus)' input is now '90', the 'Zoom' input is still '1.0', the 'Preto e Branco' checkbox is checked, and the 'Aplicar Desfoque' checkbox is unchecked. Three red arrows point to the '90' value, the 'Preto e Branco' checkbox, and the 'Gerar' button. The green 'Gerar' button is still present at the bottom.

- IV. Depois de definidas as transformações basta gerar o resultado que automaticamente irá ser exibido ao lado do formulário, o processo pode ser repetido quantas vezes forem necessárias.



V. Por último é possível baixar tudo em formato zip.





### 3.2 Considerações finais

Das tecnologias desse trabalho, vale destacar a biblioteca OpenCV, com ela foi possível aplicar facilmente as modificações necessárias nas imagens. Outro ponto positivo é a quantidade de recursos disponíveis, abrindo um leque vasto de possibilidades para futuras funcionalidades para o sistema.

Uma grande dificuldade foi trabalhar com o armazenamento das imagens, foi necessário bastante tempo estruturando uma infraestrutura simples e eficaz, sendo essa parte um ponto fraco do Framework, mas que pôde ser resolvido com bastante pesquisa já que é uma questão comum em sistemas normalmente.

Esse trabalho se mostrou promissor na tentativa de agilizar a geração de casos de teste para sistemas de reconhecimento de imagem digital e por tanto cumpriu com a expectativa inicial.