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 Resultado visual

3 Resultados e Discussão

1 INTRODUÇÃO

Teste de Software é um processo que faz parte do desenvolvimento de software e tem como principal objetivo revelar falhas/bugs para que sejam corrigidas até que o produto final atinja a qualidade desejada. Basicamente são gerados algoritmos que pegam as funções do software e expõe a diversos tipos de entradas verificando se um resultado esperado é obtido.

1.1 Definição do Problema

Muitos sistemas utilizados em reconhecimento de imagens necessitam de casos de teste para avaliar sua robustez em relação a diversos fatores, tais como reamostragem, transformações afins, etc. A geração de imagens de teste pode se tornar uma tarefa fastidiosa caso não seja automatizada.

1.2 Objetivo

O objetivo deste trabalho consiste em um sistema para geração de casos de teste envolvendo diversas transformações associadas a imagens digitais, ou seja, a partir de uma imagem base, serão serão criadas diversas outras com algum tipo alteração.

Será desenvolvido tanto os módulos para execução de tais transformações, como um sistema web que os utilize e exemplifique o resultado da geração de casos de teste, visando deixar a criação desse tipo de teste uma tarefa menos custosa.

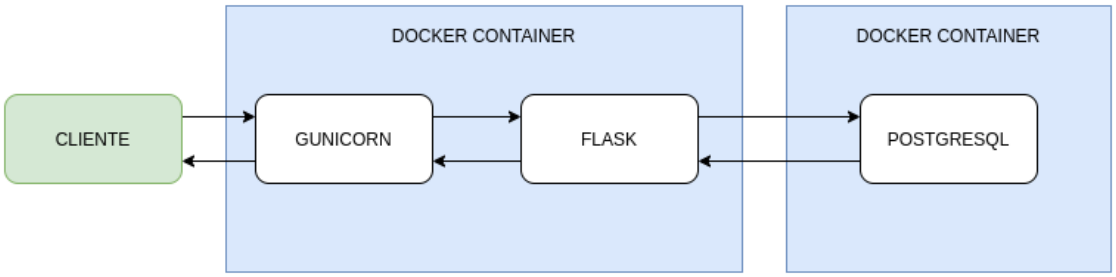
2 DESENVOLVIMENTO

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:

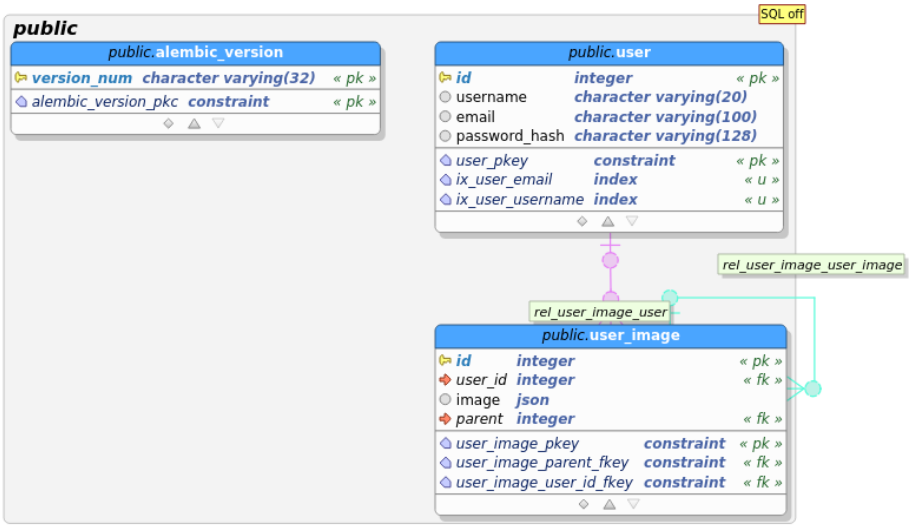
- **Cliente** - usuário que acessa a aplicação.
- **Gunicorn (Green Unicorn)** - Servidor de HTTP WSGI para Python usado para controlar o tráfego de requisições para a aplicação.
- **Flask** – Microframework Python para aplicações WEB.
- **PostgreSQL** – base de dados relacional.



2.2 Modelo de dados

A figura 2 apresenta o modelo de dados do sistema, apresentando as seguintes tabelas:

- **alembic_version** – Tabela usada no controle das migrações do banco de dados, que são a representação das modificações feitas, mantendo uma espécie de histórico.
- **user** – Armazena os dados do usuário do sistema.
- **user_image** – Armazena a relação de cada usuário com as imagens e a relação das imagens entre si.



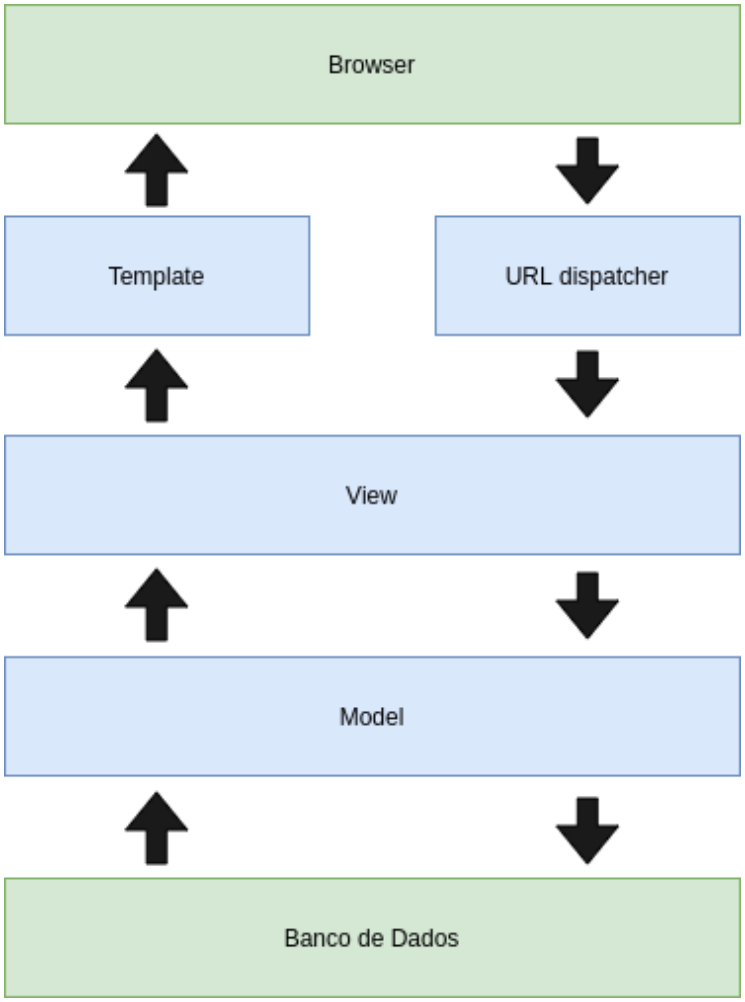
2.3 Detalhes do desenvolvimento

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 MVT (Model, View, Template), onde:

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

View: É a camada responsável por controlar as regras de negócio do sistema, nela é feita a comunicação com Model e Template.

Template: É a camada composta por HTML, CSS, Javascript e afins, sendo responsável por retornar a parte visual ao usuário.



O banco de dados escolhido foi PostgreSQL por ser um rebusto e de fácil integração com Flask e SQLAlchemy.

2.3.1 Conexão com o banco de dados

Toda a comunicação e estruturação do banco de dados acontece por meio da biblioteca SQLAchemy, 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:

```python
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), index=True, unique=True)
    email = db.Column(db.String(100), index=True, unique=True)
    password_hash = db.Column(db.String(128))
    images = db.relationship('UserImage', backref='user', lazy=True)

    def __repr__(self):
        return '<User {}>'.format(self.username)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

Dessa forma é possível integrar facilmente os dados com a View, onde estarão as regras de negócio que trabalharão com eles.

2.3.2 Integração de Model e View

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

```python
@app.route('/registro', methods=['GET', 'POST'])
def registro():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(username=form.username.data, email=form.email.data)
        user.set_password(form.password.data)
        db.session.add(user)
        db.session.commit()
        flash('Parabéns, usuário cadastrado!')
        return redirect(url_for('login'))
    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 autentificado, caso não e caso o formulário recebido seja válido, o ORM entra em prática, é gerada uma instância de User, atribuída uma senha e salvo no banco, tudo feito rápido e facilmente com código Python.

A última parte do fluxo de uma view é geralmente usada para retornar algum elemento visual ao usuário, nesse caso, um templete, que nada mais é que HTML e afins.

2.3.3 Manipulação das imagens

Para a 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 constantantemente 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:

```
if form.validate_on_submit():
    file = form.image.data
    user_image = UserImage(user=current_user, image=file)
    db.session.add(user_image)
    db.session.commit()
    flash('Imagem base salva!')
    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, salvamos sua referência no banco vinculando ao usuário.

```
if form.validate_on_submit():
    nw_image = img_transform(
        imagem.image.url,
        rotacao=form.data['rotacionar'],
        zoom=form.data['zoom'],
        force_gray=form.data['preto_branco'],
        desfoque=form.data['desfoque'],
    )
    user_image = UserImage(user=current_user, image=nw_image, parent=imagem.id)
    db.session.add(user_image)
    db.session.commit()
    flash('Nova imagem gerada com sucesso!')
```

Aqui o path de uma imagem e algumas informações obtidas do formulário são passadas para função *img_transform*, ela e retornará a imagem modificada que é salva no modelo UserImage, vinculada ao usuário e a imagem base utilizada.

```python
def img_transform(path_img, rotacao=0, zoom=1.0, force_gray=False, desfoque=False):
    imagem = cv2.imread(path_img)
    altura, largura = imagem.shape[:2]
    ponto = (largura / 2, altura / 2)

    # Rotaciona a imagem
    img_rotacao = cv2.getRotationMatrix2D(ponto, rotacao, zoom)
    result = cv2.warpAffine(imagem, img_rotacao, (largura, altura))

    # Converte imagem para escala de cinza
    if force_gray:
        result = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)

    # Aplica um desfoque na imagem
    if desfoque:
        result = cv2.GaussianBlur(result, (35, 35), 0)

    folder = tempfile.mkdtemp()
    cv2.imwrite(os.path.join(folder, 'image.png'), result)

    f = open(os.path.join(folder, 'image.png'), 'rb')
    file = FileStorage(f, 'image.png', name='file', content_type='image/png')
    return file
```

Nessa função primeiro carregamos a imagem com o path informado utilizando o método imread do OpenCV, 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. Após realizamos as modificações com base nos parâmetros informados. Por último, para não perder o resultado, é criado um arquivo temporário, salva a imagem e como nosso modelo espera uma instância de FileStorage da biblioteca werkzeug, criamos e retornamos uma, que depois é salva como foi descrevido anteriormente.

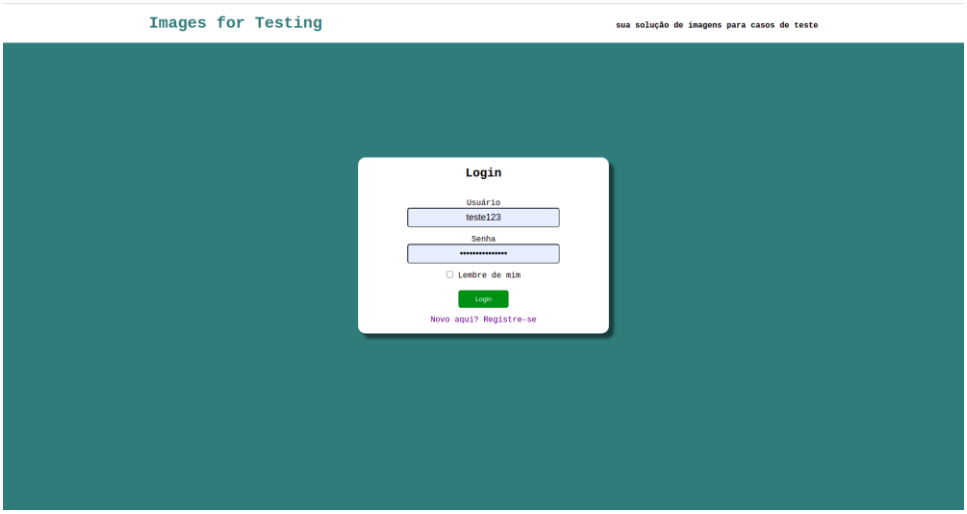
2.3.4 Download dos resultados

Para dar acesso aos arquivos gerados ao usuário, é gerado um arquivo zip em memória e depois enviado com a função send_file do Flask, o código responsável por isso pode ser observado a seguir:

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

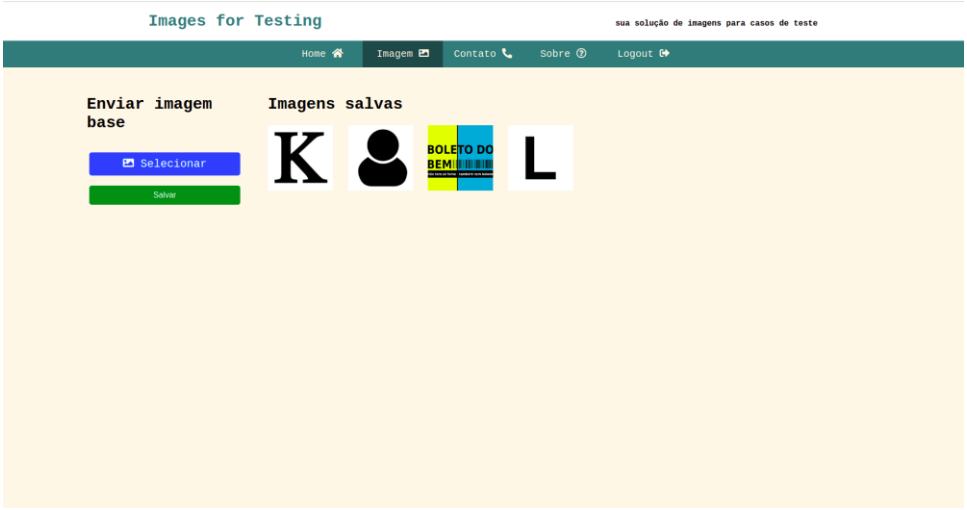
2.3.5 Resultado visual

Tela de Login

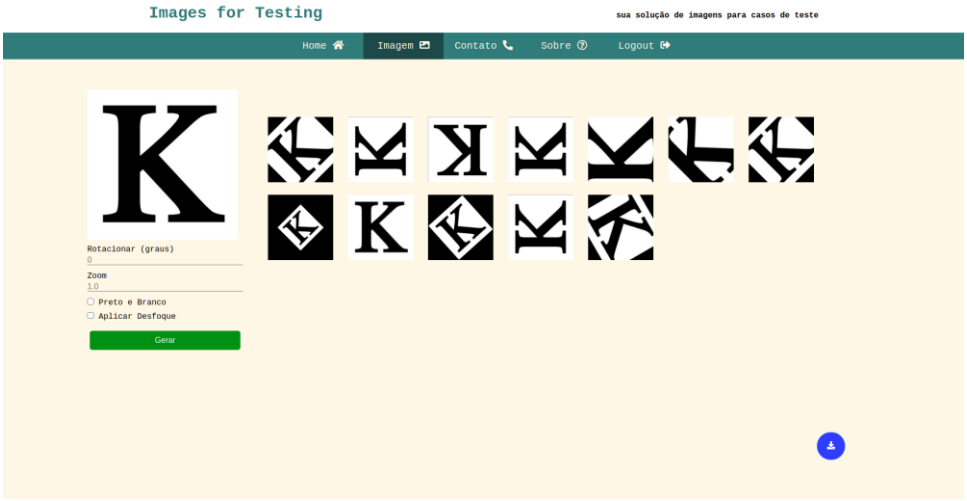


Home com instruções de uso

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

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 rescursos 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 tentaiva de agilizar a geração de casos de teste para sistemas de reconhecimento de imagem digital e por tanto cumpriu com a expectativa inicial.