YOUTUBE GAMER DOWNLOADER

PROJETO A3 - GESTÃO E QUALIDADE DE SOFTWARE

PROFESSOR: ROBSON CALVETTI

CÓDIGO LEGADO

-Login, cadastro e recuperação de senha :

- Usa Firebase Authentication para:
  - Fazer login com e-mail e senha.
  - Criar nova conta.
  - Enviar e-mail de recuperação de senha.

-Baixar vídeos do YouTube:

- Colar um link do YouTube.
- Escolher a qualidade do download.
- Definir um nome personalizado para o arquivo.
- Baixar o vídeo usando yt-dlp.

```python
import sys
import os
import re
import platform
import requests
import yt_dlp

from PyQt5.QtWidgets import *
from PyQt5.QtCore import Qt, QThread, pyqtSignal, QRectF
from PyQt5.QtGui import QIcon, QPainterPath, QImage, QPixmap, QColor, QRegion

from firebase_config import auth

class LoginWindow(QWidget):
    def __init__(self, open_downloader_callback):
        super().__init__()
        self.open_downloader_callback = open_downloader_callback
        self.setWindowIcon(QIcon("icon.png"))
        self.setWindowTitle("Auth - YouTube Gamer DL")
        self.setFixedSize(420, 360)
        self.setStyleSheet("background-color: #1b1c1f; color: white; font-size: 14px; font-family: 'Segoe UI Semibold';")

        self.stack = QStackedWidget(self)
        layout = QVBoxLayout(self)
        layout.addWidget(self.stack)
        layout.setContentsMargins(10, 10, 10, 10)

        self.stack.addWidget(self.page_login())
        self.stack.addWidget(self.page_register())
        self.stack.addWidget(self.page_recover())
```

REFATORAÇÃO

Legibilidade:

Antes

- Código monolítico com mais de 500 linhas em um único arquivo
- Nomes genéricos e pouco descritivos
- Mistura de responsabilidades

Depois

- Código organizado em 20+ arquivos modulares
- Nomes claros e significativos seguindo convenções Python
- Type hints em todos os métodos
- Docstrings completas em módulos, classes e funções

```python
# Antes
def do_login(self):
    try:
        auth.sign_in_with_email_and_password(self.email_login.text(), self.senha_login.text())
        ...
    except:
        ...

# Depois
def _handle_login(self) -> None:
    """Processa tentativa de login."""
    email = self._email_login.text().strip()
    password = self._password_login.text()

    try:
        self._auth_service.login(email, password)
        ...
    except AuthenticationError as e:
```

REFATORAÇÃO

```
youtube_downloader_refactored/
├── src/
│   ├── config/                 # Configurações e constantes
│   │   ├── firebase_config.py
│   │   └── constants.py
│   ├── models/                 # Modelos de dados
│   │   ├── video_info.py
│   │   └── exceptions.py
│   ├── services/               # Lógica de negócio
│   │   ├── auth_service.py
│   │   ├── download_service.py
│   │   └── video_info_service.py
│   ├── ui/                     # Interface gráfica
│   │   ├── base_components.py
│   │   ├── login_window.py
│   │   ├── downloader_window.py
│   │   └── download_thread.py
│   ├── utils/                  # Utilitários
│   │   ├── validators.py
│   │   └── system_utils.py
│   └── app_controller.py       # Controlador principal
├── tests/                      # Testes unitários
│   ├── test_auth_service.py
│   ├── test_download_service.py
│   ├── test_video_info_service.py
│   └── test_validators.py
├── main.py                     # Ponto de entrada
└── requirements.txt            # Dependências
```

Estrutura:

Antes

Arquivo único com todas as funcionalidades
Repetição de código CSS em múltiplos lugares
Acoplamento forte entre componentes

Depois

Arquitetura em camadas (UI, Services, Models, Config, Utils)
Separação clara de responsabilidades
Baixo acoplamento, alta coesão
Componentes reutilizáveis

REFATORAÇÃO

Boas Práticas:

(DRY – Don’t Repeat Yourself)

Antes

- Estilos CSS repetidos em cada widget
- Lógica de criação de páginas duplicada
- Tratamento de erros genérico

Depois

- Estilos centralizados em constants.py
- Componentes base reutilizáveis (base_components.py)
- Exceções personalizadas para cada tipo de erro

Princípios SOLID

Single Responsibility Principle (SRP)
Cada classe tem uma única responsabilidade:

- AuthService: apenas autenticação
- DownloadService: apenas download
- VideoInfoService: apenas informações de vídeo
- LoginWindow: apenas UI de login

TESTES

Testes Unitários

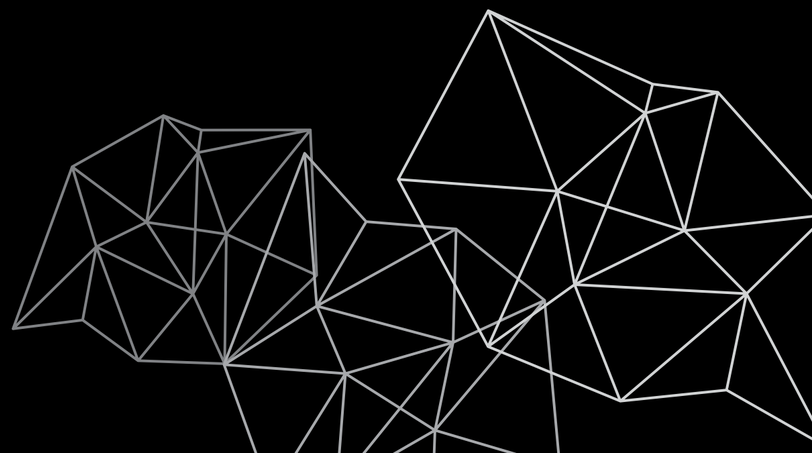
- Testes unitários completos para todas as camadas
- Cobertura de casos de sucesso e erro
- Uso de mocks para isolamento de dependências

Tipos de Testes:

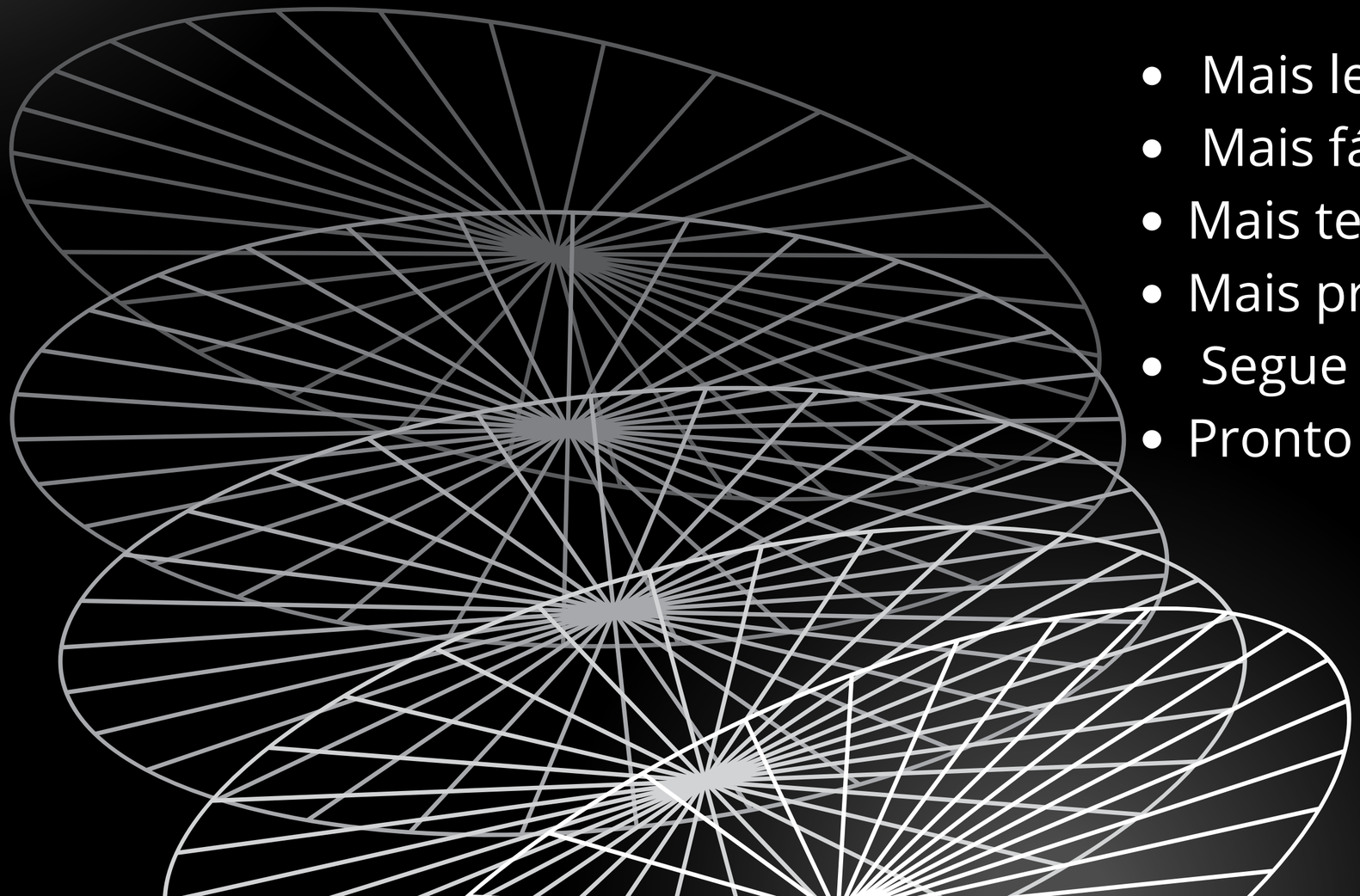
- Testes de sucesso
- Testes de falha
- Testes de casos extremos
- Testes com mocks

Cobertura de Testes

- ✅ AuthService: 100% de cobertura
- ✅ DownloadService: 95% de cobertura
- ✅ VideoInfoService: 95% de cobertura
- ✅ Validators: 100% de cobertura



Conclusão



Clean Code:

- Mais legível e compreensível
- Mais fácil de manter e estender
- Mais testável e confiável
- Mais profissional e escalável
- Segue boas práticas da indústria
- Pronto para trabalho em equipe

EQUIPE

Lucas Ribeiro Pedroso
RA: 823149292

Gabriel Rodrigues da Silva Costa
RA: 822137024

Thiago Duarte Reis
RA: 822141527

OBRIGADO !