Aqui está um roteiro simples e direto para sua apresentação do **Trabalho Prático 2: Desenvolvimento Seguro de Tela de Login e Registro**. Ele está dividido em partes para você organizar a fala e demonstrar o sistema com segurança, clareza e objetividade:

🎯 1. Introdução (1-2 minutos)

- **Apresente-se** e os membros do grupo (se houver).
- Explique o objetivo do trabalho:

"Nosso objetivo foi desenvolver uma tela de login e registro aplicando boas práticas de segurança da informação. Utilizamos a linguagem Python com o framework Flask, por ser leve, didático e eficiente para aplicações web seguras."

🔧 2. Tecnologias Utilizadas (1 minuto)

- **Python 3 + Flask** (estrutura do backend).
- **SQLite** (banco de dados local e simples).
- **Werkzeug** (para hash seguro de senhas).
- **email-validator** (validação de e-mail).
- **flask-talisman** (proteção de headers HTTP).
- **HTML/CSS** para front-end.

"Optamos por essas tecnologias por serem compatíveis com o foco educacional do projeto e por permitirem implementar rapidamente os princípios básicos de segurança."

🔒 3. Implementação da Segurança (3-4 minutos)

a) Registro

- Validação de e-mail com email-validator.
- Verificação de senha e confirmação.
- Armazenamento da senha com generate_password_hash().

b) Login

- Comparação segura com check_password_hash().
- Mensagens genéricas para evitar vazamento de informações (“Usuário ou senha incorretos”).

c) Proteções extras

- Headers HTTP seguros com flask-talisman.
- Prevenção contra SQL Injection usando SQLite com parâmetros seguros (?).

- Arquivo requirements.txt para dependências rastreáveis.

"Nos preocupamos em evitar vulnerabilidades comuns como SQL Injection, vazamento de informações sensíveis e exposição de senhas."

💻 **4. Demonstração (3-4 minutos)**

- **Mostrar o formulário de registro.**
    - Inserir e-mail válido e senhas diferentes (mostrar erro).
    - Corrigir e cadastrar corretamente.
- **Mostrar login.**
    - Tentar com senha errada (mostrar mensagem genérica).
    - Logar corretamente e exibir a tela de boas-vindas.
- **Abrir o banco de dados** (opcional, para mostrar hash da senha).

📄 **5. Relatório Técnico (1 minuto)**

"No relatório técnico, explicamos:

- As tecnologias utilizadas;
- As boas práticas aplicadas;
- As principais dificuldades (por exemplo, integração de bibliotecas e tratamento de sessões);
- E como resolvemos cada uma."

✅ **6. Conclusão (30 segundos)**

- Reforçar o aprendizado adquirido:

"Esse trabalho foi essencial para compreendermos que segurança não é um recurso opcional, mas sim uma base para qualquer sistema web."

- Agradecer:

"Agradecemos a oportunidade e estamos abertos para perguntas."

🛡️ **Relatório Técnico – Trabalho Prático 2**

Disciplina: Segurança da Informação
Tema: Desenvolvimento Seguro de Tela de Login e Registro
Data de Entrega: 24/04/2025

🧰 **Tecnologia Utilizada**

- **Linguagem:** Python 3
- **Framework:** Flask
- **Banco de Dados:** SQLite
- **Outras Bibliotecas:**
    - email-validator
    - Werkzeug.security
    - Flask-Talisman

🔐 **Implementações de Segurança**

📌 **1. Registro de Usuário**

✅ **Validação de E-mail**

📂 **Arquivo:** app.py

```
from email_validator import validate_email, EmailNotValidError

# Dentro da rota /register
try:
    validate_email(email)
except EmailNotValidError:
    flash("E-mail inválido.")
    return redirect(url_for("register"))
```

✅ **Verificação de Senha e Confirmação**

📂 **Arquivo:** app.py

```
if password != confirm:
    flash("As senhas não coincidem.")
    return redirect(url_for("register"))
```

✅ **Armazenamento Seguro da Senha**

📁 **Arquivo:** app.py

```
from werkzeug.security import generate_password_hash

hashed_password = generate_password_hash(password)
cursor.execute("INSERT INTO users (name, email, password) VALUES (?, ?, ?)",
               (name, email, hashed_password))
```

📌 **2. Login de Usuário**

✅ **Comparação de Senha com Hash**

📁 **Arquivo:** app.py

```
from werkzeug.security import check_password_hash

if user and check_password_hash(user[3], password):
    session["user_id"] = user[0]
    return redirect(url_for("dashboard"))
```

✅ **Mensagens Genéricas de Erro**

📁 **Arquivo:** app.py

```
else:
    flash("Usuário ou senha incorretos.")
    return redirect(url_for("login"))
```

📌 **3. Proteções Extras**

✅ **Headers HTTP Seguros com flask-talisman**

📁 **Arquivo:** app.py

```
from flask_talisman import Talisman

app = Flask(__name__)
Talisman(app)
```

✅ **Prevenção contra SQL Injection**

📁 **Arquivo:** app.py

```
# Uso de placeholders seguros
cursor.execute("SELECT * FROM users WHERE email = ?", (email,))
```

✅ **Dependências Rastreáveis**

📂 **Arquivo:** requirements.txt

```
Flask
email-validator
Flask-Talisman
```

🩹 **Dificuldades Enfrentadas**

- Erro KeyError: 'confirm' resolvido ajustando o name="confirm" no formulário HTML.
- Problemas de importação da biblioteca email-validator resolvidos com pip install email-validator.
- Página inicial retornava 404; solucionado criando index.html.

📎 **Conclusão**

O sistema implementa todas as boas práticas exigidas pelo enunciado, com técnicas modernas e bibliotecas seguras. Mesmo sendo minimalista, oferece uma base robusta para autenticação segura, ideal para aplicações reais.
