Documentação

Decidimos estruturar o projeto utilizando algo semelhante a arquitetura MVC. No arquivo *app.py* ficam as rotas da API onde os usuários irão acessar e interagir com o sistema, seguindo a arquitetura MVC é o ***Controller***.

O diretório *templates* representa a ***View*** da arquitetura, nela se encontram as páginas que os usuários irão acessar e interagir.

O arquivo *db.py* representa a ***Model***, nele estão contidas as regras de negócio e manipulação dos arquivos **.txt** utilizados como banco de dados do sistema.

```
56    @app.route("/", methods=['GET', 'POST'])
57    def index():
58      if (auth_guard()):
59        return app.redirect(app.url_for('aluno_menu'))
60
61      if (request.method == 'POST'):
62        remove_session()
63        cpf = request.form.get('cpf')
64        if service.logar_aluno(cpf):
65          session['cpf_aluno'] = request.form['cpf']
66          session['role'] = 'ALUNO'
67
68          if (not periodo_guard()):
69            return app.redirect(app.url_for('periodo_fechado'))
70
71          res = check_inscription(cpf)
72          if (res):
73            return app.redirect(app.url_for('inscricao_realizada', turma=res))
74
75          return app.redirect(app.url_for('aluno_menu'))
76        flash("CPF inválido ou não cadastrado", "error")
77
78      return render_template('aluno/login.html')
79
80    @app.route("/admin", methods=['GET', 'POST'])
81    def admin():
82      if (not admin_guard() and auth_guard()):
83        return app.redirect(app.url_for('index'))
84
85      if (request.method == 'POST'):
86        remove_session()
87        login = request.form.get('login')
88        senha = request.form.get('password')
89        if service.logar_admin(login, senha):
90          session['role'] = 'ADMIN'
91          return app.redirect(app.url_for('admin_menu'))
92        else:
```

Figura 1 - Trecho de código do app.py

As páginas em HTML estão contidas na pasta *templates*. Com o Flask definimos ela como local onde se encontram. Para estilizá-las colocamos os arquivos css na pasta

static como sugere a documentação do Flask.

Figura 2 - Trecho de código do db.py

Dentro do db.py estamos fazendo uso de 3 classes:

1. **Service**: classe responsável por acessar os arquivos, ela contém métodos para ler e escrever nos arquivos. Além disso, decidimos colocar nela os métodos de autenticação do sistema.

2. **Turma**: classe responsável por armazenar informações das turmas como quantidade de vagas, alunos totais, quais alunos estão presentes nas turmas. Ela faz uma abstração do banco de dados, ou seja, cada turma é instanciada para facilitar a leitura das informações. Com ela, não é necessário abrir arquivos a todo momento para verificar os alunos e a quantidade total de alunos.

3. **Período**: classe responsável por toda lógica e regras de negócio do período e turmas. Ela controla o período de matrícula, ou seja, abre ou fecha o período. Dentro dela contém classes de turmas. Além disso, realiza operações nas turmas como verificar os estudantes das turmas e a quantidade respectiva de alunos.

Além disso, dentro do db.py que ocorre o tratamento de concorrência do sistema como discutido na sprint 2. Aqui fazemos uso do lock para garantir que cada matrícula seja feita corretamente evitando a condição de corrida.

Árvore de pastas e arquivos do sistema

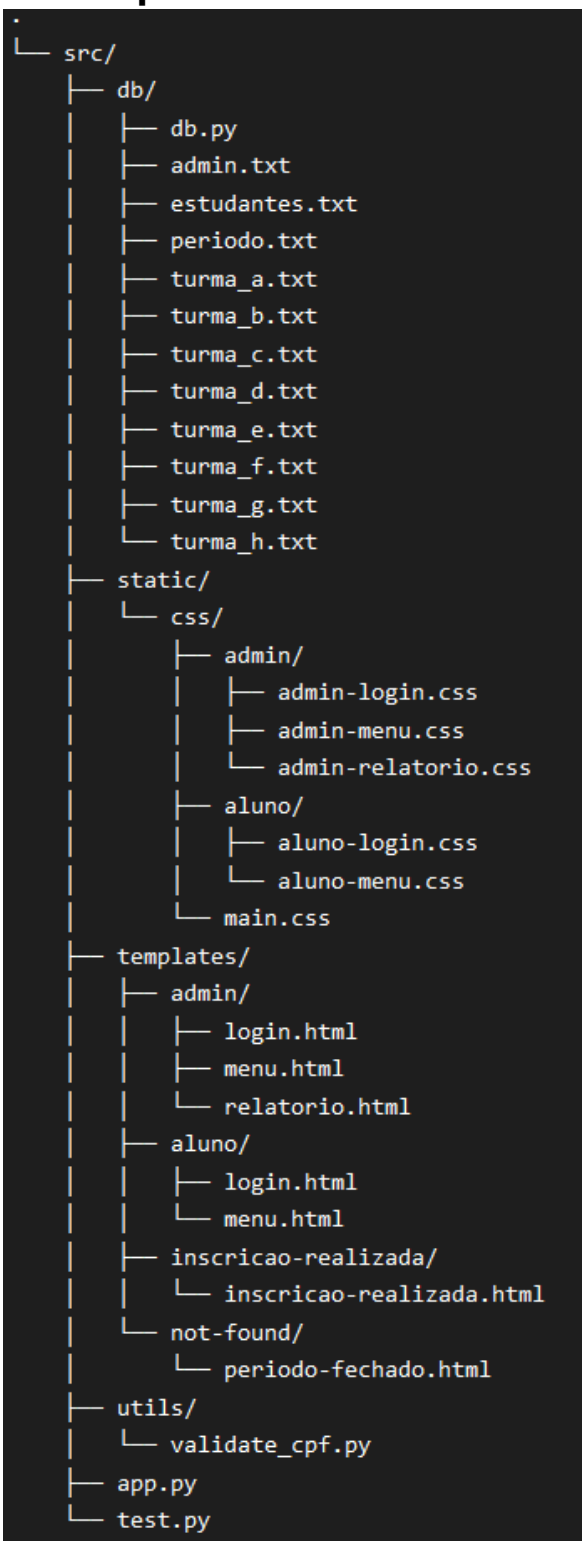

Figura 3 - Árvore de diretórios

Como pode-se observar, usamos nomes claros para pastas e arquivos. Além disso, a organização facilita encontrar o html da página, css utilizado ou lógica atribuída ao banco de dados, pois definimos uma pasta para conter os arquivos txt e classe responsável por gerenciar leitura e escrita nos arquivos.

Concorrência

Para facilitar a leitura e não ser necessário abrir arquivos da turma a cada requisição, faremos uso de atributos internos das classes **Turma** e **Período**.

Para requisições de escrita, utilizaremos locks para garantir o correto acesso aos atributos das classes e escrita nos arquivos garantindo amplo acesso aos recursos compartilhados.

Escrita:
- Matrícula nas turmas
- Inscrição de alunos
- Abertura e fechamento do período

Leitura
- Ao iniciar o sistema é feito uma leitura dos arquivos e para armazenar nos atributos das classes
- Login e senha do admin
- CPFs dos alunos
- Informações da turma (nº alunos, qtd vagas)
- Período

Foi realizado alguns testes sobre a concorrência utilizando threads para simular acessos às rotas HTTP.

- Uso de barreira para que todas as threads executem simultaneamente o caso de teste para verificar o funcionamento. Testes estarão disponíveis no arquivo teste.py do projeto.

Testes

Realizamos diversos testes manuais durante o desenvolvimento e término do mesmo para validar o correto funcionamento. O mais complicado é validar a concorrência do sistema. Para isso foi criado uma bateria de testes que pode ser encontrada no arquivo test.py.

Para verificar os testes utilize o comando `python3 src/test.py` estando presente no diretório do projeto. Este teste faz uso de threads para simular usuários que farão uso do sistema. Utilizamos a classe de Service juntamente com as threads para simular a interação.

```python
def teste_matricular_alunos():
    NUM_THREADS = 8
    barreira = threading.Barrier(NUM_THREADS)

    time.sleep(1)
    service.sobrescrever('turma_a.txt', '')

    def inscrever_aluno(cpf, turma):
        barreira.wait()
        periodo.turmas[turma].inscrever_aluno(cpf, service)

    # TESTE DE CONCORRÊNCIA NA MATRÍCULA DE ALUNOS
    maximo_alunos_turma = periodo.turmas['A'].max

    args_list = [(f"t{i}", "A") for i in range(1, 9)]
    create_threads(inscrever_aluno, args_list)

    print("TESTANDO CONCORRÊNCIA NA MATRÍCULA DE ALUNOS\n")
    alunos_turma = periodo.turmas['A'].alunos
    time.sleep(2)
    casos = [
        "Quantidade máxima de alunos permitidos é maior ou igual a quantidade de alunos na classe da turma A",
        "Quantidade máxima de alunos permitidos é maior ou igual a quantidade de alunos no arquivo turma_a.txt",
        "Alunos presentes na classe da turma A são os mesmos que estão no arquivo turma_a.txt"
    ]
    resultados = []

    # CASO 0
    if (maximo_alunos_turma >= len(alunos_turma)):
        resultados.append("OK")
    else:
        resultados.append("ERRO")
```

Figura 4 - Trecho de código dos testes

Links úteis

Link do repositório
Documentação do Flask