Sistema de Agendamento de Salas

Relatório Técnico de Aplicação Web desenvolvida com Django

Disciplina: Programação para Web II
Autor: Glauber Giordano De Morais Lima
Instituição: Instituto Federal Do Piauí
Ano: 2026

Sumário

1 Introdução

Este documento apresenta um **relatório de atividade** sobre o desenvolvimento do projeto **Sistema de Agendamento de Salas**, realizado na disciplina de **Programação para Web II**. O foco do texto è descrever **o que foi implementado**, como a aplicação foi organizada no Django e como foi executada/testada durante o desenvolvimento.

1.1 Objetivo da atividade

A atividade teve como objetivo construir uma aplicação funcional para **registrar, listar e remover reservas de salas**, permitindo que usuários consultem a ocupação e criem novos agendamentos de forma simples.

1.2 Escopo implementado

No escopo efetivamente implementado, o sistema contempla:

- tela de **listagem** das reservas;
- formulário para **criação** de reservas;
- ação de **remoção** de reservas;
- validações básicas (campos obrigatórios e consistência de data/horário);
- persistência em banco de dados local.

1.3 Resumo das etapas realizadas

As principais etapas executadas durante o desenvolvimento foram:

1. criação do projeto e de um *app* para o módulo de reservas;
2. modelagem das entidades e aplicação de migrações;
3. implementação das rotas e das funcionalidades de listagem, criação e exclusão;
4. criação dos templates HTML para interface;
5. execução do sistema localmente e teste de acesso na rede.

2 Tecnologias Utilizadas

Esta seção registra as tecnologias utilizadas no desenvolvimento e **como** elas foram aplicadas no projeto.

2.1 Python

Foi a linguagem utilizada para implementar toda a lógica do sistema no back-end, incluindo rotas, validações e integração com o banco de dados.

2.2 Django

Foi o framework utilizado para estruturar o projeto, criar o *app* de reservas, definir os modelos no banco, implementar as telas (rotas + templates) e executar o servidor durante os testes.

2.3 HTML e CSS

Foram utilizados nos templates para construir as páginas do sistema. O CSS foi aplicado para deixar a interface mais organizada (título, tabela/lista, botões e formulários).

2.4 Banco de dados SQLite

Foi utilizado como banco de dados local durante o desenvolvimento, permitindo persistir as reservas sem necessidade de configuração adicional.

2.5 Justificativa (contexto da atividade)

As escolhas priorizaram um fluxo de desenvolvimento simples e compatível com o escopo da disciplina: Django + SQLite para criar rapidamente o CRUD de reservas e HTML/CSS para construir uma interface funcional.

3 Arquitetura da Aplicação

Nesta atividade, a organização do código foi mantida de forma simples, separando os arquivos principais do *app* de reservas para facilitar manutenção e localização de funcionalidades.

3.1 Estrutura criada no projeto

O projeto foi organizado com um *app* dedicado ao agendamento (por exemplo, `reservas` ou similar). Os arquivos mais utilizados durante o desenvolvimento foram:

- `models.py`: definição das tabelas utilizadas no sistema (por exemplo, Sala e Reserva);
- `views.py`: implementação das telas/ações (listar, criar e remover);

- `urls.py`: rotas do *app* e integração com as rotas principais do projeto;
- `templates/`: páginas HTML do sistema (listagem e formulário);
- `migrations/`: migrações geradas/aplicadas para criar as tabelas no banco.

3.2 Organização por responsabilidade (o que ficou em cada arquivo)

Durante a implementação, foi adotada a seguinte organização prática:

- regras de dados e campos ficaram concentradas nos **models**;
- regras de fluxo (salvar, excluir, redirecionar e tratar erros simples) ficaram nas **views**;
- a estrutura de tela (tabela, botões e formulários) ficou nos **templates**.

Essa separação foi suficiente para o escopo da atividade e ajudou a manter o projeto organizado.

4 Funcionamento do Servidor

4.1 Como o sistema foi executado durante o desenvolvimento

O sistema foi executado e verificado utilizando o servidor de desenvolvimento do Django, iniciado com:

```
python manage.py runserver
```

Durante a atividade, esse modo de execução foi suficiente para testar as funcionalidades implementadas (listar, criar e remover reservas), verificando o resultado diretamente no navegador.

4.2 Testes de acesso via rede local (`0.0.0.0`)

Para testar o acesso em outro dispositivo na mesma rede, o servidor foi iniciado escutando em todas as interfaces:

```
python manage.py runserver 0.0.0.0:8000
```

Em seguida, foi utilizado o IP da máquina servidora para acessar a aplicação (por exemplo, `http://192.168.0.10:8000/`). Quando necessário, foram ajustados `ALLOWED_HOSTS` e permissões de rede (porta 8000) para permitir o acesso.

4.3 Fluxo de navegação utilizado nos testes

Os testes foram realizados navegando pelas telas do sistema: abrir a página de listagem, criar uma reserva via formulário e, por fim, remover um registro para confirmar a persistência e a atualização da interface.

5 Funcionalidades do Sistema

5.1 Listagem de reservas

O sistema disponibiliza uma página para consulta das reservas cadastradas. A listagem pode ser organizada por data, sala ou horário, permitindo visão rápida da ocupação. Essa funcionalidade è essencial para evitar conflitos e orientar a tomada de decisão sobre novos agendamentos.

5.2 Criação de reservas

A criação de reservas è realizada por meio de formulário web. Os campos recomendados incluem: sala, data, horário de início, horário de término e responsável. Após a submissão, a *view* valida os dados e persiste a reserva no banco de dados.

5.3 Remoção de reservas

A remoção permite cancelar um agendamento previamente registrado. Em implementações usuais, essa ação è feita por um botão de exclusão na listagem, que aciona uma rota específica e, após confirmação, remove o registro do banco.

5.4 Interface web

A interface web foi planejada para ser objetiva: uma página de listagem com acesso direto ao formulário de criação e ações de remoção. Essa abordagem minimiza a quantidade de cliques e reduz ambiguidades, facilitando o uso em cenários de consulta rápida.

5.5 Validação básica de dados

O sistema aplica validações básicas para garantir consistência: campos obrigatórios, tipos de dados corretos (datas e horários) e restrições simples (por exemplo, horário inicial anterior ao final). Quando uma validação falha, o usuário recebe feedback na própria página do formulário.

6 Interface da Aplicação

6.1 Descrição das páginas HTML

A aplicação pode ser organizada em páginas como:

- **Página inicial/listagem**: exibe as reservas cadastradas e atalhos para criação;
- **Página de criação**: apresenta formulário para inserir uma nova reserva;
- **Página de confirmação (opcional)**: confirmação de exclusão, prevenindo remoções acidentais.

6.2 Organização dos elementos

A organização prioriza legibilidade: cabeçalho com título, *container* central para o conteúdo, tabela ou lista para reservas e *feedback* de mensagens (sucesso/erro) em local visível. Esse padrão facilita a orientação do usuário e reduz esforço cognitivo.

6.3 Botões e formulários

Botões de ação (*Criar*, *Cancelar*, *Remover*) devem ser consistentes em cor e posicionamento. Os formulários devem indicar campos obrigatórios, fornecer rótulos claros e, quando possível, usar componentes adequados (como `type="date"` e `type="time"` em HTML) para reduzir erros de entrada.

6.4 Experiência do usuário

A experiência do usuário (UX) busca simplicidade e previsibilidade: o usuário deve entender rapidamente como consultar reservas e como registrar uma nova. Mensagens de validação e confirmação são importantes para orientar correções e dar segurança ao executar operações potencialmente destrutivas.

7 Testes e Execução

7.1 Como executar o projeto localmente

Em um ambiente local, a execução envolve, tipicamente:

1. Criar e ativar um ambiente virtual (opcional, mas recomendado);
2. Instalar dependências (por exemplo, via `pip`);
3. Aplicar migrações do Django (`python manage.py migrate`);

4. Iniciar o servidor (`python manage.py runserver`).

Após isso, a aplicação pode ser acessada em `http://127.0.0.1:8000/`.

7.2 Como acessar a aplicação em outro dispositivo da mesma rede

Para acesso via rede local, o servidor deve ser iniciado com `0.0.0.0` e a máquina deve estar conectada à mesma rede do dispositivo cliente. Em seguida, o cliente acessa pelo IP da máquina servidora, mantendo a porta configurada (por exemplo, 8000).

7.3 Exemplos de uso

Alguns fluxos de uso comuns incluem:

- Consultar a listagem para verificar disponibilidade;
- Registrar nova reserva preenchendo sala, data e horário;
- Remover uma reserva caso o agendamento seja cancelado;
- Repetir a consulta para confirmar que a alteração foi aplicada.

8 Limitações e Possíveis Melhorias

8.1 Limitações do servidor de desenvolvimento

O servidor de desenvolvimento do Django não foi projetado para produção. Em geral, ele não oferece o mesmo desempenho, segurança e recursos de robustez esperados em ambientes reais, como balanceamento de carga e gerenciamento avançado de conexões.

8.2 Possibilidade de uso futuro de Nginx ou outro servidor

Como melhoria, a aplicação pode ser implantada com um servidor apropriado (por exemplo, Nginx como *reverse proxy*) e um servidor de aplicação WSGI/ASGI (como Gunicorn ou uWSGI). Essa arquitetura tende a oferecer maior desempenho, melhor controle de arquivos estáticos e maior segurança operacional.

8.3 Melhorias visuais

O sistema pode ser aprimorado com um *design* responsivo, uso de componentes padronizados (por exemplo, bibliotecas CSS) e melhor comunicação visual de estados (mensagens, alertas, validações e confirmações).

8.4 Autenticação de usuários

Uma extensão natural do projeto è introduzir autenticação, permitindo que apenas usuários autorizados criem ou removam reservas. Além disso, pode-se registrar quem realizou cada agendamento e implementar perfis (administrador, usuário comum, etc.).

8.5 Persistência avançada de dados

Em cenários maiores, recomenda-se migrar de SQLite para um banco de dados mais robusto (por exemplo, PostgreSQL) e incluir mecanismos de auditoria, backup e regras de integridade mais estritas, como prevenção de sobreposição de horários por sala.

9 Conclusão

A atividade resultou em um **protótipo funcional** do **Sistema de Agendamento de Salas**, com as funcionalidades principais implementadas: listagem de reservas, criação via formulário e remoção de registros. O objetivo de entregar um sistema simples, executável localmente e testável na rede, foi atendido.

Como principal aprendizado, destacou-se a organização do projeto em arquivos separados (dados, telas e rotas) e a importância de validar entradas básicas antes de salvar informações no banco. Por fim, a atividade contribuiu para consolidar o fluxo completo de desenvolvimento: criação do projeto, implementação incremental, testes e ajustes até a entrega.

Referências

[1] Django Software Foundation. *Django Documentation*. Disponível em: `https://docs.djangoproject.com/`. Acesso em: 2026.

[2] Django Software Foundation. *Writing your first Django app*. Disponível em: `https://docs.djangoproject.com/en/stable/intro/tutorial01/`. Acesso em: 2026.

[3] MDN Web Docs. *HTTP overview*. Disponível em: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview`. Acesso em: 2026.