Plano de Implementação Backend — Projeto Fyyur (Versão Profissional)

> **Guia Completo e Didático para Implementação Backend Profissional**
> Utilizando Flask, SQLAlchemy, Pydantic, Flask-Migrate e boas práticas de engenharia de software.

📋 Índice

---

1. Visão Geral & Objetivos do Projeto

1.1 Contexto do Projeto

Fyyur é uma plataforma de agendamento de shows que conecta artistas locais e venues (locais de apresentação). A aplicação permite:

- Cadastro e busca de venues
- Cadastro e busca de artistas
- Agendamento de shows entre artistas e venues
- Visualização de shows passados e futuros

1.2 Objetivos Principais

1. **Substituir Mock Data**: Eliminar todos os dados fictícios e implementar persistência real com PostgreSQL
2. **Implementar CRUD Completo**: Create, Read, Update, Delete para todas as entidades
3. **Busca Avançada**: Implementar busca parcial, case-insensitive, com filtros
4. **Validação Robusta**: Usar Pydantic para garantir integridade dos dados
5. **Arquitetura Profissional**: Separação de responsabilidades, código testável e manutenível
6. **Qualidade de Código**: Seguir PEP 8, type hints, docstrings e princípios SOLID

1.3 Stack Tecnológica

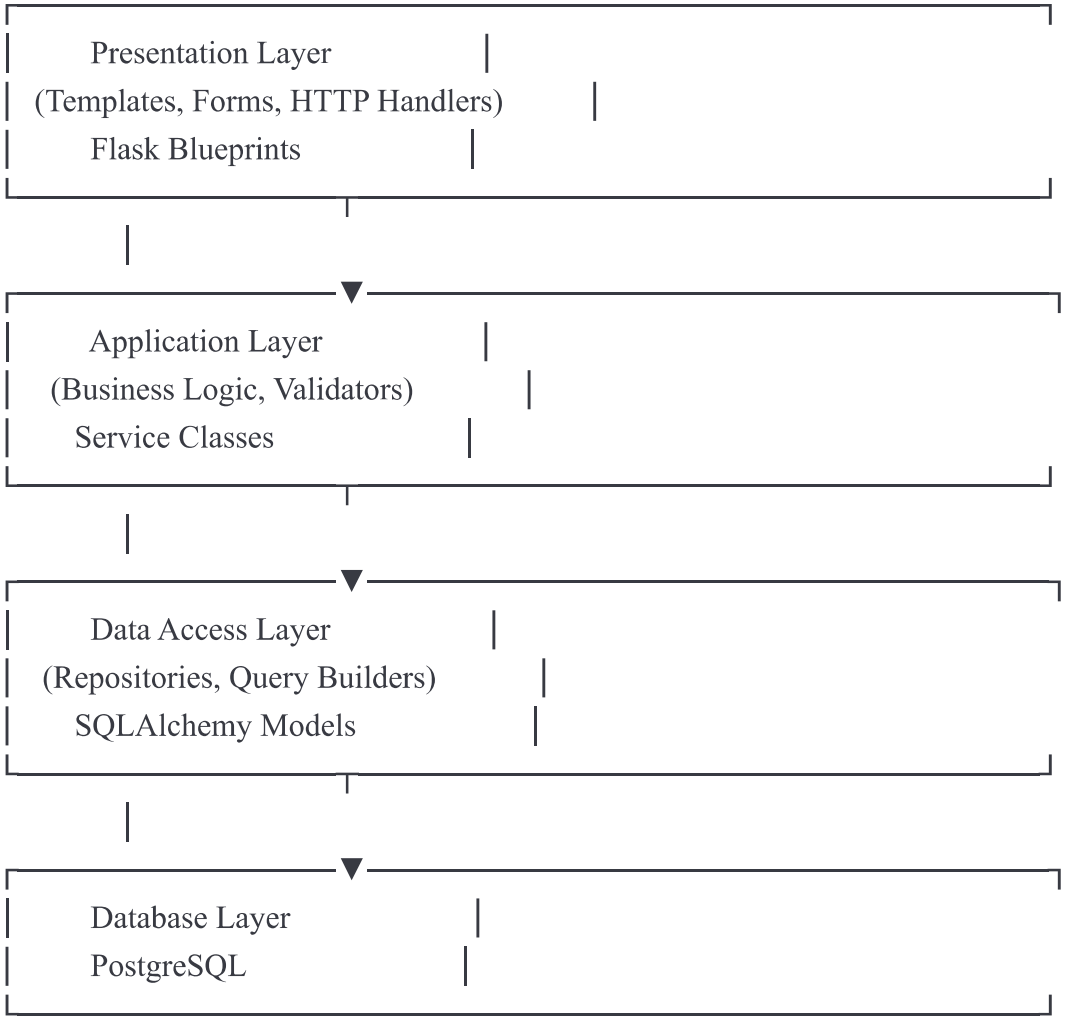
Backend:
├── Python 3.9-3.11
├── Flask 2.0.3 (WSGI Framework)
├── SQLAlchemy (ORM)
├── Flask-Migrate (Alembic wrapper)
├── Pydantic 2.x (Validação de dados)
├── Flask-WTF (Formulários web)
└── PostgreSQL 13+ (Banco de dados)

Desenvolvimento:
├── UV (Gerenciador de pacotes moderno)
├── pytest (Framework de testes)
├── black (Formatador de código)
├── mypy (Type checker)
├── flake8 (Linter)
└── pre-commit (Git hooks)

---

2. Arquitetura da Aplicação

2.1 Padrão Arquitetural: Layered Architecture



2.2 Fluxo de Requisição

1. Cliente → HTTP Request
2. Flask Router → Blueprint Controller
3. Controller → Service Layer (validação Pydantic)
4. Service → Repository (queries SQLAlchemy)
5. Repository → Database
6. Database → Retorna dados
7. Service → Serializa resposta
8. Controller → Renderiza template/JSON
9. Cliente ← HTTP Response

2.3 Princípios de Design

- **Single Responsibility**: Cada classe/função tem uma única responsabilidade
- **Dependency Injection**: Services recebem dependencies via construtor
- **Interface Segregation**: Interfaces pequenas e específicas
- **DRY (Don't Repeat Yourself)**: Reutilização de código
- **KISS (Keep It Simple, Stupid)**: Simplicidade primeiro

- **YAGNI (You Aren't Gonna Need It)**: Implementar apenas o necessário

---

3. Estrutura de Pastas & Organização do Código

3.1 Estrutura Proposta (Detalhada)

```
fyyur-project/
│
├── app/                          # Aplicação principal
│   ├── __init__.py               # Factory pattern para app
│   ├── config.py                 # Configurações por ambiente
│   │
│   ├── models/                   # SQLAlchemy models
│   │   ├── __init__.py
│   │   ├── base.py               # Base model com campos comuns
│   │   ├── venue.py              # Venue model
│   │   ├── artist.py             # Artist model
│   │   ├── show.py               # Show model
│   │   └── genre.py              # Genre model (opcional)
│   │
│   ├── schemas/                  # Pydantic schemas
│   │   ├── __init__.py
│   │   ├── venue_schema.py       # VenueCreate, VenueUpdate, VenueResponse
│   │   ├── artist_schema.py      # ArtistCreate, ArtistUpdate, ArtistResponse
│   │   ├── show_schema.py        # ShowCreate, ShowResponse
│   │   └── common.py             # Schemas compartilhados
│   │
│   ├── repositories/             # Data access layer
│   │   ├── __init__.py
│   │   ├── base_repository.py    # Repository genérico
│   │   ├── venue_repository.py   # Queries específicas de Venue
│   │   ├── artist_repository.py  # Queries específicas de Artist
│   │   └── show_repository.py    # Queries específicas de Show
│   │
│   ├── services/                 # Business logic
│   │   ├── __init__.py
│   │   ├── venue_service.py      # Lógica de negócio de Venue
│   │   ├── artist_service.py     # Lógica de negócio de Artist
│   │   └── show_service.py       # Lógica de negócio de Show
│   │
│   ├── controllers/              # Flask Blueprints
│   │   ├── __init__.py
│   │   ├── main.py               # Homepage, rotas principais
│   │   ├── venues.py             # CRUD de venues
│   │   ├── artists.py            # CRUD de artists
│   │   └── shows.py              # CRUD de shows
│   │
│   ├── utils/                    # Utilitários
│   │   ├── __init__.py
│   │   ├── validators.py         # Validadores customizados
```

```
│   │   ├── formatters.py          # Formatadores de data/telefone
│   │   ├── decorators.py          # Decoradores customizados
│   │   └── constants.py           # Constantes da aplicação
│   │
│   └── exceptions/                # Exceções customizadas
│       ├── __init__.py
│       ├── base.py                # BaseException customizada
│       └── errors.py              # Exceções específicas
│
├── migrations/                    # Flask-Migrate (Alembic)
│   ├── versions/
│   └── alembic.ini
│
├── tests/                         # Testes automatizados
│   ├── __init__.py
│   ├── conftest.py                # Fixtures do pytest
│   ├── fixtures/                  # Dados de teste
│   │   ├── __init__.py
│   │   ├── venue_fixtures.py
│   │   ├── artist_fixtures.py
│   │   └── show_fixtures.py
│   │
│   ├── unit/                      # Testes unitários
│   │   ├── test_models.py
│   │   ├── test_schemas.py
│   │   ├── test_services.py
│   │   └── test_repositories.py
│   │
│   ├── integration/               # Testes de integração
│   │   ├── test_venue_endpoints.py
│   │   ├── test_artist_endpoints.py
│   │   └── test_show_endpoints.py
│   │
│   └── e2e/                       # Testes end-to-end
│       └── test_full_workflow.py
│
├── scripts/                       # Scripts auxiliares
│   ├── seed.py                    # Seed de dados
│   ├── clean_db.py                # Limpar banco
│   └── check_db.py                # Verificar conexão
│
├── static/                        # Assets estáticos
│   ├── css/
│   ├── js/
```

```
│   ├── img/
│   └── fonts/
│
├── templates/                  # Templates Jinja2
│   ├── layouts/
│   ├── pages/
│   ├── forms/
│   └── errors/
│
├── forms.py                    # Flask-WTF forms (legacy)
├── app.py                      # Entry point da aplicação
├── config.py                   # Config root (se necessário)
├── .env.example                # Exemplo de variáveis
├── .gitignore
├── .pre-commit-config.yaml     # Pre-commit hooks
├── pyproject.toml              # Configuração do projeto
├── requirements.txt            # Dependências Python
├── pytest.ini                  # Configuração pytest
├── setup.py                    # Setup do pacote
└── README.md
```

3.2 Benefícios da Estrutura

1. **Separação de Responsabilidades**: Cada camada tem função específica
2. **Testabilidade**: Fácil mockar dependencies e testar isoladamente
3. **Manutenibilidade**: Código organizado e fácil de encontrar
4. **Escalabilidade**: Fácil adicionar novos recursos
5. **Reutilização**: Services e repositories podem ser compartilhados

---

4. Ambiente de Desenvolvimento com UV

4.1 Instalação do UV

bash

```bash
# macOS / Linux
curl -LsSf https://astral.sh/uv/install.sh | sh

# macOS via Homebrew
brew install uv

# Windows (PowerShell)
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"

# Verificar instalação
uv --version
```

4.2 Configuração do Projeto

bash

```bash
# 1. Navegar para o diretório do projeto
cd fyyur-project

# 2. Inicializar projeto UV (cria pyproject.toml)
uv init --name fyyur --python 3.11

# 3. Criar ambiente virtual
uv venv

# 4. Ativar ambiente
source .venv/bin/activate  # Linux/Mac
# ou
.venv\Scripts\activate     # Windows

# 5. Instalar dependências do requirements.txt existente
uv pip install -r requirements.txt

# 6. Adicionar novas dependências
uv add flask-migrate pydantic psycopg[binary] python-dotenv

# 7. Adicionar dependências de desenvolvimento
uv add --dev pytest pytest-flask pytest-cov black mypy flake8 pre-commit
```

4.3 Vantagens do UV

- **Velocidade**: 10-100x mais rápido que pip

- **Resolução de Dependências**: Resolve conflitos automaticamente
- **Lockfile**: Garante reprodutibilidade
- **Cache Global**: Economiza espaço em disco
- **Compatibilidade**: 100% compatível com pip/requirements.txt

---

5. Gestão de Dependências & Versões

5.1 Dependencies Principais

txt

```
# requirements.txt (atualizado)

# Flask Core
flask==2.3.3
flask-sqlalchemy==3.0.5
flask-migrate==4.0.5
flask-wtf==1.1.1
flask-moment==1.0.5

# Banco de Dados
psycopg[binary]==3.1.10
sqlalchemy==2.0.20

# Validação
pydantic==2.3.0
pydantic-settings==2.0.3
email-validator==2.0.0

# Utilitários
python-dotenv==1.0.0
babel==2.12.1
python-dateutil==2.8.2

# Segurança
werkzeug==2.3.7
```

5.2 Dependencies de Desenvolvimento

txt

```
# requirements-dev.txt

# Testing
pytest==7.4.2
pytest-flask==1.2.0
pytest-cov==4.1.0
pytest-mock==3.11.1
factory-boy==3.3.0
faker==19.6.2

# Code Quality
black==23.9.1
isort==5.12.0
flake8==6.1.0
mypy==1.5.1
pylint==2.17.5

# Pre-commit
pre-commit==3.4.0

# Database
alembic==1.12.0
```

5.3 Configuração do pyproject.toml

toml

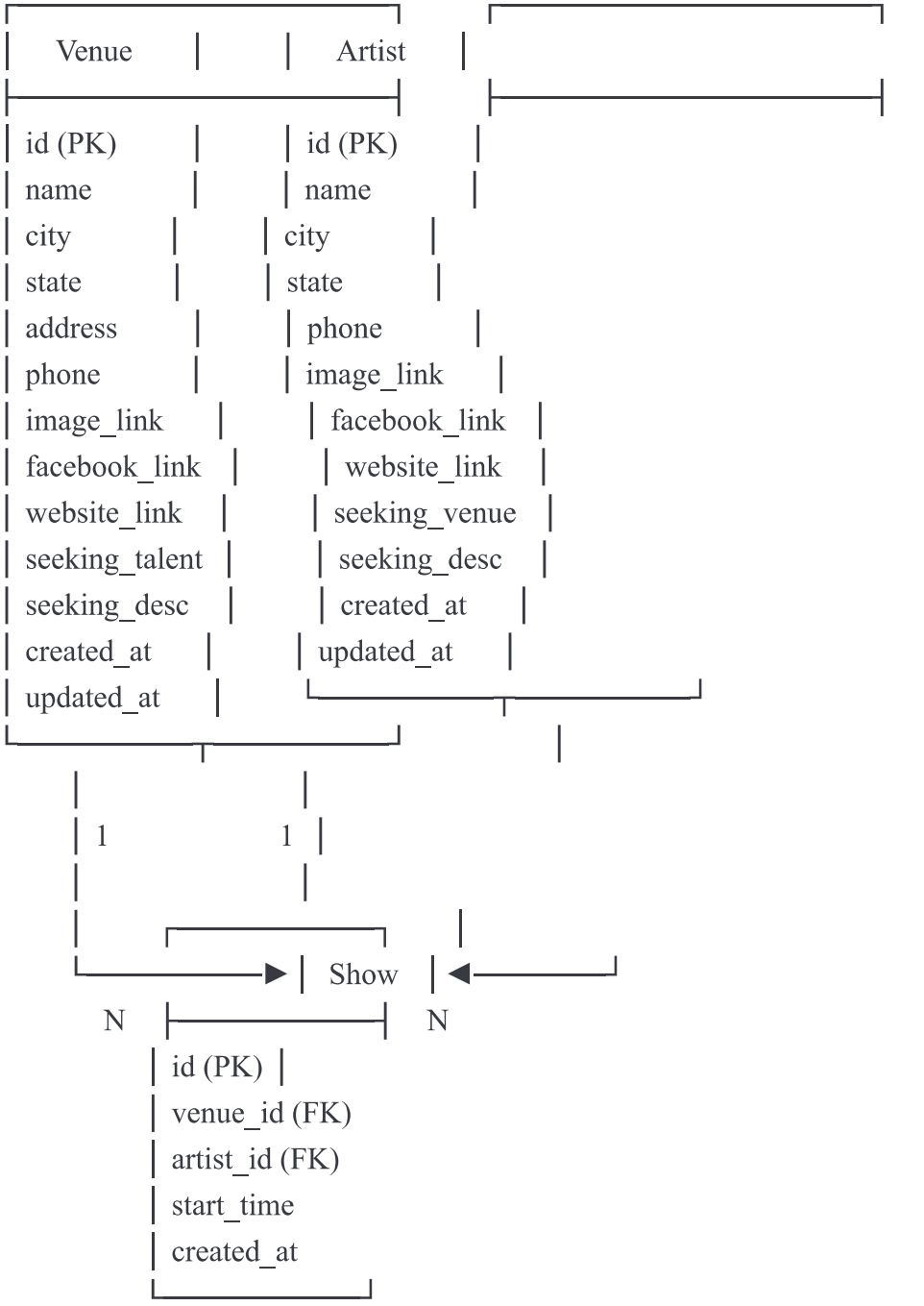
```toml
[project]
name = "fyyur"
version = "1.0.0"
description = "Musical venue and artist booking platform"
authors = [{name = "Your Name", email = "you@example.com"}]
requires-python = ">=3.9,<3.12"
dependencies = [
    "flask>=2.3.3",
    "flask-sqlalchemy>=3.0.5",
    "flask-migrate>=4.0.5",
    "pydantic>=2.3.0",
    "psycopg[binary]>=3.1.10",
]

[project.optional-dependencies]
dev = [
    "pytest>=7.4.2",
    "black>=23.9.1",
    "mypy>=1.5.1",
]

[tool.black]
line-length = 100
target-version = ['py39', 'py310', 'py311']
include = '\.pyi?$'
extend-exclude = '''
/(
  migrations
  | .venv
)/
'''

[tool.isort]
profile = "black"
line_length = 100

[tool.mypy]
python_version = "3.11"
warn_return_any = true
warn_unused_configs = true
disallow_untyped_defs = true

[tool.pytest.ini_options]
testpaths = ["tests"]
```

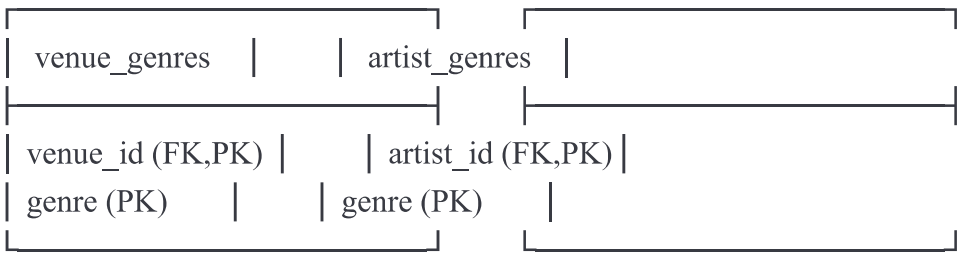
```
python_files = "test_*.py"
python_classes = "Test*"
python_functions = "test_*"
addopts = "-v --cov=app --cov-report=html --cov-report=term"
```

---

6. Modelagem de Dados & Design de Banco

6.1 Diagrama Entidade-Relacionamento





6.2 Decisões de Design

6.2.1 Normalização (3NF)

Tabela de Gêneros Separada:

- **Opção 1 (Recomendada)**: Association tables (`venue_genres`, `artist_genres`)
  - ✅ Normalizado (3NF)

    - ✅ Fácil adicionar novos gêneros
    - ✅ Evita duplicação
    - ❌ Mais tabelas e JOINs
- **Opção 2 (Simplificada)**: Array PostgreSQL
    - ✅ Menos tabelas
    - ✅ Queries mais simples
    - ❌ Menos portável
    - ❌ Difícil adicionar validação

Decisão: Usar association tables para melhor normalização e portabilidade.

6.2.2 Relacionamento Show

- **Tipo**: Many-to-Many através de Show (association object)
- **Justificativa**:
    - Um artista pode ter múltiplos shows
    - Um venue pode hospedar múltiplos shows
    - Show tem atributos próprios (start_time)

6.2.3 Timestamps

Todos os modelos terão:

- `created_at`: Quando foi criado (auto)
- `updated_at`: Última atualização (auto)

6.2.4 Soft Deletes (Opcional)

Para auditoria, considerar:

- `deleted_at`: NULL ou timestamp
- `is_deleted`: Boolean

6.3 Constraints e Validações no Banco


sql

```
-- Venue
CONSTRAINT pk_venue PRIMARY KEY (id)
CONSTRAINT nn_venue_name NOT NULL (name)
CONSTRAINT nn_venue_city NOT NULL (city)
CONSTRAINT ck_venue_state CHECK (LENGTH(state) = 2)
CONSTRAINT uq_venue_name_city UNIQUE (name, city)

-- Artist
CONSTRAINT pk_artist PRIMARY KEY (id)
CONSTRAINT nn_artist_name NOT NULL (name)
CONSTRAINT nn_artist_city NOT NULL (city)
CONSTRAINT ck_artist_state CHECK (LENGTH(state) = 2)

-- Show
CONSTRAINT pk_show PRIMARY KEY (id)
CONSTRAINT fk_show_venue FOREIGN KEY (venue_id) REFERENCES venues(id)
CONSTRAINT fk_show_artist FOREIGN KEY (artist_id) REFERENCES artists(id)
CONSTRAINT nn_show_start_time NOT NULL (start_time)
CONSTRAINT ck_show_future CHECK (start_time >= NOW())
CONSTRAINT uq_show_artist_time UNIQUE (artist_id, start_time)

-- Indexes
CREATE INDEX idx_venue_city_state ON venues(city, state);
CREATE INDEX idx_venue_name ON venues(name);
CREATE INDEX idx_artist_city_state ON artists(city, state);
CREATE INDEX idx_artist_name ON artists(name);
CREATE INDEX idx_show_start_time ON shows(start_time);
CREATE INDEX idx_show_venue ON shows(venue_id);
CREATE INDEX idx_show_artist ON shows(artist_id);
```

---

7. Validação com Pydantic

7.1 Filosofia de Validação em Camadas

1. **Frontend**: Validação básica (HTML5, JavaScript)
2. **WTForms**: Validação de formulário (compatibilidade com templates)
3. **Pydantic**: Validação estrita na camada de serviço
4. **Database**: Constraints finais

7.2 Estrutura de Schemas

python

```python
# schemas/base.py
from datetime import datetime
from typing import Optional
from pydantic import BaseModel, ConfigDict

class BaseSchema(BaseModel):
    """Base schema com configuração comum"""
    model_config = ConfigDict(from_attributes=True, validate_assignment=True)

class TimestampMixin(BaseModel):
    """Mixin para timestamps"""
    created_at: datetime
    updated_at: Optional[datetime] = None

# schemas/venue_schema.py
from typing import List, Optional
from pydantic import BaseModel, Field, HttpUrl, field_validator
from datetime import datetime

class VenueBase(BaseModel):
    """Campos comuns de Venue"""
    name: str = Field(..., min_length=1, max_length=255)
    city: str = Field(..., min_length=1, max_length=120)
    state: str = Field(..., min_length=2, max_length=2, pattern=r'^[A-Z]{2}$')
    address: str = Field(..., min_length=1, max_length=120)

    phone: Optional[str] = Field(None, pattern=r'^\d{3}-\d{3}-\d{4}$')
    image_link: Optional[HttpUrl] = None
    facebook_link: Optional[HttpUrl] = None
    website_link: Optional[HttpUrl] = None

    genres: List[str] = Field(..., min_length=1)
    seeking_talent: bool = False
    seeking_description: Optional[str] = Field(None, max_length=500)

    @field_validator('genres')
    @classmethod
    def validate_genres(cls, v: List[str]) -> List[str]:
        valid_genres = {
            'Alternative', 'Blues', 'Classical', 'Country', 'Electronic',
            'Folk', 'Funk', 'Hip-Hop', 'Heavy Metal', 'Instrumental',
            'Jazz', 'Musical Theatre', 'Pop', 'Punk', 'R&B',
            'Reggae', 'Rock n Roll', 'Soul', 'Other'
        }
```

```
        for genre in v:
            if genre not in valid_genres:
                raise ValueError(f'Invalid genre: {genre}')
        return v

    @field_validator('state')
    @classmethod
    def validate_state(cls, v: str) -> str:
        return v.upper()

class VenueCreate(VenueBase):
    """Schema para criação de Venue"""
    pass

class VenueUpdate(BaseModel):
    """Schema para atualização de Venue"""
    name: Optional[str] = Field(None, min_length=1, max_length=255)
    city: Optional[str] = None
    state: Optional[str] = None
    address: Optional[str] = None
    phone: Optional[str] = None
    image_link: Optional[HttpUrl] = None
    facebook_link: Optional[HttpUrl] = None
    website_link: Optional[HttpUrl] = None
    genres: Optional[List[str]] = None
    seeking_talent: Optional[bool] = None
    seeking_description: Optional[str] = None

class ShowSummary(BaseModel):
    """Show resumido para exibição na página de Venue"""
    artist_id: int
    artist_name: str
    artist_image_link: Optional[str] = None
    start_time: datetime

class VenueResponse(VenueBase, TimestampMixin):
    """Schema de resposta completo"""
    id: int
    num_upcoming_shows: int = 0
    num_past_shows: int = 0
    past_shows: List[ShowSummary] = []
    upcoming_shows: List[ShowSummary] = []

    model_config = ConfigDict(from_attributes=True)
```

```python
class VenueListItem(BaseModel):
    """Item de lista de venues"""
    id: int
    name: str
    num_upcoming_shows: int = 0

    model_config = ConfigDict(from_attributes=True)

class VenueSearchResponse(BaseModel):
    """Resposta de busca de venues"""
    count: int
    data: List[VenueListItem]
```

7.3 Validadores Customizados

python

```python
# utils/validators.py
import re
from typing import List
from pydantic import field_validator

def validate_phone(phone: str) -> str:
    """Valida formato de telefone"""
    pattern = r'^\d{3}-\d{3}-\d{4}$'
    if not re.match(pattern, phone):
        raise ValueError('Phone must be in format XXX-XXX-XXXX')
    return phone

def validate_state_code(state: str) -> str:
    """Valida código de estado US"""
    VALID_STATES = {
        'AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'DC', 'FL',
        'GA', 'HI', 'ID', 'IL', 'IN', 'IA', 'KS', 'KY', 'LA', 'ME',
        'MD', 'MA', 'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH',
        'NJ', 'NM', 'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI',
        'SC', 'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY'
    }
    state = state.upper()
    if state not in VALID_STATES:
        raise ValueError(f'Invalid state code: {state}')
    return state

def validate_genres(genres: List[str]) -> List[str]:
    """Valida lista de gêneros"""
    VALID_GENRES = {
        'Alternative', 'Blues', 'Classical', 'Country', 'Electronic',
        'Folk', 'Funk', 'Hip-Hop', 'Heavy Metal', 'Instrumental',
        'Jazz', 'Musical Theatre', 'Pop', 'Punk', 'R&B',
        'Reggae', 'Rock n Roll', 'Soul', 'Other'
    }

    if not genres:
        raise ValueError('At least one genre is required')

    invalid = set(genres) - VALID_GENRES
    if invalid:
        raise ValueError(f'Invalid genres: {", ".join(invalid)}')
```

```
    return genres
```

---

8. Camada de Serviços & Repositórios

8.1 Repository Pattern

python

```python
# repositories/base_repository.py
from typing import Generic, TypeVar, Type, List, Optional
from sqlalchemy.orm import Session
from app.models.base import BaseModel

T = TypeVar('T', bound=BaseModel)

class BaseRepository(Generic[T]):
    """Repository genérico com operações CRUD"""

    def __init__(self, model: Type[T], session: Session):
        self.model = model
        self.session = session

    def get_by_id(self, id: int) -> Optional[T]:
        """Busca por ID"""
        return self.session.query(self.model).filter_by(id=id).first()

    def get_all(self, skip: int = 0, limit: int = 100) -> List[T]:
        """Busca todos com paginação"""
        return self.session.query(self.model).offset(skip).limit(limit).all()

    def create(self, obj_in: dict) -> T:
        """Cria novo registro"""
        db_obj = self.model(**obj_in)
        self.session.add(db_obj)
        self.session.commit()
        self.session.refresh(db_obj)
        return db_obj

    def update(self, db_obj: T, obj_in: dict) -> T:
        """Atualiza registro existente"""
        for key, value in obj_in.items():
            if value is not None:
                setattr(db_obj, key, value)
        self.session.commit()
        self.session.refresh(db_obj)
        return db_obj

    def delete(self, id: int) -> bool:
        """Deleta registro"""
        db_obj = self.get_by_id(id)
        if db_obj:
            self.session.delete(db_obj)
```

```
            self.session.commit()
            return True
        return False

# repositories/venue_repository.py
from typing import List, Optional, Tuple
from sqlalchemy import func, or_
from datetime import datetime
from app.models.venue import Venue
from app.models.show import Show
from app.repositories.base_repository import BaseRepository

class VenueRepository(BaseRepository[Venue]):
    """Repository específico de Venue"""

    def search(
        self,
        search_term: str,
        city: Optional[str] = None,
        state: Optional[str] = None
    ) -> List[Venue]:
        """Busca venues por nome, cidade ou estado"""
        query = self.session.query(Venue)

        if search_term:
            query = query.filter(
                func.lower(Venue.name).like(f'%{search_term.lower()}%')
            )

        if city:
            query = query.filter(func.lower(Venue.city) == city.lower())

        if state:
            query = query.filter(func.lower(Venue.state) == state.lower())

        return query.all()

    def get_with_shows(self, venue_id: int) -> Optional[Venue]:
        """Busca venue com shows relacionados"""
        return (
            self.session.query(Venue)
            .filter_by(id=venue_id)
            .first()
        )
```

```
def get_grouped_by_location(self) -> List[dict]:
    """Agrupa venues por cidade/estado"""
    venues = self.session.query(Venue).all()

    locations = {}
    for venue in venues:
        key = (venue.city, venue.state)
        if key not in locations:
            locations[key] = {
                'city': venue.city,
                'state': venue.state,
                'venues': []
            }

        num_upcoming = self._count_upcoming_shows(venue.id)
        locations[key]['venues'].append({
            'id': venue.id,
            'name': venue.name,
            'num_upcoming_shows': num_upcoming
        })

    return list(locations.values())

def _count_upcoming_shows(self, venue_id: int) -> int:
    """Conta shows futuros de um venue"""
    return (
        self.session.query(Show)
        .filter(
            Show.venue_id == venue_id,
            Show.start_time > datetime.utcnow()
        )
        .count()
    )

def get_past_shows(self, venue_id: int) -> List[Show]:
    """Retorna shows passados"""
    return (
        self.session.query(Show)
        .filter(
            Show.venue_id == venue_id,
            Show.start_time < datetime.utcnow()
        )
        .order_by(Show.start_time.desc())
```

```
            .all()
        )

    def get_upcoming_shows(self, venue_id: int) -> List[Show]:
        """Retorna shows futuros"""
        return (
            self.session.query(Show)
            .filter(
                Show.venue_id == venue_id,
                Show.start_time >= datetime.utcnow()
            )
            .order_by(Show.start_time.asc())
            .all()
        )

    def get_recent(self, limit: int = 10) -> List[Venue]:
        """Retorna venues criados recentemente"""
        return (
            self.session.query(Venue)
            .order_by(Venue.created_at.desc())
            .limit(limit)
            .all()
        )
```

8.2 Service Pattern

python

```python
# services/venue_service.py
from typing import List, Optional
from sqlalchemy.orm import Session
from pydantic import ValidationError
from datetime import datetime

from app.repositories.venue_repository import VenueRepository
from app.schemas.venue_schema import (
    VenueCreate, VenueUpdate, VenueResponse, VenueSearchResponse
)
from app.exceptions.errors import (
    VenueNotFoundException, DuplicateVenueException, ValidationException
)

class VenueService:
    """Serviço com lógica de negócio de Venue"""

    def __init__(self, session: Session):
        self.session = session
        self.repository = VenueRepository(session)

    def create_venue(self, venue_data: VenueCreate) -> VenueResponse:
        """
        Cria um novo venue

        Args:
            venue_data: Dados validados do venue

        Returns:
            VenueResponse com dados do venue criado

        Raises:
            DuplicateVenueException: Se venue já existe
            ValidationException: Se dados inválidos
        """
        try:
            # Verificar duplicação
            existing = self.repository.search(
                search_term=venue_data.name,
                city=venue_data.city
            )
            if existing:
                raise DuplicateVenueException(
                    f"Venue '{venue_data.name}' already exists in {venue_data.city}"
```

```
            )

        # Criar venue
        venue_dict = venue_data.model_dump()

        # Processar genres (salvar em association table)
        genres = venue_dict.pop('genres', [])

        venue = self.repository.create(venue_dict)

        # Adicionar genres
        for genre in genres:
            # Lógica para adicionar à association table
            pass

        return self._to_response(venue)

    except ValidationError as e:
        raise ValidationException(str(e))

def get_venue(self, venue_id: int) -> VenueResponse:
    """Busca venue por ID com shows"""
    venue = self.repository.get_with_shows(venue_id)
    if not venue:
        raise VenueNotFoundException(f"Venue {venue_id} not found")

    return self._to_response(venue)

def update_venue(
    self,
    venue_id: int,
    venue_data: VenueUpdate
) -> VenueResponse:
    """Atualiza venue existente"""
    venue = self.repository.get_by_id(venue_id)
    if not venue:
        raise VenueNotFoundException(f"Venue {venue_id} not found")

    update_dict = venue_data.model_dump(exclude_unset=True)
    updated_venue = self.repository.update(venue, update_dict)

    return self._to_response(updated_venue)

def delete_venue(self, venue_id: int) -> bool:
```

```
    """Deleta venue"""
    venue = self.repository.get_by_id(venue_id)
    if not venue:
        raise VenueNotFoundException(f"Venue {venue_id} not found")

    # Verificar se tem shows futuros
    upcoming = self.repository.get_upcoming_shows(venue_id)
    if upcoming:
        raise ValidationException(
            "Cannot delete venue with upcoming shows"
        )

    return self.repository.delete(venue_id)

def search_venues(
    self,
    search_term: str,
    city: Optional[str] = None,
    state: Optional[str] = None
) -> VenueSearchResponse:
    """Busca venues"""
    venues = self.repository.search(search_term, city, state)

    data = [
        {
            'id': v.id,
            'name': v.name,
            'num_upcoming_shows': len(
                self.repository.get_upcoming_shows(v.id)
            )
        }
        for v in venues
    ]

    return VenueSearchResponse(count=len(data), data=data)

def list_venues_grouped(self) -> List[dict]:
    """Lista venues agrupados por cidade/estado"""
    return self.repository.get_grouped_by_location()

def _to_response(self, venue) -> VenueResponse:
    """Converte model para response schema"""
    past_shows = self.repository.get_past_shows(venue.id)
    upcoming_shows = self.repository.get_upcoming_shows(venue.id)
```

```
return VenueResponse(
    id=venue.id,
    name=venue.name,
    city=venue.city,
    state=venue.state,
    address=venue.address,
    phone=venue.phone,
    image_link=venue.image_link,
    facebook_link=venue.facebook_link,
    website_link=venue.website_link,
    genres=[],  # Buscar da association table
    seeking_talent=venue.seeking_talent,
    seeking_description=venue.seeking_description,
    created_at=venue.created_at,
    updated_at=venue.updated_at,
    past_shows=[
        {
            'artist_id': show.artist_id,
            'artist_name': show.artist.name,
            'artist_image_link': show.artist.image_link,
            'start_time': show.start_time
        }
        for show in past_shows
    ],
    upcoming_shows=[
        {
            'artist_id': show.artist_id,
            'artist_name': show.artist.name,
            'artist_image_link': show.artist.image_link,
            'start_time': show.start_time
        }
        for show in upcoming_shows
    ],
    num_past_shows=len(past_shows),
    num_upcoming_shows=len(upcoming_shows)
)
```

---

9. Controllers & Blueprints

9.1 Estrutura de Blueprint

python

```python
# controllers/venues.py
from flask import Blueprint, render_template, request, flash, redirect, url_for, jsonify
from pydantic import ValidationError

from app.services.venue_service import VenueService
from app.schemas.venue_schema import VenueCreate, VenueUpdate
from app.exceptions.errors import (
    VenueNotFoundException, DuplicateVenueException, ValidationException
)
from app.extensions import db
from forms import VenueForm

venues_bp = Blueprint('venues', __name__, url_prefix='/venues')

@venues_bp.route('/')
def index():
    """Lista todos os venues agrupados por localização"""
    try:
        service = VenueService(db.session)
        areas = service.list_venues_grouped()
        return render_template('pages/venues.html', areas=areas)
    except Exception as e:
        flash(f'Error loading venues: {str(e)}', 'error')
        return render_template('pages/venues.html', areas=[])

@venues_bp.route('/search', methods=['POST'])
def search():
    """Busca venues"""
    search_term = request.form.get('search_term', '')

    try:
        service = VenueService(db.session)
        results = service.search_venues(search_term)

        return render_template(
            'pages/search_venues.html',
            results=results.model_dump(),
            search_term=search_term
        )
    except Exception as e:
        flash(f'Search error: {str(e)}', 'error')
        return render_template(
            'pages/search_venues.html',
            results={'count': 0, 'data': []},
```

```
            search_term=search_term
        )

@venues_bp.route('/<int:venue_id>')
def show(venue_id):
    """Exibe detalhes de um venue"""
    try:
        service = VenueService(db.session)
        venue = service.get_venue(venue_id)
        return render_template(
            'pages/show_venue.html',
            venue=venue.model_dump()
        )
    except VenueNotFoundException:
        flash(f'Venue {venue_id} not found', 'error')
        return redirect(url_for('venues.index'))
    except Exception as e:
        flash(f'Error: {str(e)}', 'error')
        return redirect(url_for('venues.index'))

@venues_bp.route('/create', methods=['GET'])
def create_form():
    """Renderiza formulário de criação"""
    form = VenueForm()
    return render_template('forms/new_venue.html', form=form)

@venues_bp.route('/create', methods=['POST'])
def create_submit():
    """Processa criação de venue"""
    form = VenueForm(request.form)

    if not form.validate():
        flash('Form validation failed', 'error')
        return render_template('forms/new_venue.html', form=form)

    try:
        # Converter form para Pydantic schema
        venue_data = VenueCreate(
            name=form.name.data,
            city=form.city.data,
            state=form.state.data,
            address=form.address.data,
            phone=form.phone.data,
            image_link=form.image_link.data,
```

```
            facebook_link=form.facebook_link.data,
            website_link=form.website_link.data,
            genres=form.genres.data,
            seeking_talent=form.seeking_talent.data,
            seeking_description=form.seeking_description.data
        )

        service = VenueService(db.session)
        venue = service.create_venue(venue_data)

        flash(f'Venue {venue.name} was successfully listed!', 'success')
        return redirect(url_for('venues.show', venue_id=venue.id))

    except DuplicateVenueException as e:
        flash(str(e), 'error')
        return render_template('forms/new_venue.html', form=form)
    except ValidationException as e:
        flash(f'Validation error: {str(e)}', 'error')
        return render_template('forms/new_venue.html', form=form)
    except Exception as e:
        flash(f'An error occurred: {str(e)}', 'error')
        return render_template('forms/new_venue.html', form=form)

@venues_bp.route('/<int:venue_id>/edit', methods=['GET'])
def edit_form(venue_id):
    """Renderiza formulário de edição"""
    try:
        service = VenueService(db.session)
        venue = service.get_venue(venue_id)

        # Popular form com dados existentes
        form = VenueForm(obj=venue)

        return render_template(
            'forms/edit_venue.html',
            form=form,
            venue=venue.model_dump()
        )
    except VenueNotFoundException:
        flash(f'Venue {venue_id} not found', 'error')
        return redirect(url_for('venues.index'))

@venues_bp.route('/<int:venue_id>/edit', methods=['POST'])
def edit_submit(venue_id):
```

```
    """Processa atualização de venue"""
    form = VenueForm(request.form)

    if not form.validate():
        flash('Form validation failed', 'error')
        return render_template('forms/edit_venue.html', form=form)

    try:
        # Apenas campos modificados
        venue_data = VenueUpdate(
            **{k: v for k, v in form.data.items() if v is not None}
        )

        service = VenueService(db.session)
        venue = service.update_venue(venue_id, venue_data)

        flash(f'Venue {venue.name} updated successfully!', 'success')
        return redirect(url_for('venues.show', venue_id=venue.id))

    except VenueNotFoundException:
        flash(f'Venue {venue_id} not found', 'error')
        return redirect(url_for('venues.index'))
    except Exception as e:
        flash(f'Error: {str(e)}', 'error')
        return render_template('forms/edit_venue.html', form=form)

@venues_bp.route('/<int:venue_id>/delete', methods=['POST'])
def delete(venue_id):
    """Deleta venue"""
    try:
        service = VenueService(db.session)
        service.delete_venue(venue_id)
        flash(f'Venue deleted successfully', 'success')
        return redirect(url_for('venues.index'))
    except VenueNotFoundException:
        flash(f'Venue {venue_id} not found', 'error')
        return redirect(url_for('venues.index'))
    except ValidationException as e:
        flash(str(e), 'error')
        return redirect(url_for('venues.show', venue_id=venue_id))

# API Endpoints (JSON responses)
@venues_bp.route('/api/venues', methods=['GET'])
def api_list():
```

```python
    """API: Lista venues"""
    try:
        service = VenueService(db.session)
        venues = service.list_venues_grouped()
        return jsonify(venues), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@venues_bp.route('/api/venues/<int:venue_id>', methods=['GET'])
def api_get(venue_id):
    """API: Busca venue"""
    try:
        service = VenueService(db.session)
        venue = service.get_venue(venue_id)
        return jsonify(venue.model_dump()), 200
    except VenueNotFoundException:
        return jsonify({'error': 'Venue not found'}), 404
```

9.2 Registro de Blueprints

python

```python
# app/__init__.py
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_moment import Moment

db = SQLAlchemy()
migrate = Migrate()
moment = Moment()

def create_app(config_name='development'):
    """Application Factory"""
    app = Flask(__name__)

    # Configurações
    app.config.from_object(f'config.{config_name.capitalize()}Config')

    # Inicializar extensões
    db.init_app(app)
    migrate.init_app(app, db)
    moment.init_app(app)

    # Registrar Blueprints
    from app.controllers.main import main_bp
    from app.controllers.venues import venues_bp
    from app.controllers.artists import artists_bp
    from app.controllers.shows import shows_bp

    app.register_blueprint(main_bp)
    app.register_blueprint(venues_bp)
    app.register_blueprint(artists_bp)
    app.register_blueprint(shows_bp)

    # Error handlers
    register_error_handlers(app)

    # Jinja filters
    register_filters(app)

    return app

def register_error_handlers(app):
    """Registra handlers de erro"""
    @app.errorhandler(404)
```

```python
    def not_found(error):
        return render_template('errors/404.html'), 404

    @app.errorhandler(500)
    def server_error(error):
        return render_template('errors/500.html'), 500

def register_filters(app):
    """Registra filtros Jinja customizados"""
    from app.utils.formatters import format_datetime
    app.jinja_env.filters['datetime'] = format_datetime
```

---

10. Tratamento de Erros & Logging

10.1 Exceções Customizadas

python

```python
# exceptions/base.py
class FyyurException(Exception):
    """Base exception para a aplicação"""
    def __init__(self, message: str, status_code: int = 400):
        self.message = message
        self.status_code = status_code
        super().__init__(self.message)

# exceptions/errors.py
class VenueNotFoundException(FyyurException):
    def __init__(self, message: str):
        super().__init__(message, status_code=404)

class DuplicateVenueException(FyyurException):
    def __init__(self, message: str):
        super().__init__(message, status_code=409)

class ValidationException(FyyurException):
    def __init__(self, message: str):
        super().__init__(message, status_code=422)

class DatabaseException(FyyurException):
    def __init__(self, message: str):
        super().__init__(message, status_code=500)
```

10.2 Configuração de Logging

python

```python
# config.py
import logging
from logging.handlers import RotatingFileHandler
import os

class Config:
    # Logging
    LOG_LEVEL = os.getenv('LOG_LEVEL', 'INFO')
    LOG_FILE = 'logs/fyyur.log'
    LOG_MAX_BYTES = 10485760  # 10MB
    LOG_BACKUP_COUNT = 10

def setup_logging(app):
    """Configura sistema de logging"""

    # Criar diretório de logs
    os.makedirs('logs', exist_ok=True)

    # Formatter
    formatter = logging.Formatter(
        '[%(asctime)s] %(levelname)s in %(module)s: %(message)s'
    )

    # File handler
    file_handler = RotatingFileHandler(
        app.config['LOG_FILE'],
        maxBytes=app.config['LOG_MAX_BYTES'],
        backupCount=app.config['LOG_BACKUP_COUNT']
    )
    file_handler.setFormatter(formatter)
    file_handler.setLevel(getattr(logging, app.config['LOG_LEVEL']))

    # Console handler
    console_handler = logging.StreamHandler()
    console_handler.setFormatter(formatter)
    console_handler.setLevel(logging.DEBUG)

    # Configurar logger da aplicação
    app.logger.addHandler(file_handler)
    app.logger.addHandler(console_handler)
    app.logger.setLevel(getattr(logging, app.config['LOG_LEVEL']))

    # Logging de SQL (apenas em dev)
```

```
    if app.config['DEBUG']:
        logging.getLogger('sqlalchemy.engine').setLevel(logging.INFO)
```

10.3 Decorador para Logging

python

```python
# utils/decorators.py
import functools
import time
import logging
from flask import request

logger = logging.getLogger(__name__)

def log_execution_time(func):
    """Loga tempo de execução de uma função"""
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()

        logger.info(
            f"{func.__name__} executed in {end_time - start_time:.4f}s"
        )
        return result
    return wrapper

def log_request(func):
    """Loga detalhes da requisição"""
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        logger.info(
            f"Request: {request.method} {request.path} "
            f"from {request.remote_addr}"
        )
        return func(*args, **kwargs)
    return wrapper
```

---

11. Migrações de Banco & Seeds

11.1 Configuração Flask-Migrate

bash

```
# Inicializar migrações
flask db init

# Criar migração
flask db migrate -m "Initial models: Venue, Artist, Show"

# Aplicar migração
flask db upgrade

# Reverter migração
flask db downgrade

# Mostrar histórico
flask db history
```

11.2 Script de Seed

python

```python
# scripts/seed.py
"""Seed script para popular banco com dados de teste"""
import sys
import os

# Adicionar diretório raiz ao path
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from app import create_app, db
from app.models.venue import Venue
from app.models.artist import Artist
from app.models.show import Show
from datetime import datetime, timedelta

def seed_venues():
    """Seed de venues"""
    venues = [
        {
            'name': 'The Musical Hop',
            'city': 'San Francisco',
            'state': 'CA',
            'address': '1015 Folsom Street',
            'phone': '123-123-1234',
            'genres': ['Jazz', 'Reggae', 'Swing', 'Classical', 'Folk'],
            'image_link': 'https://images.unsplash.com/photo-1543900694-133f37abaaa5',
            'facebook_link': 'https://www.facebook.com/TheMusicalHop',
            'website_link': 'https://www.themusicalhop.com',
            'seeking_talent': True,
            'seeking_description': 'Looking for local artists'
        },
        {
            'name': 'The Dueling Pianos Bar',
            'city': 'New York',
            'state': 'NY',
            'address': '335 Delancey Street',
            'phone': '914-003-1132',
            'genres': ['Classical', 'R&B', 'Hip-Hop'],
            'image_link': 'https://images.unsplash.com/photo-1497032205916',
            'facebook_link': 'https://www.facebook.com/theduelingpianos',
            'website_link': 'https://www.theduelingpianos.com',
            'seeking_talent': False
        }
    ]
```

```
    for venue_data in venues:
        genres = venue_data.pop('genres')
        venue = Venue(**venue_data)
        db.session.add(venue)
        db.session.flush()

        # Adicionar genres (association table)
        for genre in genres:
            # Adicionar à association table
            pass

    db.session.commit()
    print(f"✓ Seeded {len(venues)} venues")

def seed_artists():
    """Seed de artists"""
    artists = [
        {
            'name': 'Guns N Petals',
            'city': 'San Francisco',
            'state': 'CA',
            'phone': '326-123-5000',
            'genres': ['Rock n Roll'],
            'image_link': 'https://images.unsplash.com/photo-1549213783',
            'facebook_link': 'https://www.facebook.com/GunsNPetals',
            'website_link': 'https://www.gunsnpetalsband.com',
            'seeking_venue': True,
            'seeking_description': 'Looking for shows'
        }
    ]

    for artist_data in artists:
        genres = artist_data.pop('genres')
        artist = Artist(**artist_data)
        db.session.add(artist)
        db.session.flush()

        # Adicionar genres
        for genre in genres:
            pass

    db.session.commit()
    print(f"✓ Seeded {len(artists)} artists")
```

```python
def seed_shows():
    """Seed de shows"""
    # Buscar venues e artists
    venue = Venue.query.first()
    artist = Artist.query.first()

    if not venue or not artist:
        print("✗ Skipping shows: no venues or artists found")
        return

    shows = [
        {
            'venue_id': venue.id,
            'artist_id': artist.id,
            'start_time': datetime.now() + timedelta(days=7)
        },
        {
            'venue_id': venue.id,
            'artist_id': artist.id,
            'start_time': datetime.now() - timedelta(days=7)
        }
    ]

    for show_data in shows:
        show = Show(**show_data)
        db.session.add(show)

    db.session.commit()
    print(f"✓ Seeded {len(shows)} shows")

def clear_data():
    """Limpa todos os dados"""
    Show.query.delete()
    Artist.query.delete()
    Venue.query.delete()
    db.session.commit()
    print("✓ Cleared all data")

def main():
    """Executa seed"""
    app = create_app('development')

    with app.app_context():
        print("Starting seed...")
```

```
    # Limpar dados existentes
    clear_data()

    # Seed
    seed_venues()
    seed_artists()
    seed_shows()

    print("✓ Seed completed successfully!")

if __name__ == '__main__':
    main()
```

11.3 Comandos Flask Customizados

python

```python
# commands.py
import click
from flask.cli import with_appcontext
from app import db

@click.command('init-db')
@with_appcontext
def init_db_command():
    """Cria tabelas do banco"""
    db.create_all()
    click.echo('✓ Database initialized')

@click.command('seed-db')
@with_appcontext
def seed_db_command():
    """Popula banco com dados de teste"""
    from scripts.seed import seed_venues, seed_artists, seed_shows
    seed_venues()
    seed_artists()
    seed_shows()
    click.echo('✓ Database seeded')

@click.command('reset-db')
@with_appcontext
def reset_db_command():
    """Reseta banco (cuidado!)"""
    if click.confirm('This will delete all data. Continue?'):
        db.drop_all()
        db.create_all()
        click.echo('✓ Database reset')

def register_commands(app):
    """Registra comandos CLI"""
    app.cli.add_command(init_db_command)
    app.cli.add_command(seed_db_command)
    app.cli.add_command(reset_db_command)
```

---

12. Testes Automatizados

12.1 Estrutura de Testes

python

```python
# tests/conftest.py
import pytest
from app import create_app, db as _db
from app.models.venue import Venue
from app.models.artist import Artist
from app.models.show import Show

@pytest.fixture(scope='session')
def app():
    """Cria aplicação para testes"""
    app = create_app('testing')
    return app

@pytest.fixture(scope='session')
def client(app):
    """Cliente de teste"""
    return app.test_client()

@pytest.fixture(scope='function')
def db(app):
    """Banco de dados para cada teste"""
    with app.app_context():
        _db.create_all()
        yield _db
        _db.session.close()
        _db.drop_all()

@pytest.fixture
def venue_data():
    """Dados de venue para testes"""
    return {
        'name': 'Test Venue',
        'city': 'San Francisco',
        'state': 'CA',
        'address': '123 Test St',
        'phone': '123-456-7890',
        'genres': ['Jazz', 'Blues'],
        'seeking_talent': True
    }

@pytest.fixture
def artist_data():
    """Dados de artist para testes"""
    return {
```

```
    'name': 'Test Artist',
    'city': 'New York',
    'state': 'NY',
    'phone': '987-654-3210',
    'genres': ['Rock n Roll'],
    'seeking_venue': True
}
```

12.2 Testes Unitários

python

```python
# tests/unit/test_models.py
import pytest
from datetime import datetime
from app.models.venue import Venue
from app.models.artist import Artist
from app.models.show import Show

def test_venue_creation(db, venue_data):
    """Testa criação de venue"""
    venue = Venue(**venue_data)
    db.session.add(venue)
    db.session.commit()

    assert venue.id is not None
    assert venue.name == venue_data['name']
    assert venue.city == venue_data['city']
    assert venue.created_at is not None

def test_artist_creation(db, artist_data):
    """Testa criação de artist"""
    artist = Artist(**artist_data)
    db.session.add(artist)
    db.session.commit()

    assert artist.id is not None
    assert artist.name == artist_data['name']

def test_show_relationship(db, venue_data, artist_data):
    """Testa relacionamento entre Show, Venue e Artist"""
    venue = Venue(**venue_data)
    artist = Artist(**artist_data)

    db.session.add(venue)
    db.session.add(artist)
    db.session.commit()

    show = Show(
        venue_id=venue.id,
        artist_id=artist.id,
        start_time=datetime.now()
    )
    db.session.add(show)
    db.session.commit()
```

```python
    assert show.venue.name == venue.name
    assert show.artist.name == artist.name

# tests/unit/test_schemas.py
import pytest
from pydantic import ValidationError
from app.schemas.venue_schema import VenueCreate

def test_venue_schema_valid(venue_data):
    """Testa validação de schema válido"""
    schema = VenueCreate(**venue_data)
    assert schema.name == venue_data['name']
    assert schema.state == venue_data['state'].upper()

def test_venue_schema_invalid_state():
    """Testa validação de estado inválido"""
    with pytest.raises(ValidationError):
        VenueCreate(
            name='Test',
            city='City',
            state='XXX',  # Inválido
            address='123 St',
            genres=['Jazz']
        )

def test_venue_schema_invalid_phone():
    """Testa validação de telefone inválido"""
    with pytest.raises(ValidationError):
        VenueCreate(
            name='Test',
            city='City',
            state='CA',
            address='123 St',
            phone='invalid',  # Inválido
            genres=['Jazz']
        )
```

12.3 Testes de Integração

python

```python
# tests/integration/test_venue_endpoints.py
import pytest
from flask import url_for

def test_venue_list(client, db, venue_data):
    """Testa listagem de venues"""
    # Criar venue
    from app.models.venue import Venue
    venue = Venue(**venue_data)
    db.session.add(venue)
    db.session.commit()

    # Fazer request
    response = client.get('/venues')

    assert response.status_code == 200
    assert venue.name.encode() in response.data

def test_venue_search(client, db, venue_data):
    """Testa busca de venues"""
    from app.models.venue import Venue
    venue = Venue(**venue_data)
    db.session.add(venue)
    db.session.commit()

    response = client.post('/venues/search', data={
        'search_term': 'Test'
    })

    assert response.status_code == 200
    assert b'Test Venue' in response.data

def test_venue_creation(client):
    """Testa criação de venue via form"""
    response = client.post('/venues/create', data={
        'name': 'New Venue',
        'city': 'San Francisco',
        'state': 'CA',
        'address': '456 New St',
        'phone': '111-222-3333',
        'genres': ['Jazz', 'Blues'],
        'seeking_talent': True
    })
```

```
    assert response.status_code == 302  # Redirect

def test_venue_detail(client, db, venue_data):
    """Testa página de detalhes"""
    from app.models.venue import Venue
    venue = Venue(**venue_data)
    db.session.add(venue)
    db.session.commit()

    response = client.get(f'/venues/{venue.id}')

    assert response.status_code == 200
    assert venue.name.encode() in response.data
```

12.4 Executar Testes

bash

```
# Todos os testes
pytest

# Com coverage
pytest --cov=app --cov-report=html

# Testes específicos
pytest tests/unit/test_models.py

# Verbose
pytest -v

# Parar no primeiro erro
pytest -x

# Ver print statements
pytest -s
```

---

13. Segurança & Boas Práticas

13.1 Configuração de Segurança

python

```python
# config.py
import os
from datetime import timedelta

class Config:
    # Security
    SECRET_KEY = os.getenv('SECRET_KEY', os.urandom(32))
    WTF_CSRF_ENABLED = True
    WTF_CSRF_TIME_LIMIT = None

    # Session
    SESSION_COOKIE_SECURE = True  # HTTPS only
    SESSION_COOKIE_HTTPONLY = True
    SESSION_COOKIE_SAMESITE = 'Lax'
    PERMANENT_SESSION_LIFETIME = timedelta(hours=24)

    # Database
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SQLALCHEMY_ECHO = False  # Não logar SQL em produção

    # Rate limiting (usar Flask-Limiter)
    RATELIMIT_ENABLED = True
    RATELIMIT_DEFAULT = "100 per hour"

class ProductionConfig(Config):
    DEBUG = False
    TESTING = False

    # Force HTTPS
    PREFERRED_URL_SCHEME = 'https'
```

13.2 Input Sanitization

python

```python
# utils/sanitizers.py
import html
import re
from typing import Optional

def sanitize_string(value: Optional[str]) -> Optional[str]:
    """Sanitiza string para prevenir XSS"""
    if value is None:
        return None
    # HTML escape
    value = html.escape(value.strip())
    return value

def sanitize_phone(phone: Optional[str]) -> Optional[str]:
    """Sanitiza telefone"""
    if phone is None:
        return None
    # Remove tudo exceto números e hífens
    phone = re.sub(r'[^0-9-]', '', phone)
    # Formatar XXX-XXX-XXXX
    digits = re.sub(r'-', '', phone)
    if len(digits) == 10:
        return f"{digits[:3]}-{digits[3:6]}-{digits[6:]}"
    return phone
```

13.3 SQL Injection Prevention

python

```python
# SEMPRE usar parametrização com SQLAlchemy
# ✓ CORRETO
user = User.query.filter_by(email=user_input).first()

# ✓ CORRETO
result = db.session.execute(
    text("SELECT * FROM users WHERE email = :email"),
    {"email": user_input}
)

# ✗ ERRADO - Vulnerável a SQL injection
query = f"SELECT * FROM users WHERE email = '{user_input}'"
result = db.session.execute(text(query))
```

13.4 Environment Variables

bash

```bash
# .env.example
# Database
DATABASE_URL=postgresql://user:password@localhost:5432/fyyur

# Security
SECRET_KEY=your-secret-key-here
WTF_CSRF_SECRET_KEY=csrf-secret

# Flask
FLASK_APP=app.py
FLASK_ENV=development

# Logging
LOG_LEVEL=INFO
```

python

```python
# Carregar .env
from dotenv import load_dotenv
load_dotenv()

# Usar variáveis
import os
DATABASE_URL = os.getenv('DATABASE_URL')
```

---

14. Documentação de API

14.1 Swagger/OpenAPI (Opcional)


python

```python
# Instalar flask-swagger-ui
uv add flask-swagger-ui

# app/__init__.py
from flask_swagger_ui import get_swaggerui_blueprint

def create_app():
    # ... código existente ...

    # Swagger UI
    SWAGGER_URL = '/api/docs'
    API_URL = '/static/swagger.json'
    swaggerui_blueprint = get_swaggerui_blueprint(
        SWAGGER_URL,
        API_URL,
        config={'app_name': "Fyyur API"}
    )
    app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)
```

14.2 Docstrings Padronizadas


python

```
def create_venue(self, venue_data: VenueCreate) -> VenueResponse:
    """
    Cria um novo venue no sistema.

    Este método valida os dados do venue, verifica duplicações e
    persiste o novo venue no banco de dados.

    Args:
        venue_data: Dados validados do venue (VenueCreate schema)

    Returns:
        VenueResponse: Objeto com dados completos do venue criado,
                       incluindo ID gerado e timestamps

    Raises:
        DuplicateVenueException: Se já existe venue com mesmo nome/cidade
        ValidationException: Se dados forem inválidos após validação Pydantic
        DatabaseException: Se ocorrer erro ao salvar no banco

    Example:
        >>> venue_data = VenueCreate(
        ...     name="Jazz Club",
        ...     city="New York",
        ...     state="NY",
        ...     address="123 Main St",
        ...     genres=["Jazz", "Blues"]
        ... )
        >>> service = VenueService(db.session)
        >>> venue = service.create_venue(venue_data)
        >>> print(venue.id)
        1

    Note:
        - Nome + cidade devem ser únicos
        - Gêneros são validados contra lista predefinida
        - Timestamps são gerados automaticamente
    """
    # Implementação...
```

---

15. Performance & Otimização

15.1 Query Optimization

python

```python
# ✗ N+1 Problem - EVITAR
venues = Venue.query.all()
for venue in venues:
    # Cada iteração faz query adicional
    shows = venue.shows  # Query!

# ✓ Eager Loading - USAR
from sqlalchemy.orm import joinedload

venues = Venue.query.options(
    joinedload(Venue.shows)
).all()

for venue in venues:
    shows = venue.shows  # Já carregado, sem query
```

15.2 Indexes

python

```python
# models/venue.py
class Venue(db.Model):
    __tablename__ = 'venues'

    # Adicionar indexes para campos frequentemente buscados
    __table_args__ = (
        db.Index('idx_venue_city_state', 'city', 'state'),
        db.Index('idx_venue_name', 'name'),
    )
```

15.3 Caching (Opcional)

python

```python
# Instalar Flask-Caching
uv add Flask-Caching

# app/__init__.py
from flask_caching import Cache

cache = Cache(config={
    'CACHE_TYPE': 'simple',  # ou 'redis' em produção
    'CACHE_DEFAULT_TIMEOUT': 300
})

# Uso
@cache.cached(timeout=300)
def get_venues():
    return Venue.query.all()
```

15.4 Paginação

python

```python
# repositories/venue_repository.py
def get_all_paginated(
    self,
    page: int = 1,
    per_page: int = 20
) -> dict:
    """Retorna venues paginados"""
    pagination = (
        self.session.query(Venue)
        .paginate(page=page, per_page=per_page, error_out=False)
    )

    return {
        'items': pagination.items,
        'total': pagination.total,
        'page': page,
        'per_page': per_page,
        'pages': pagination.pages
    }
```

---

16. Roadmap de Implementação

FASE 1: Setup e Fundação (Dias 1-2)

Dia 1: Configuração Inicial

- [ ] Setup UV e ambiente virtual
- [ ] Instalar dependências principais
- [ ] Configurar pre-commit hooks
- [ ] Estruturar pastas do projeto
- [ ] Configurar config.py com ambientes
- [ ] Setup PostgreSQL local
- [ ] Criar .env e .gitignore
- [ ] Inicializar Git

Comandos:


bash

```
cd fyyur-project
uv venv
source .venv/bin/activate
uv add -r requirements.txt
uv add --dev pytest black mypy
pre-commit install
createdb fyyur_dev
```

Dia 2: Modelos e Migrações

- [ ] Criar models/base.py com BaseModel
- [ ] Implementar models/venue.py
- [ ] Implementar models/artist.py
- [ ] Implementar models/show.py
- [ ] Configurar Flask-Migrate
- [ ] Criar primeira migração
- [ ] Aplicar migrações
- [ ] Testar conexão com banco

Comandos:


bash

```
flask db init
flask db migrate -m "Initial models"
flask db upgrade
python scripts/check_db.py
```

FASE 2: Camada de Dados (Dias 3-4)

Dia 3: Repositories

- [ ] Criar repositories/base_repository.py
- [ ] Implementar VenueRepository
- [ ] Implementar ArtistRepository
- [ ] Implementar ShowRepository
- [ ] Escrever testes unitários de repositories
- [ ] Documentar queries complexas

Dia 4: Services e Schemas

- [ ] Criar todos os Pydantic schemas
- [ ] Implementar VenueService
- [ ] Implementar ArtistService
- [ ] Implementar ShowService
- [ ] Validadores customizados
- [ ] Testes de services

FASE 3: Controllers e API (Dias 5-6)

Dia 5: Blueprints de Venues

- [ ] Criar venues blueprint
- [ ] Implementar GET /venues
- [ ] Implementar POST /venues/search
- [ ] Implementar GET /venues/<id>
- [ ] Implementar POST /venues/create
- [ ] Implementar POST /venues/<id>/edit
- [ ] Testes de integração

Dia 6: Blueprints de Artists e Shows

- [ ] Criar artists blueprint
- [ ] Implementar endpoints de Artists
- [ ] Criar shows blueprint
- [ ] Implementar endpoints de Shows
- [ ] Testes de integração completos

FASE 4: Frontend Integration (Dia 7)

Dia 7: Templates e Forms

- [ ] Atualizar templates com dados reais
- [ ] Integrar WTForms com Pydantic

- [ ] Implementar flash messages
- [ ] Testar fluxo completo de criação
- [ ] Ajustar CSS/JS se necessário

FASE 5: Qualidade e Deploy (Dias 8-9)

Dia 8: Testes e Qualidade

- [ ] Completar cobertura de testes (>80%)
- [ ] Rodar linters (black, mypy, flake8)
- [ ] Corrigir todos os warnings
- [ ] Documentar código faltante
- [ ] Review de segurança

Dia 9: Deploy e Documentação

- [ ] Preparar para deploy (Heroku/Railway)
- [ ] Configurar variáveis de ambiente
- [ ] Criar scripts de seed para produção
- [ ] Escrever README completo
- [ ] Criar documentação de API

FASE 6: Features Avançadas (Opcional - Dia 10+)

Recursos Stand Out

- [ ] Artist availability system
- [ ] Recent listings na homepage
- [ ] Search por cidade/estado
- [ ] Sistema de favoritos
- [ ] Notificações por email
- [ ] Dashboard administrativo

---

17. Checklist de Aceitação

✅ Requisitos Obrigatórios

Banco de Dados

- [ ] Conexão PostgreSQL funcional
- [ ] Models completos e normalizados (3NF)
- [ ] Relationships corretos entre Venue-Show-Artist
- [ ] Migrations funcionando (`flask db migrate/upgrade`)
- [ ] Constraints e indexes implementados
- [ ] Timestamps automáticos

CRUD Completo

- [ ] **Venues**:
  - [ ] Criar novo venue
  - [ ] Listar venues por cidade/estado

    - [ ] Buscar venue por ID
    - [ ] Editar venue existente
    - [ ] Deletar venue (com validações)
    - [ ] Busca parcial case-insensitive
- [ ] **Artists**:
    - [ ] Criar novo artist
    - [ ] Listar artists
    - [ ] Buscar artist por ID
    - [ ] Editar artist
    - [ ] Busca de artists
- [ ] **Shows**:
    - [ ] Criar novo show
    - [ ] Listar todos os shows
    - [ ] Separar shows passados/futuros
    - [ ] Validar conflitos de horário

Validação

- [ ] WTForms validando no frontend
- [ ] Pydantic validando no backend
- [ ] Constraints do banco funcionando
- [ ] Mensagens de erro claras
- [ ] Tratamento de duplicatas
- [ ] Validação de estados US
- [ ] Validação de formatos (phone, URL)

Busca

- [ ] Busca parcial (substring matching)
- [ ] Case-insensitive
- [ ] Busca de venues retorna upcoming shows
- [ ] Busca de artists retorna informações completas

UI/UX

- [ ] Venues agrupados por cidade/estado
- [ ] Página de venue mostra past/upcoming shows
- [ ] Página de artist mostra past/upcoming shows
- [ ] Flash messages para feedback
- [ ] Validação de formulários no client-side
- [ ] Error pages 404/500 customizadas

Código

- [ ] Separação em camadas (models, services, controllers)
- [ ] Code documentation (docstrings)
- [ ] Type hints onde aplicável
- [ ] PEP 8 compliance
- [ ] Logs configurados
- [ ] Exception handling robusto

Testes

- [ ] Testes unitários (models, schemas, services)
- [ ] Testes de integração (endpoints)
- [ ] Cobertura > 70%
- [ ] Fixtures para dados de teste
- [ ] Testes passando no CI

Segurança

- [ ] CSRF protection ativo
- [ ] SQL injection prevention
- [ ] XSS protection
- [ ] Secrets em variáveis de ambiente
- [ ] Session security configurado
- [ ] Input sanitization

🌟 Recursos Stand Out (Opcional)

- [ ] Artist availability system
- [ ] Recent artists/venues na homepage
- [ ] Search por cidade e estado
- [ ] API REST com JSON responses
- [ ] Swagger documentation
- [ ] Rate limiting
- [ ] Caching de queries
- [ ] Admin dashboard
- [ ] Email notifications
- [ ] User authentication
- [ ] Favorites system
- [ ] Reviews and ratings

---

18. Recursos Avançados (Stand Out)

18.1 Artist Availability System

python

```python
# models/availability.py
class ArtistAvailability(db.Model):
    """Disponibilidade de artistas"""
    __tablename__ = 'artist_availability'

    id = db.Column(db.Integer, primary_key=True)
    artist_id = db.Column(db.Integer, db.ForeignKey('artists.id'))
    day_of_week = db.Column(db.Integer)  # 0-6 (Monday-Sunday)
    start_time = db.Column(db.Time)
    end_time = db.Column(db.Time)
    available = db.Column(db.Boolean, default=True)

    artist = db.relationship('Artist', back_populates='availability')

# services/show_service.py
def validate_show_time(self, artist_id: int, show_time: datetime) -> bool:
    """Valida se artista está disponível no horário"""
    day = show_time.weekday()
    time = show_time.time()

    availability = ArtistAvailability.query.filter_by(
        artist_id=artist_id,
        day_of_week=day
    ).first()

    if not availability:
        return False

    return (
        availability.available and
        availability.start_time <= time <= availability.end_time
    )
```

18.2 Recent Listings

python

```python
# services/venue_service.py
def get_recent_venues(self, limit: int = 10) -> List[VenueResponse]:
    """Retorna venues recentes"""
    venues = (
        Venue.query
        .order_by(Venue.created_at.desc())
        .limit(limit)
        .all()
    )
    return [self._to_response(v) for v in venues]

# controllers/main.py
@main_bp.route('/')
def index():
    """Homepage com recent listings"""
    venue_service = VenueService(db.session)
    artist_service = ArtistService(db.session)

    recent_venues = venue_service.get_recent_venues(10)
    recent_artists = artist_service.get_recent_artists(10)

    return render_template(
        'pages/home.html',
        recent_venues=recent_venues,
        recent_artists=recent_artists
    )
```

18.3 Advanced Search

python

```python
# schemas/search_schema.py
class SearchParams(BaseModel):
    """Parâmetros de busca avançada"""
    query: Optional[str] = None
    city: Optional[str] = None
    state: Optional[str] = None
    genres: Optional[List[str]] = None
    seeking: Optional[bool] = None

# repositories/venue_repository.py
def advanced_search(self, params: SearchParams) -> List[Venue]:
    """Busca avançada com múltiplos filtros"""
    query = self.session.query(Venue)

    if params.query:
        query = query.filter(
            Venue.name.ilike(f'%{params.query}%')
        )

    if params.city:
        query = query.filter(Venue.city == params.city)

    if params.state:
        query = query.filter(Venue.state == params.state)

    if params.seeking is not None:
        query = query.filter(Venue.seeking_talent == params.seeking)

    return query.all()
```

---

19. Deploy & CI/CD

19.1 Heroku Deploy

bash

```
# Procfile
web: gunicorn app:app

# runtime.txt
python-3.11.0

# Adicionar gunicorn
uv add gunicorn psycopg2-binary

# Heroku CLI
heroku create fyyur-app
heroku addons:create heroku-postgresql:hobby-dev
heroku config:set SECRET_KEY=your-secret-key
heroku config:set FLASK_ENV=production

# Deploy
git push heroku main

# Migrations
heroku run flask db upgrade
heroku run python scripts/seed.py
```

19.2 GitHub Actions

yaml

```yaml
# .github/workflows/test.yml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_PASSWORD: postgres
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.11

    - name: Install UV
      run: pip install uv

    - name: Install dependencies
      run: |
        uv venv
        source .venv/bin/activate
        uv add -r requirements.txt
        uv add --dev pytest pytest-cov

    - name: Run tests
      run: |
        source .venv/bin/activate
        pytest --cov=app --cov-report=xml
```

```yaml
    - name: Upload coverage
      uses: codecov/codecov-action@v2
```

19.3 Docker (Opcional)

dockerfile

```dockerfile
# Dockerfile
FROM python:3.11-slim

WORKDIR /app

# Install UV
RUN pip install uv

# Copy requirements
COPY requirements.txt .
RUN uv pip install --system -r requirements.txt

# Copy app
COPY . .

# Run migrations
RUN flask db upgrade

EXPOSE 5000

CMD ["gunicorn", "-b", "0.0.0.0:5000", "app:app"]
```

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: fyyur
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data

  web:
    build: .
    ports:
      - "5000:5000"
    environment:
      DATABASE_URL: postgresql://postgres:postgres@db:5432/fyyur
      FLASK_ENV: development
    depends_on:
      - db
    volumes:
      - .:/app

volumes:
  postgres_data:
```

---

20. Anexos: Código de Referência

20.1 Model Completo

python

```python
# models/venue.py
from datetime import datetime
from app.extensions import db

venue_genres = db.Table(
    'venue_genres',
    db.Column('venue_id', db.Integer, db.ForeignKey('venues.id'), primary_key=True),
    db.Column('genre', db.String(50), primary_key=True)
)

class Venue(db.Model):
    """Modelo de Venue"""
    __tablename__ = 'venues'

    # Primary Key
    id = db.Column(db.Integer, primary_key=True)

    # Basic Info
    name = db.Column(db.String(255), nullable=False)
    city = db.Column(db.String(120), nullable=False)
    state = db.Column(db.String(2), nullable=False)
    address = db.Column(db.String(120), nullable=False)
    phone = db.Column(db.String(120))

    # Links
    image_link = db.Column(db.String(500))
    facebook_link = db.Column(db.String(200))
    website_link = db.Column(db.String(200))

    # Seeking
    seeking_talent = db.Column(db.Boolean, default=False, nullable=False)
    seeking_description = db.Column(db.Text)

    # Timestamps
    created_at = db.Column(db.DateTime, default=datetime.utcnow, nullable=False)
    updated_at = db.Column(db.DateTime, onupdate=datetime.utcnow)

    # Relationships
    shows = db.relationship(
        'Show',
        back_populates='venue',
        cascade='all, delete-orphan',
        lazy='dynamic'
    )
```

```
    # Genres (many-to-many via association table)
    genres = db.relationship(
        'Genre',
        secondary=venue_genres,
        lazy='subquery',
        backref=db.backref('venues', lazy=True)
    )

    # Constraints
    __table_args__ = (
        db.UniqueConstraint('name', 'city', name='uq_venue_name_city'),
        db.CheckConstraint("length(state) = 2", name='ck_venue_state_length'),
        db.Index('idx_venue_city_state', 'city', 'state'),
        db.Index('idx_venue_name', 'name'),
    )

    def __repr__(self):
        return f'<Venue {self.id}: {self.name}>'

    def to_dict(self):
        """Serializa para dicionário"""
        return {
            'id': self.id,
            'name': self.name,
            'city': self.city,
            'state': self.state,
            'address': self.address,
            'phone': self.phone,
            'image_link': self.image_link,
            'facebook_link': self.facebook_link,
            'website_link': self.website_link,
            'seeking_talent': self.seeking_talent,
            'seeking_description': self.seeking_description,
            'created_at': self.created_at.isoformat() if self.created_at else None
        }
```

📚 Conclusão

Este plano fornece um guia completo e profissional para implementar o backend do projeto Fyyur. Seguindo as fases e práticas descritas, você criará uma aplicação robusta, testável e escalável.

Principais Takeaways

1. **Separação de Responsabilidades**: Mantenha cada camada independente
2. **Validação em Múltiplas Camadas**: WTForms + Pydantic + Database
3. **Testes São Essenciais**: Não pule os testes!
4. **Documentação**: Code autodocumentado + docstrings + README
5. **Segurança First**: Sempre valide e sanitize inputs
6. **Performance Matters**: Use indexes, eager loading, pagination

Próximos Passos

1. Começar pela Fase 1 (Setup)
2. Seguir o roadmap sequencialmente
3. Fazer commits frequentes
4. Rodar testes a cada feature
5. Deploy incremental

Recursos Úteis

- Flask Documentation
- SQLAlchemy Docs
- Pydantic Documentation
- UV Documentation
- pytest Documentation

Boa sorte com a implementação! 🚀