Patterns aplicados na camada de persistência

1. Repository / DAO

Nome: Repository (Repository/DAO) Local: arquivos em `persistence/*_repository.py` (ex.: `user_repository.py`) Trecho de código (exemplo):

```
class UserRepository:
    def create(self, id, name, email, is_admin=False,
created_at=None):
        with get_connection() as conn:
            conn.execute('INSERT INTO users
(id,name,email,is_admin,created_at) VALUES (?,?,?,?,?)',
                         (id, name, email, int(is_admin),
created_at))
```

Justificativa: separa a lógica de acesso a dados da lógica de domínio e permite trocar a fonte de dados sem afetar serviços ou modelos. Facilita testes (mock/spy) e centraliza queries SQL.

2. Connection Factory (variante de Factory / Singleton)

Nome: Factory / Singleton (connection factory) Local: `persistence/connection_factory.py` Trecho de código (exemplo):

```
def get_connection():
    conn = sqlite3.connect(str(DB_PATH))
    conn.row_factory = sqlite3.Row
    conn.execute('PRAGMA foreign_keys = ON;')
    return conn
```

Justificativa: centraliza a configuração de conexões, garante PRAGMA de foreign keys e facilita migração para outro backend (Postgres, MySQL) apenas alterando a fábrica.

3. Transaction scope (context manager usage)

Uso de `with get_connection() as conn:` para delimitar o escopo da transação e garantir `commit/rollback`. Benefício: aumenta segurança e consistência nas operações de escrita.

Conclusão

A camada de persistência usa padrões simples e testáveis (Repository/DAO e Factory), favorecendo o desacoplamento entre domínio e armazenamento e facilitando futuros upgrades (ORM, replica sets, cache).