Naming Conventions

Mapa Completo de Case Conventions

TypeScript/JavaScript/React

PascalCase

- Classes: `UserService`, `PaymentProcessor`
- Interfaces: `IUserRepository`, `PaymentGateway`
- Types: `UserType`, `PaymentStatus`
- Enums: `OrderStatus`, `UserRole`
- Componentes React: `UserProfile`, `PaymentForm`
- Páginas Next.js: `UserDashboard`, `CheckoutPage`

camelCase

- Variáveis: `userName`, `totalAmount`
- Funções: `calculateTotal`, `validateEmail`
- Métodos: `getUserById`, `processPayment`
- Props: `isLoading`, `onSubmit`
- Hooks personalizados: `useUserData`, `usePaymentForm`

UPPER_SNAKE_CASE

- Constantes: `MAX_RETRY_ATTEMPTS`, `DEFAULT_TIMEOUT`
- Environment Variables: `DATABASE_URL`, `API_KEY`
- Enum Values: `ORDER_STATUS.PENDING`, `USER_ROLE.ADMIN`

kebab-case

- Arquivos: `user-service.ts`, `payment-form.tsx`
- URLs: `/user-profile`, `/payment-history`
- CSS Classes: `.user-card`, `.payment-button`
- Diretórios: `user-management`, `payment-processing`
- Package names: `@company/user-utils`

Python

PascalCase

- Classes: `UserService`, `PaymentProcessor`
- Exceptions: `UserNotFoundError`, `PaymentFailedError`
- Type Annotations: `UserType`, `PaymentData`

snake_case

- Variáveis: `user_name`, `total_amount`
- Funções: `calculate_total`, `validate_email`
- Métodos: `get_user_by_id`, `process_payment`
- Módulos: `user_service`, `payment_utils`
- Arquivos: `user_service.py`, `payment_processor.py`
- Database columns: `created_at`, `user_id`

UPPER_SNAKE_CASE

- Constantes: `MAX_RETRY_ATTEMPTS`, `DEFAULT_TIMEOUT`
- Settings: `DATABASE_URL`, `SECRET_KEY`

Prefixos Especiais

- `_private`: Métodos/variáveis privadas com `_` prefix
- `__dunder__`: Métodos mágicos do Python (`__init__`, `__str__`)

Domain-Driven Design

Commands

- **Format**: PascalCase + Command suffix
- **Naming**: Verbo imperativo + substantivo + Command
- **Examples**:
- `CreateUserCommand`
- `UpdateOrderCommand`
- `ProcessPaymentCommand`

Events

- **Format**: PascalCase + Event suffix
- **Naming**: Substantivo + verbo passado + Event
- **Examples**:
- `UserCreatedEvent`
- `OrderCompletedEvent`
- `PaymentProcessedEvent`

Queries

- **Format**: PascalCase + Query suffix
- **Naming**: Get/Find + critério + Query
- **Examples**:
- `GetUserByIdQuery`
- `FindActiveOrdersQuery`
- `SearchProductsQuery`

Value Objects

- **Format**: PascalCase descritivo
- **Examples**:
- `EmailAddress`
- `Money`
- `UserId`
- `OrderNumber`

Aggregates

- **Format**: PascalCase singular
- **Examples**:
- `User`
- `Order`
- `Product`
- `Payment`

Handlers

- **Format**: PascalCase + Handler suffix
- **Examples**:
- `CreateUserCommandHandler`
- `GetUserByIdQueryHandler`
- `UserCreatedEventHandler`

Padrões por Contexto

Database

Tables

- **Format**: snake_case plural
- **Examples**: `users`, `orders`, `payment_transactions`

Columns

- **Format**: snake_case
- **Examples**: `user_id`, `created_at`, `total_amount`

Indexes

- **Format**: `idx_table_column(s)`
- **Examples**: `idx_users_email`, `idx_orders_user_id_status`

Foreign Keys

- **Format**: `fk_table_referenced_table`
- **Examples**: `fk_orders_users`, `fk_order_items_products`

APIs

Endpoints

- **Format**: kebab-case
- **Examples**: `/api/users`, `/api/payment-methods`, `/api/order-history`

Query Parameters

- **Format**: snake_case
- **Examples**: `?user_id=123`, `?created_after=2023-01-01`

Headers

- **Format**: Kebab-Case (HTTP standard)
- **Examples**: `Content-Type`, `Authorization`, `X-Request-ID`

Files and Directories

Configuration Files

- **Format**: kebab-case ou snake_case
- **Examples**:
- `docker-compose.yml`
- `pyproject.toml`
- `.env.example`

Test Files

- **Format**: Mirror source + test suffix

- **Examples**:
- `user-service.test.ts`
- `payment_processor_test.py`
- `user-component.spec.tsx`

Documentation

- **Format**: kebab-case
- **Examples**:
- `api-documentation.md`
- `deployment-guide.md`
- `troubleshooting.md`

Validação e Consistência

Regras de Validação

1. **Consistência dentro do contexto**: Mesmo padrão dentro de um arquivo/módulo
2. **Clareza sobre brevidade**: Nomes descritivos > nomes curtos
3. **Evitar abreviações**: `user` ao invés de `usr`, `calculate` ao invés de `calc`
4. **Contexto específico**: `UserService` ao invés de `Service`

Anti-Padrões

- ❌ Misturar convenções: `userName` e `user_name` no mesmo contexto
- ❌ Abreviações obscuras: `usr`, `calc`, `proc`
- ❌ Nomes genéricos: `data`, `info`, `manager`
- ❌ Números em nomes: `user1`, `service2`
- ❌ Prefixos desnecessários: `strUserName`, `intUserId`

Ferramentas de Validação

- **ESLint**: Regras de naming para TypeScript/JavaScript
- **Ruff**: Naming conventions para Python
- **SonarQube**: Análise de qualidade de nomes
- **Custom Linters**: Regras específicas do projeto

Exemplos Práticos

TypeScript/React

```typescript
// ✅ Good
interface UserProfile {
  id: number;
  emailAddress: string;
  displayName: string;
}

class UserService {
  async getUserById(userId: number): Promise<UserProfile | null> {
    // Implementation
  }
}

const UserProfileComponent: React.FC<UserProfileProps> = ({ userId }) => {
  const { userData, isLoading } = useUserProfile(userId);
  // Implementation
};

// ❌ Bad
interface user_profile {
  ID: number;
  email: string;
  name: string;
}

class userSvc {
  async getUsrById(id: number): Promise<user_profile | null> {
    // Implementation
  }
}
```

Python

```python
# ✅ Good
class UserService:
    def __init__(self, user_repository: IUserRepository):
        self._user_repository = user_repository

    async def get_user_by_id(self, user_id: int) -> Optional[User]:
        return await self._user_repository.get_by_id(user_id)

# ❌ Bad
class UserSvc:
    def __init__(self, userRepo: IUserRepository):
        self.userRepo = userRepo

    async def getUsrById(self, ID: int) -> Optional[User]:
        return await self.userRepo.getById(ID)
```

Referências

- @ref:global-standards#solid-principles
- @ref:error-handling#naming-patterns

- @docs:https://peps.python.org/pep-0008/
- @docs:https://google.github.io/styleguide/tsguide.html