Hướng Dẫn Hiểu Dự Án 1Test Question Service

📋 Tổng Quan Dự Án

1Test Question Service là một microservice được xây dựng bằng **gRPC** để quản lý câu hỏi và danh mục câu hỏi. Dự án này được thiết kế theo kiến trúc **Clean Architecture** với các lớp phân tách rõ ràng.

🏗️ Kiến Trúc Tổng Thể

1. Công Nghệ Sử Dụng

- **gRPC**: Protocol buffer cho API communication
- **TypeScript**: Ngôn ngữ chính
- **PostgreSQL**: Database chính (sử dụng Prisma ORM)
- **DynamoDB**: NoSQL database (cho một số tính năng)
- **AWS Services**: S3, EventBridge, CloudFront
- **pnpm**: Package manager
- **Jest**: Testing framework

2. Cấu Trúc Monorepo

```
1test-question-service/
├── app/                    # Ứng dụng chính
├── proto/                  # Protocol Buffer definitions
├── infra/                  # Infrastructure as Code (CDK)
├── docker/                 # Docker configuration
└── docs/                   # Documentation
```

🔧 Cấu Trúc Chi Tiết

📁 `/app/src/` - Ứng Dụng Chính

1. Entry Point (`index.ts`)

- Khởi tạo gRPC server
- Load proto files
- Khởi tạo AWS services
- Register các handlers

2. Handlers (`handlers/`)

```
handlers/
├── services.ts             # Export tất cả proto packages
└── v1/
    ├── index.ts            # Export v1 packages
```

```
    └── category/
        ├── handler.ts      # CategoryHandler implementation
        └── mapper.ts       # Proto ↔ Domain mapping
```

Luồng hoạt động của Handler:

1. Nhận gRPC request
2. Validate và map request data
3. Gọi Use Case tương ứng
4. Map response về proto format
5. Trả về gRPC response

3. Core Business Logic (`core/`)

Domain Layer (`core/business/domain/`)

- **Models**: Định nghĩa business entities
- **Repositories**: Interfaces cho data access

Use Cases (`core/business/usecases/`)

- **CategoryUseCases**:
  - `CreateCategoryUseCase`
  - `UpdateCategoryUseCase`
  - `DeleteCategoryUseCase`
  - `DeleteCategoriesUseCase`
  - `ListCategoriesUseCase`

Luồng Use Case:

1. Nhận input từ handler
2. Validate business rules
3. Gọi repository để thao tác data
4. Trả về domain model

Infrastructure (`core/infrastructure/`)

```
infrastructure/
├── data/
│   ├── dynamodb/           # DynamoDB repositories
│   └── postgresql/         # PostgreSQL repositories + Prisma
└── services/
    ├── eventbridge/        # AWS EventBridge client
    └── s3/                 # AWS S3 client
```

4. Registry Pattern (`registry.ts`)

- **Dependency Injection Container**
- Quản lý singleton instances
- Khởi tạo repositories và services

```
class ServiceRegistry {
  // Singletons
  private _categoryRepository: ICategoryRepository;

  initialize() {
    // Khởi tạo datasources
    const postgresqlDataSource = new PostgresqlDataSource();

    // Khởi tạo repositories
    this._categoryRepository = new CategoryRepository(postgresqlDataSource);
  }
}
```

5. Configuration (`config.ts`)

- Load environment variables
- Định nghĩa proto files paths
- AWS configuration
- Server settings

6. Utilities (`core/utils/`)

- **AWS Credentials Manager**: Quản lý AWS authentication
- **gRPC Error Mapper**: Convert errors sang gRPC format
- **Validation utilities**

🗀 `/proto/` - Protocol Buffer Definitions

```
proto/
├── common/
│   └── common.proto        # Shared messages (ErrorResponse, etc.)
└── services/
    └── question/v1/
        ├── category.proto  # Category service definition
        └── question.proto  # Question service definition
```

Ví dụ Category Service:

```
service CategoryService {
  rpc CreateCategory(CreateCategoryRequest) returns (CreateCategoryResponse);
  rpc UpdateCategory(UpdateCategoryRequest) returns (UpdateCategoryResponse);
  rpc DeleteCategory(DeleteCategoryRequest) returns (DeleteCategoryResponse);
```

```
  rpc ListCategories(ListCategoriesRequest) returns (ListCategoriesResponse);
}
```

📁 `/infra/` - Infrastructure as Code

Sử dụng AWS CDK để định nghĩa infrastructure:

- ECR repositories
- ECS services
- Load balancers
- Database configurations

🔄 Luồng Hoạt Động

1. Request Processing Flow

```
gRPC Client → Handler → UseCase → Repository → Database
                ↓
            Response ← Mapper ← Domain Model ← Database
```

2. Chi Tiết Luồng Xử Lý CreateCategory

1. **Client** gửi `CreateCategoryRequest` qua gRPC
2. **CategoryHandler** nhận request
3. **ProtoMapper** convert request thành domain input
4. **CreateCategoryUseCase** xử lý business logic
5. **CategoryRepository** lưu vào PostgreSQL
6. **Domain Model** được trả về
7. **ProtoMapper** convert domain model thành proto response
8. **Handler** trả response về client

3. Dependency Flow

```
Handler → UseCase → Repository → DataSource → Database
   ↑         ↑          ↑           ↑
Registry.getInstance() provides all dependencies
```

🗄️ Database Design

PostgreSQL (Primary)

- **Categories table**: Quản lý danh mục câu hỏi
- **Questions table**: Quản lý câu hỏi
- **Relationships**: Category có thể có parent-child relationship

DynamoDB (Secondary)

- Có thể được sử dụng cho caching hoặc specific use cases
- Event sourcing hoặc audit logs

🚀 Cách Chạy Dự Án

1. Setup

```
# Clone proto definitions
git clone https://github.com/TE-System/1test-service-proto-definition proto

# Install dependencies
pnpm install -r

# Generate TypeScript từ proto files
pnpm gen:proto
```

2. Development

```
# Chạy development mode (auto-restart)
pnpm dev:app

# Hoặc chạy normal mode
pnpm start:app
```

3. Build & Production

```
# Build application
pnpm build:app

# Run production
pnpm start:prod
```

🧪 Testing

```
# Chạy tests
pnpm test:app

# Test với coverage
pnpm test:cov

# Watch mode
pnpm test:watch
```

📦 Package Management

Dự án sử dụng **pnpm workspace** với:

- **Root package.json**: Scripts để manage toàn bộ workspace
- **App package.json**: Dependencies cho application
- **Infra package.json**: Dependencies cho infrastructure

🔧 Development Workflow

1. Thêm Feature Mới

1. Định nghĩa proto message/service trong `/proto/`
2. Generate TypeScript code: `pnpm gen:proto`
3. Tạo domain model trong `/core/business/domain/models/`
4. Tạo repository interface trong `/core/business/domain/repositories/`
5. Implement repository trong `/core/infrastructure/data/`
6. Tạo use case trong `/core/business/usecases/`
7. Tạo handler trong `/handlers/v1/`
8. Register trong registry
9. Viết tests

2. Database Changes

1. Cập nhật Prisma schema
2. Chạy migration: `pnpm prisma:migrate:app`
3. Generate Prisma client: `pnpm prisma:generate:app`

🎯 Điểm Mạnh Của Kiến Trúc

1. **Clean Architecture**

- **Separation of Concerns**: Mỗi layer có responsibility riêng biệt
- **Dependency Inversion**: Business logic không phụ thuộc vào infrastructure
- **Testability**: Dễ dàng mock và test từng layer

2. **gRPC Benefits**

- **Performance**: Binary protocol, HTTP/2
- **Type Safety**: Strong typing với protobuf
- **Language Agnostic**: Có thể generate code cho nhiều ngôn ngữ
- **Streaming**: Support real-time communication

3. **Registry Pattern**

- **Dependency Injection**: Centralized dependency management
- **Singleton Management**: Đảm bảo single instance cho shared resources
- **Easy Testing**: Có thể inject mock dependencies

4. **Error Handling**

- **Centralized Error Mapping**: Convert domain errors sang gRPC errors
- **Consistent Error Format**: Unified error response structure

🔍 Debugging Tips

1. gRPC Debugging

- Sử dụng gRPC reflection để inspect services
- Tools như BloomRPC hoặc Postman để test gRPC endpoints

2. Database Debugging

- Prisma Studio: `npx prisma studio`
- Check database logs
- Monitor query performance

3. AWS Services

- CloudWatch logs
- X-Ray tracing
- AWS CLI tools

📈 Scaling Considerations

1. **Horizontal Scaling**

- Stateless design cho phép scale containers
- Database connection pooling
- Load balancing với multiple instances

2. **Performance**

- Database indexing
- Caching strategies
- Connection pooling
- Query optimization

3. **Monitoring**

- Health checks
- Metrics collection
- Distributed tracing
- Error monitoring

🔐 Security

1. **Authentication**

- AWS Roles Anywhere cho service authentication

- Certificate-based authentication

2. **Authorization**

- Role-based access control
- API rate limiting
- Input validation

3. **Data Protection**

- Encryption at rest và in transit
- Secure credential management
- Audit logging

🎯 Best Practices

1. **Code Organization**

- Follow Clean Architecture principles
- Use TypeScript strict mode
- Implement proper error handling
- Write comprehensive tests

2. **gRPC Design**

- Design proto messages carefully
- Use proper versioning (v1, v2...)
- Handle backward compatibility
- Document API changes

3. **Database Design**

- Normalize data appropriately
- Use indexes wisely
- Handle migrations safely
- Monitor performance

Tài liệu này sẽ được cập nhật theo sự phát triển của dự án. Nếu có thắc mắc hoặc cần bổ sung, vui lòng tạo issue hoặc liên hệ team.