# مستند فنی پروژه SmartRecommender (نسخه فارسی نهایی)

## معرفی کلی

SmartRecommender یک سیستم پیشنهاددهی هوشمند پیشرفته است که به طور خاص برای درک عمیق نیازهای کاربران از طریق تحلیل نیت کاربر (Intent Extraction) طراحی شده است. این پروژه با پیروی دقیق از اصول Clean Architecture و Domain-Driven Design (DDD) توسعه یافته است. هدف اصلی این رویکرد، تضمین انعطاف‌پذیری، قابلیت نگهداری و مقیاس‌پذیری سیستم از طریق جداسازی کامل دغدغه‌های مختلف است.

جداسازی لایه‌ها امکان می‌دهد تا منطق هسته کسب‌وکار (Domain) کاملاً از جزئیات پیاده‌سازی لایه‌های زیرین مانند پایگاه داده (EF Core) و سرویس‌های خارجی (مانند موتورهای NLP برای هوش مصنوعی) مستقل باقی بماند. این امر به ویژه در پروژه‌هایی که بخش‌های هوش مصنوعی نقش محوری دارند، حیاتی است.

---

## ساختار کلی پروژه

پروژه SmartRecommender به پنج لایه اصلی تقسیم می‌شود که هر کدام دارای مسئولیت مشخص و جهت وابستگی کنترل‌شده‌ای هستند (وابستگی‌ها فقط از لایه‌های خارجی به داخلی مجاز هستند).

| لایه | وظیفه اصلی | وابستگی | توضیحات کلیدی |
| --- | --- | --- | --- |
| Domain | تعریف مدل‌های اصلی، قوانین کسب‌وکار، و Value Objectها. | ندارد (خالص) | قلب پروژه؛ کاملاً مستقل و بدون وابستگی به فریمورک‌ها. |
| Application | تعریف UseCaseها، مدیریت جریان داده‌ها، و تعریف قراردادهای (Interfaces) دسترسی به زیرساخت. | Domain، AI (از طریق Interface) | Orchestration منطق کسب‌وکار و تعریف UseCaseها. |
| Infrastructure | | Application، Domain | شامل DbContext، Migrations و |

| لایه | وظیفه اصلی | وابستگی | توضیحات کلیدی |
| --- | --- | --- | --- |
| Infrastructure | پیاده‌سازی جزئیات دسترسی به داده (EF Core) و پیاده‌سازی Repositoryها. | Domain | پیاده‌سازی‌های واقعی Repositoryها. |
| AI | انجام پردازش زبان طبیعی (NLP)، استخراج نیت کاربر (Intent)، و فیلترها. | Domain | شامل مدل‌های یادگیری ماشین و سرویس‌های تحلیل متن. |
| API | لایه ارائه (Presentation Layer) شامل Web API، کنترلرها، و DTOها برای تعامل با دنیای خارج. | Application، AI، Infrastructure | نقطه ورودی و خروجی سیستم؛ مسئول سریال‌سازی و اعتبارسنجی اولیه. |

---

# 🏗️ معماری و الگوی طراحی

## Clean Architecture (معماری تمیز)

اصل اساسی در این پروژه، پیروی از قانون وابستگی است: قانون وابستگی (Dependency Rule) حکم می‌کند که وابستگی‌ها باید همیشه به سمت مرکز (Domain) باشند.

- Domain (مرکز): بالاترین سطح انتزاع و استقلال.
- Application: وابسته به Domain و AI (از طریق Interface).
- Infrastructure: وابسته به Application و Domain برای پیاده‌سازی انتزاعات.
- API (بیرون): وابسته به Application، AI و Infrastructure برای هماهنگی جریان.

این ساختار تضمین می‌کند که تغییر در EF Core (Infrastructure) یا مدل‌های NLP (AI) نمی‌تواند منطق اصلی کسب‌وکار (Domain) را تحت تأثیر قرار دهد.

## Domain Layer

این لایه مدل‌سازی خالص از هسته کسب‌وکار است.

پوشه‌ها: `Entities` , `ValueObjects` , `DomainEvents`

- Entities: شامل موجودیت‌های اصلی مانند `Product` , `User` , `Session` . این‌ها دارای هویت (ID) هستند و تغییرات وضعیت آن‌ها توسط قوانین Domain مدیریت می‌شود.

- Value Objects: مانند `UserIntent` (که شامل نوع نیت و پارامترهای آن است) و `IntentFilters` (که حاوی فیلترهای استخراج شده مانند "رنگ: آبی"، "برند: سامسونگ" است). این‌ها دارای هویت نیستند و بر اساس مقادیرشان مقایسه می‌شوند.

## Application Layer

این لایه "چسب" منطق دامنه و زیرساخت است و از طریق واسط‌ها (Interfaces) با لایه‌های پایین‌تر ارتباط برقرار می‌کند.

پوشه‌ها: `Abstractions` , `DTOs` , `UseCases` , `Services`

1. Abstractions: شامل `IProductRepository` , `IIntentExtractorService` . این‌ها واسط‌هایی هستند که لایه Infrastructure آن‌ها را پیاده‌سازی می‌کند.
2. UseCases (Command/Query Handlers): نمایانگر عملیات‌های اصلی سیستم (مثلاً `GetProductRecommendationsUseCase` ).
3. Services: شامل سرویس‌هایی که عملیات‌های پیچیده چندمرحله‌ای را هماهنگ می‌کنند، مانند `ChatService` که نیت کاربر را گرفته و جریان پیشنهادسازی را مدیریت می‌کند.

## Infrastructure Layer

این لایه جزئیات فنی مربوط به ذخیره‌سازی و دسترسی به منابع خارجی را پنهان می‌کند.

پوشه‌ها: `Context` , `Repositories` , `DataServices`

- Context: شامل `AppDbContext` (پیاده‌سازی EF Core) و تنظیمات مدل‌سازی (Fluent API).
- Repositories: پیاده‌سازی‌های عینی از واسط‌های تعریف شده در لایه Application (مثلاً `EfProductRepository` که از `IProductRepository` ارث می‌برد).

## AI Layer

این لایه متمرکز بر وظایف هوش مصنوعی است، به ویژه تحلیل متن کاربر.

پوشه‌ها: `Interfaces` , `Services` , `Models`

- IntentExtractor: هسته پردازش NLP که ورودی متنی کاربر را دریافت کرده و آن را به یک ساختار داده‌ای قابل استفاده (معمولاً یک `UserIntent` Value Object) تبدیل می‌کند.
- IntentNormalizer: وظیفه نرمال‌سازی و استانداردسازی عبارات استخراج شده (مثلاً تبدیل "گوشی موبایل" به "ProductType:Phone").

این لایه با Domain در تعامل است زیرا نتایج آن باید در قالب Value Objectهای Domain (مثل `IntentFilters` ) ارائه شوند.

## API Layer

این لایه دروازه ورودی خارجی سیستم است و وظیفه تبدیل درخواست‌های HTTP به پیام‌های داخلی و بالعکس را بر عهده دارد.

پوشه‌ها: `Controllers` , `Mapping` , `Configuration`

- Controllers: مانند `ChatController` که درخواست‌های کاربر را دریافت کرده و آن‌ها را به UseCaseهای لایه Application ارسال می‌کند.
- DTOs (Data Transfer Objects): مدل‌های سبک وزن برای تبادل داده بین لایه API و لایه‌های داخلی.

---

# جریان داده (Data Flow Example: User Query)

یک مثال از نحوه پردازش یک درخواست کاربر در سیستم:

1. کاربر: متنی را وارد می‌کند (مثلاً: "بهترین گوشی سامسونگ با قیمت زیر 20 میلیون").
2. API (ChatController): درخواست HTTP را دریافت کرده و DTO ورودی را به تابع مربوطه در `ChatService` (UseCase) پاس می‌دهد.
3. Application (ChatService):
   - متن خام را به `AI.IntentExtractor` ارسال می‌کند.
4. AI Layer (IntentExtractor):
   - متن را تحلیل کرده و یک `UserIntent` شامل نیت اصلی (جستجوی محصول) و فیلترهای استخراج شده (Brand=Samsung, MaxPrice=20000000) تولید می‌کند.

5. Application (ChatService):
   - از `IProductRepository` (که یک واسط است) برای اجرای جستجو استفاده می‌کند و فیلترهای استخراج شده را به آن می‌دهد.
6. Infrastructure (EfProductRepository):
   - واسط را پیاده‌سازی کرده و با استفاده از EF Core، کوئری بهینه شده‌ای را بر روی `AppDbContext` اجرا می‌کند.
   - فرمول جستجوی اصلی در اینجا به صورت مفهومی می‌تواند به شکل زیر باشد:

$$\text{Query} = \{P \in \text{Products} \mid P.\text{Category} = \text{Phone} \land P.\text{Brand} = \text{'Samsung'} \land P.\text{Price}$$

1. Infrastructure/Application: نتایج (لیستی از `Product` Entities) به لایه Application بازگردانده می‌شود.
2. API: نتایج نهایی به DTO تبدیل شده و در قالب پاسخ HTTP به کاربر ارسال می‌شود.

---

# ⚙ راه‌اندازی سریع (Quick Start)

برای شروع کار با پروژه SmartRecommender، مراحل زیر را دنبال کنید:

## 1. دریافت کد منبع

کلون کردن مخزن پروژه:

```
git clone https://github.com/<YourUser>/SmartRecommender.git
cd SmartRecommender
```

## 2. تنظیمات پایگاه داده

اطمینان حاصل کنید که سرویس پایگاه داده (پیش‌فرض: SQL Server یا PostgreSQL بسته به تنظیمات) در حال اجرا است.

فایل `appsettings.json` واقع در پوشه `SmartRecommender.API` را ویرایش کرده و `Connection String` مربوط به پایگاه داده خود را تنظیم کنید.

```
} :"ConnectionStrings"
r;Database=SmartRecommenderDB;User Id=YourUser;Password=YourPassword"
{
```

## 3. اعمال Migration اولیه

برای ایجاد ساختار جداول بر اساس مدل‌های Domain و Migration ،Infrastructure اول را اجرا کنید:

```
# مسیر اجرای دستور باید از پوشه ریشه Solution باشد
nitialCreate -p SmartRecommender.Infrastructure -s SmartRecommender.API
abase update -p SmartRecommender.Infrastructure -s SmartRecommender.API
```

## 4. اجرای پروژه

پروژه اصلی API را اجرا کنید:

```
dotnet run --project SmartRecommender.API
```

پس از اجرای موفقیت‌آمیز، API در پورت پیش‌فرض http://localhost:5000 در دسترس خواهد بود. می‌توانید با ارسال درخواست‌های POST به مسیرهای تعریف شده در `ChatController` (مثلاً `api/chat/query/` )، سیستم را آزمایش کنید.

---

# وضعیت نهایی پروژه

پروژه SmartRecommender در وضعیت پایدار و آماده برای توسعه‌های آتی قرار دارد:

- معماری نهایی: ساختار Clean Architecture و وابستگی‌های DDD به طور کامل پیاده‌سازی و تأیید شده‌اند.
- قابلیت‌های فیلترینگ: توانایی سیستم در استخراج و اعمال فیلترهای چندگانه (شامل قیمت، برند، دسته محصول و ویژگی‌های کیفی) با موفقیت تست شده است.
- مستندسازی: این سند فنی (نسخه فارسی نهایی) و مستندات مرجع انگلیسی برای راهنمای توسعه‌دهندگان در دسترس است و به زودی در GitHub منتشر خواهد شد.
- پایداری: جداسازی لایه‌ها، تست‌پذیری واحدهای (Unit Testing) لایه‌های Domain و Application را به شدت تسهیل کرده است.

---