RestFullApi

Api رابط بین نرم افزار ها است برای بهره مندی از خدمات یک دیگر.

**انواع Api :**

سخت افزاری

نرم افزاری

وب api

**انواع روش های استفاده از webApi**

**openApi** که به صورت رایگان در دسترس عموم است مانند هواشناسی

**Partner api** به صورت اشتراکی است و نیاز به پرداخت هزینه دارد مثل استعلام های مختلف

**Internal api یا private** خدماتی به خود ما می دهد مثل ماژول های مختلف سیستم که به هم api می دهد مثلا در microservice ها یا سیستم های ماژولار کاربر به فروشگاه و کاربرد فقط داخلی دارد و سیستم خودمان استفاده می شود.

**Composite api** بیشترین کاربرد را در microservice ها دارد و یک api را صدا می زنیم و آن خودش چندین endpoint را صدا می زند و خروجی نهایی را به کاربر ما می دهد.

**روش های طراحی :**

**Soap – simple object access protocol**

پروتکلی برای انتقال داده است مثل درگاه پرداخت بانک های مختلف و در .net با wcf می شود استفاده کرد.

**Rpc**

روش قدیمی است که امروزه استفاده نمی شود. به صورت مستقیم یک endpoint را در یک سیستم دیگر صدا میزده.

**Grpc - Google Remote Procedure Call**

الگو گرفته از Rpc که به صورت binary دیتا را انتقال می دهد و جدید است و سرعت بالایی دارد و توسط google ارائه شده و در .net قابل استفاده است.

**Rest**

محبوب ترین معماری در webApi ها است و پیاده سازی ساده ای دارد و تقریبا تمامی کلاینت ها می توانند از آن استفاده کنند.

**Rest**

یک معماری است که دارای یک سری قوانین و قاعده است که باید از آن ها پیروی کرد تا معماری پیاده سازی شود در web api ما دارای 6 قانون است.

* **Uniform interface**
  + Identification of resources
  + Manipulation of resources though representations
  + Self-descriptive messages
  + Hypermedia as the engine of application state (hateoas)

یکپارچگی ای پی ای است که از طریق پیروی از 4 راه امکان پذیر می شود

1. هر منبع ما باید دارای شناسه ی مختص به خود باشد مثال user یا order باید شناسه جدا داشته باشند یعنی url خاص مختص به خود api/user مربوط به خود است(identification)
2. دیتای که از سمت سرور و کلاینت ارسال می شود باید جامع باشد که امکان ویرایش داده باشد و دیتای جدید ایجاد شود(manipulation)
3. درخواست کی از کلاینت به سرور ارسال می شود باید تمامی اطلاعات مورد نیاز سرور را ارسال کند و همچنین سرور نیز اطلاعات مورد نیاز کلاینت را ارسال کند و کم کاستی نباشد (self-descriptive)
4. درخواستی که سمت سرور ارسال می شود باید در نتیجه ای که سرور می دهد url های مورد نیاز نیز برگردانده شود مثال کاربر product را دریافت می کند product/1 کاربر ارسال می کند و در پاسخ آدرس حذف ویرایش و ... مربوط به product نیز از سرور بازگرداننده می شود. (hateos)

* **Client server**

باید client از server جدا توسعه داده شود و چسبندگی وجود نداشته باشد

* **State less**

درخواستی که سمت سرور ارسال می شود نباید اطلاعاتی بین 2 درخواست در سرور ما باقی بماند

* **Layer system**

سیستم ما ممکن است چند لایه باشد کش احراز هویت و ... و لایه های مختلف حق ندارند درخواست را دستکاری کنند و کلاینت نباید از لایه های ما با خبر باشد

* **Cacheable**

برای client باشد مشخص کنیم که داده ای که می دهیم قابل کش کردن است یا نه به مدت زمان خاصی

* **Code on demand (optional)**

جای ارسال داده می توان از سرور اسکریپت ارسال کرد به کاربر.

**RestFull**

4 سطح دارد از 0 تا 3 می توان هر سطح را پیاده سازی کرد.

0 pos swamp

باید متد داشته باشیم و url داشته باشیم و از مدل xml استفاده کنیم

1 uri

url های مختلفی برای Resource های خود داشته باشیم تمامی درخواست های مثالا post باشد ولی باید url های مختلفی داشته باشیم برای حذف ایجاد و ...

2 http

Resource ها و url ها جدا هستن و http verb های مختلفی داریم مثالا product داریم منبع و pug get post delete دارد.

3 hypermedia Hateoas

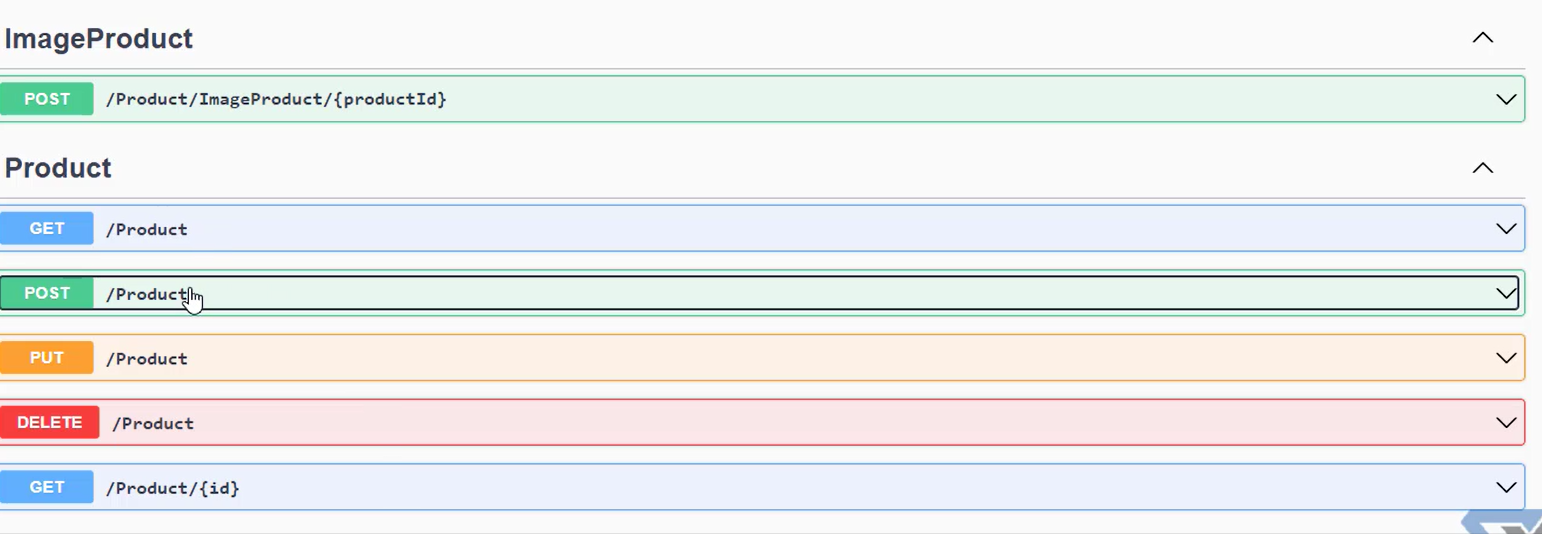
پیاده سازی hateos که در بالا توضیح داده شده است.

خلاصه:

url ها باید از یک دیگر جدا باشند.

Resource ها باید از یک دیگر جدا باشند

httpVerb ها را بزاریم مثلا product/ برای افزودن post و get و... هر کدام جدا باشد یعنی هر resource باید یک url داشته باشد مگر این که پارامتر های مختلفی بخواهیم داشته باشیم.



Verb های مختلف برای یک resource برای action های مختلف