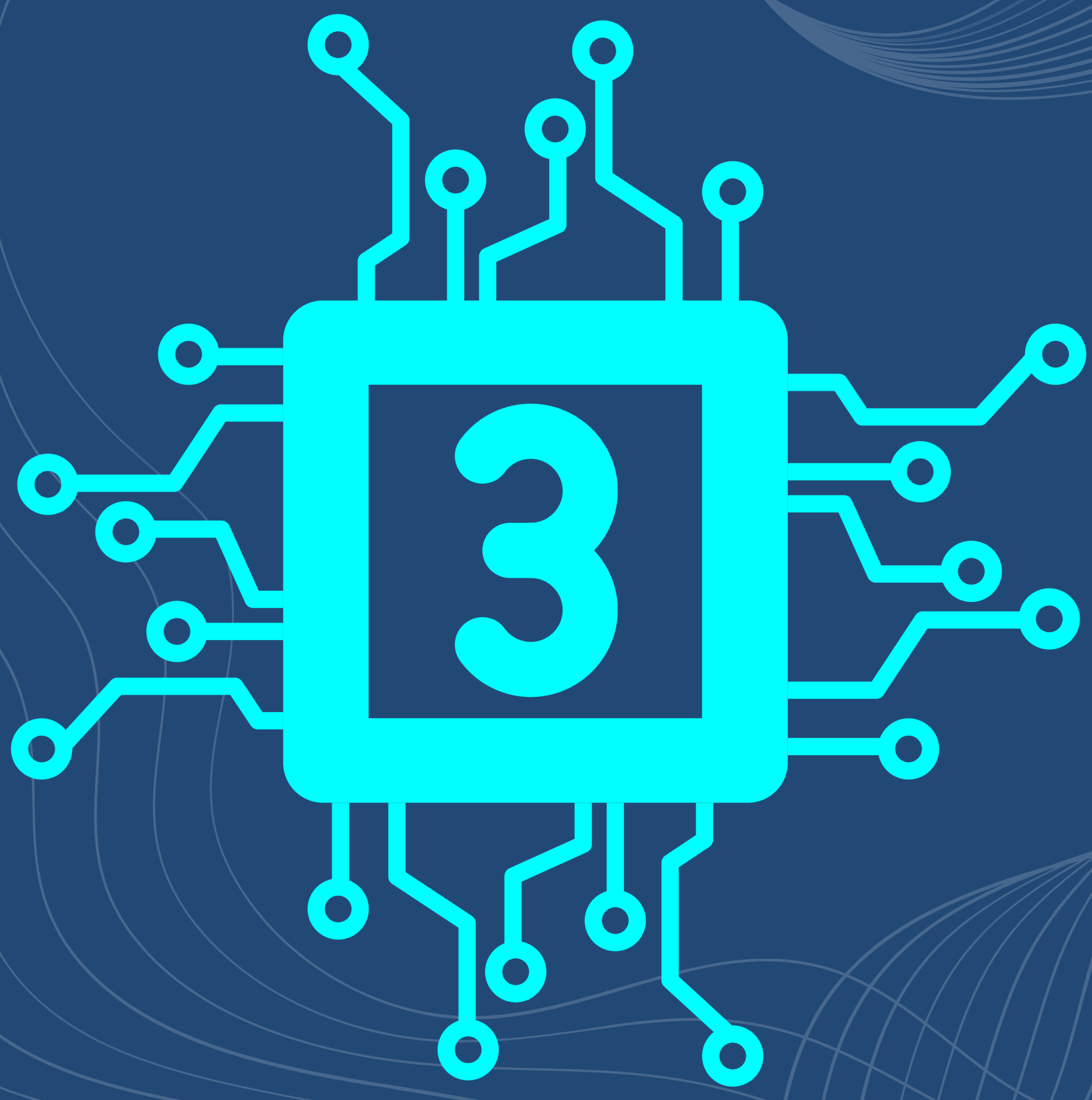
CODE TALE



توسعه بک‌اند
BACKEND DEVELOPMENT

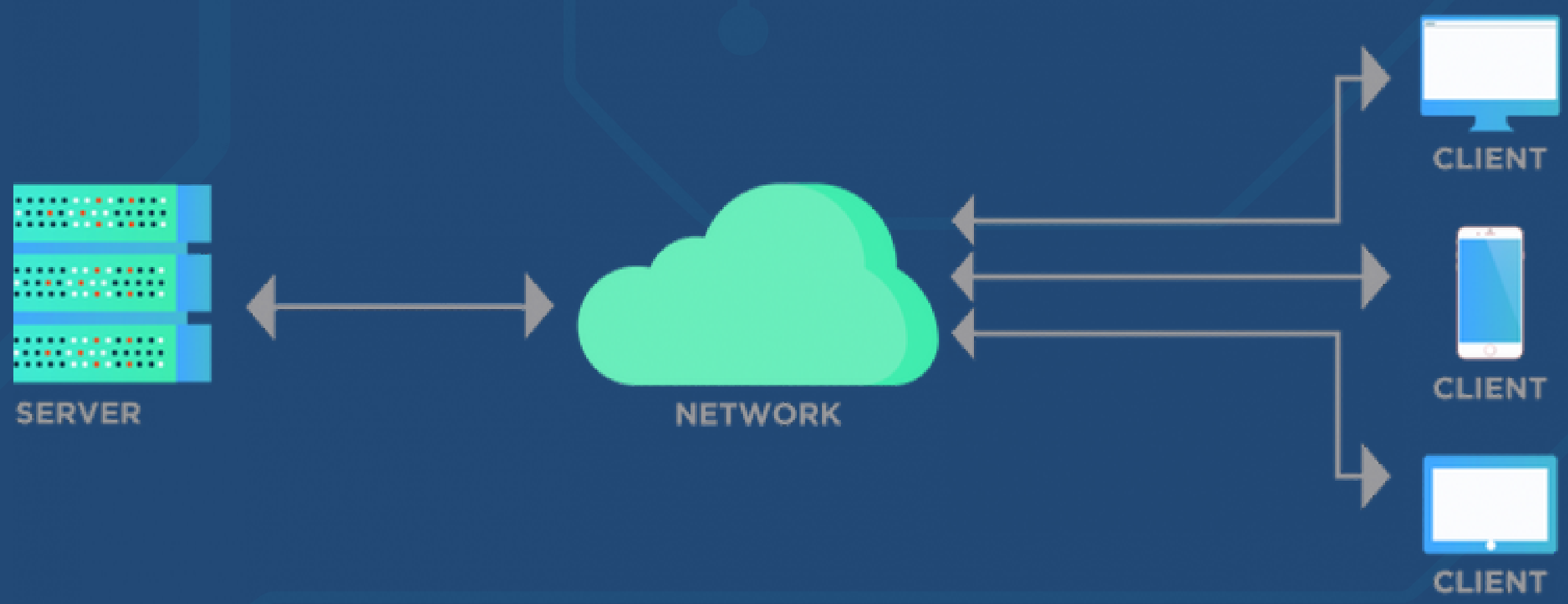
درس برنامه سازی پیشرفته

دکتر مجتبی وحیدی اصل
نشریه دانشکده مهندسی و علوم کامپیوتر

سلام! در این قسمت از CODE TALE می‌خواهیم نگاهی به دنیای بک‌اند (Backend) بیاندازیم و ضمن توضیح یک سری از مفاهیم کلی و مهم این حوزه، سعی می‌کنیم دید کلی از وب و پروسه‌ای که از دید کاربر پنهان است به شما بدهیم. از آنجایی که به عنوان یک مهندس کامپیوتر، در ادامه با این مفاهیم زیاد برخورد خواهید داشت، توصیه می‌کنیم حتما تا آخر این قسمت از CODE TALE همراه ما باشید.
در این قسمت به مفاهیم کلاینت-سرور (Client-Server)، بک‌اند، API، پایگاه‌داده (Database) خواهیم پرداخت.

Client-Server

خب بهتر است قبل از رفتن به سراغ ماجرای بک‌اند با مفاهیم کلاینت (Client) و سرور (Server) آشنا شویم. کلاینت و سرور هر دو کامپیوترهایی هستند که کارکردشان اینگونه است که کلاینت از طریق اینترنت یک درخواست (request) به سرور می‌دهد. سپس سرور پس از گرفتن درخواست، آن را می‌خواند و عملیاتی که برای آن درخواست تعیین شده است را انجام می‌دهد و در آخر یک پاسخ (Response) به کلاینت برمی‌گرداند. وظیفه سرورها این‌است که درخواست را از کلاینت‌های مختلف بگیرند و طبق پیکربندی‌شان یک پاسخ به کلاینت برگردانند.
برای مثال وقتی شما روی لینک یک سایت کلیک می‌کنید یک درخواست http یا https برای وب‌سرور می‌رود و وب‌سرور یک پاسخ در قالب html به شما برمی‌گرداند و آن کد HTML یا XHTML در کامپیوتر شما که در اینجا کلاینت هست اجرا می‌شود و صفحه آن لینکی که بر روی آن کلیک کردید برای شما نمایش داده می‌شود. یا وقتی در یک سایت بر روی یک دکمه کلیک می‌کنید یک درخواست به سمت وب‌سرور می‌رود و متناظر با آن پاسخی داده خواهد شد. در بخشی که API را توضیح می‌دهیم بیشتر با درخواست‌ها و پاسخ‌ها آشنا می‌شویم.



Backend

چرا ابتدا سراغ کلاینت و سرور رفتیم؟
برای اینکه متوجه شویم بخش فرانت‌اند اپلیکیشن‌ها در سمت کلاینت و بخش بک‌اندشان در سمت سرور اجرا می‌شود. حالا در قالب یک مثال می‌رویم تا با بک‌اند بیشتر آشنا شویم. قطعا همه ما با اپلیکیشن اینستاگرام (Instagram) آشنایی داریم. وقتی وارد این سایت یا اپلیکیشن می‌شوید با تعدادی عکس یا ویدیو مواجه می‌شوید. این عکس‌ها و ویدیوها قبلا توسط کاربرها آپلود شده‌اند و با این کار در محیطی به اسم پایگاه‌داده اینستاگرام ذخیره می‌شوند. وقتی شما روی ویدیو کلیک می‌کنید یک درخواست از کلاینت به سرورهای اینستاگرام می‌رود و سرور عکس یا ویدیو را همراه اطلاعات دیگر پست مثل کپشن و ... به کلاینت می‌فرستد و در نهایت برای شما نمایش داده می‌شود. پس یکی از مهم ترین مسائلی که در توسعه بک‌اند با آن سروکار داریم، تعیین سازوکاری برای ذخیره‌سازی اطلاعات است که در واقع منجر به وجود انواع سرویس‌ها و پایگاه‌داده‌ها شده. حال هر کاربر ممکن است اطلاعات شخصی مهمی را در اپلیکیشن ثبت کرده باشد که فقط می‌خواهد خودش یا افراد مشخصی به آنها دسترسی داشته باشند؛ بنابراین ذخیره‌سازی اطلاعات در سرورها چالش‌هایی نظیر تعیین سطح دسترسی و جلوگیری از سرقت اطلاعات را به همراه دارد که باید در توسعه بک‌اند به آنها توجه کرد.
از پراستفاده‌ترین زبان‌های حال حاضر برای توسعه بک‌اند:

JavaScript Python Ruby Java C#

برای مدیریت و راحتی بیشتر در کد زدن میتونیم از موارد زیر در فرایند توسعه بک‌اند استفاده کنیم.

1. Backend Frameworks
2. Package Managers

چند مورد از پراستفاده‌ترین فریمورک‌های بک‌اند:

ExpressJS Python Django Ruby on Rails Java Spring ASP.NET Core

همچنین ما در توسعه بک‌اند از پکیج (Package) استفاده می‌کنیم. از پکیج‌ها برای کارهای متداول مانند محاسبات، ارتباط گرفتن با پایگاه‌داده و ... کمک می‌گیریم برای همین ما نیاز ابزاری برای نصب و مدیریت این پکیج‌ها داریم و اسم این ابزار Package Manager هست.

چند تا از پراستفاده‌ترین Package Managerهای زبان های برنامه نویسی موجود:

JavaScript: npm Python: pip Ruby: RubyGems Java: Maven C#: NuGet

ابزارهایی که معرفی شدند به همراه پایگاه‌داده‌ها ابزارهای اصلی هستند که ما برای توسعه بک‌اند از آنها استفاده می‌کنیم. حالا یکی از سوال‌هایی که ممکن است برایتان پیش بیایید این است که بک‌اند و فرانت‌اند چگونه به هم وصل می‌شوند و داده با هم رد و بدل می‌کنند؟ با API
در بخش بعد نحوه رد و بدل داده بین فرانت‌اند و بک‌اند را بیشتر توضیح می‌دهیم.

API

اگر کمی با دنیای نرم‌افزار آشنا باشید قطعاً کلمه API به گوش‌تان خورده. پس بیایید ببینیم API از کجا آمده و یعنی چه؟
برای اینکه ببینیم API از کجا آمده برمی‌گردیم به اوایل زمانی که انسان‌ها از کامپیوتر و نرم‌افزارها استفاده می‌کردند. در آن زمان‌ها انسان‌ها بودند که بیشترین تعامل را با نرم‌افزارها داشتند مثل بازی‌هایی همچون Tetris, Doom و بقیه بازی‌های قدیمی یا استفاده از ماشین حساب و نرم‌افزارهای ویرایش عکس و ... که در همه آنها فقط انسان‌ها بودند که با نرم افزار تعامل می‌کردند. اما به مرور زمان و پیشرفت تکنولوژی این نیاز احساس شد که به جز اینکه فقط انسان با نرم‌افزارها تعامل کند خود نرم‌افزارها هم بتوانند بدون دخالت مستقیم انسان باهم تعامل کنند و اینجا بود که مفهوم API شکل گرفت.
API مخفف Application Programming Interface که می‌شود آن را رابط برنامه‌نویسی اپلیکیشن ترجمه کرد. API مجموعه‌ای از پروتکل‌ها، روندها و ابزارهایی است که به نرم‌افزارهای مختلف یا اجزای مختلف در یک برنامه اجازه می‌دهد تا با یکدیگر ارتباط برقرار کنند. یعنی APIها می‌توانند به عنوان پلی بین دو برنامه مجزا یا سرویس داخلی یک برنامه عمل کنند و به آن‌ها اجازه می‌دهند داده‌ها و عملکردهای خود را با همدیگر به اشتراک بگذارند.

REST API

همان طور که توضیح دادیم به پل ارتباطی بین دو یا چند جزء (Component) داخل یک نرم افزار یا خارج دو اپلیکیشن مجزا API می‌گویند ولی API یک مفهوم کلی است و برای این پل ارتباطی ما نیاز به یک ساختار و معماری برای این ارتباط و تبادل اطلاعات داریم.

حالا می‌خواهیم کمی دقیق‌تر یکی از محبوب‌ترین ساختار یا معماری‌های API یعنی REST را با هم بررسی کنیم.

کلمه REST مخفف Representational State Transfer است که شاید بتوان آن را "نمایندگی انتقال وضعیت" ترجمه کرد. همانطور که قبل‌تر اشاره کردیم از آنجایی که فرانت‌اند در سمت کلاینت و بک‌اند در سمت سرور اجرا می‌شود، از سمت کلاینت یک درخواست به سمت سرور ارسال می‌شود و سرور یک پاسخ به کلاینت برمی‌گرداند و اینگونه با هم ارتباط برقرار می‌کنند.

در معماری REST هم به همین شکل است که یک درخواست از نوع http از فرانت‌اند به بک‌اند ارسال می‌شود و بک‌اند یک پاسخ که اکثرا قالب آن JSON است برمی‌گرداند.

JSON

در میانه REST API نیاز است که کمی با JSON بیشتر آشنا شویم.

json یک فرمت فایل می‌باشد و قالب آن متن (Text) است که برای تبادل داده بین کامپیوترها استفاده می‌شود و از مزیت‌هایش می‌توان به سبکی، خوانایی و استفاده آسان از آن در زبان‌های برنامه‌سازی مختلف اشاره کرد ولی نحو (Syntax) و انواع داده آن از جاوا اسکریپت آمده‌اند.

معمولا JSON را بر پایه Object یا آرایه‌ای از Objectها می‌نویسند. Objectهای جاوا اسکریپت به صورت ""key" : "value هستند که value آن باید یکی از این انواع داده باشد:

string، number، object، array، boolean، null

و در آخر به راحتی می‌توانید با دادن نشانی داده مدنظر مانند key یا اندیس آرایه داده مورد نظرتان را بگیرید و از آن استفاده کنید.

```
{"items":
    [
        {
            "id": 0,
            "name": "Log",
            "desc": "Just an ordinary log",
            "sprite": "log",
            "value": 1
        },
        {
            "id": 1,
            "name": "Rock",
            "desc": "A heavy rock",
            "sprite": "rock",
            "value": 1
        }
    ]
}
```

نمونه ای از یک json ساده

CRUD

خب برگردیم به ادامه REST. همانطور که گفته شد در REST درخواست‌ها از نوع http هستند و ما برای ارتباط با بک‌اند در درخواست‌ها از 4 عملیات استفاده می‌کنیم که مخفف این 4 عملیات می‌شود CRUD. این چهار عملیات عبارتند از :
Create, Read, Update, Delete

- از متد POST برای ایجاد کردن یا فرستادن اطلاعات جدید به سرور (Create)
- از متد GET برای گرفتن یا خواندن اطلاعات از سرور (Read)
- از متد PUT برای جایگزین یا آپدیت کردن اطلاعاتی که قبلا مقدار دیگری داشتند (PUT)
- از متد Delete برای پاک کردن اطلاعات استفاده می‌کنند (Delete)

برای توضیح نحوه استفاده از این عملیات‌ها به سراغ یک مثال می‌رویم. تصور کنید شما می‌خواهید از یک سایت غذا سفارش دهید و در مرحله اول قصد دارید منوی رستوران را ببینید. پس اگر URL آن صفحه از سایت که در آن منو را می‌شود دید به صورت http://kababi.com/menu باشد /menu نقطه پایانی (Endpoint) ماست. یعنی همان قسمتی از بک‌اند که درخواست ما به آنجا فرستاده می‌شود. پس اگر به مثلا به آدرس http://kababi.com/menu
بروید یک درخواست با متد GET و نقطه پایانی /menu ارسال می‌شود که پاسخ آن می‌تواند یک JSON بدین شکل باشد:
و اینگونه منو از سرور گرفته می‌شود و نمایش داده می‌شود.

```
{"foods":
    [
        {
            "id" : 0,
            "name" : "koobide",
            "price" : 120
        },
            {
            "id" : 1,
            "name" : "jooje",
            "price" : 130
        }
    ]
}
```

و حالا بعد از دیدن منو و تصمیم گیری به آدرس
http://kababi.com/order
برای ثبت سفارش رفته‌اید و در نظر دارید غذا سفارش بدهید، از آنجایی که می‌خواهیم از کلاینت به سرور اطلاعات بفرستیم پس اینجا متد POST استفاده می‌کنیم و نقطه پایانی ما /order است ولی ما به یک بدنه (Body) برای این متد نیازمند هستیم که در آن اطلاعاتی که می‌خواهیم به سرور بفرستیم را به صورت JSON ارسال کنیم. بدنه ما هم برای مثال می‌تواند به این شکل باشد:

```
{
  "orderData": {
    "id": 0,
    "orderName": "koobide",
    "shipments":
    {
        "name": "Ali Alavi",
        "address": "velenjak sbu",
        "phoneNumber": "09120000000"
    }
  }
}
```

حال اگر قبل از نهایی کردن سفارش اگر نظرتان تغییر کرد با متد PUT می‌توان اطلاعات قبلی را آپدیت کرد. یا اگر کلا منصرف شدید با متد DELETE سفارش شما را می‌توان حذف کرد.

معماری REST جز محبوب‌ترین معماری API هست و معماری‌های دیگری مانند GraphQL , gRPC و... هستند که این APIها بسیار نقش مهمی در دنیای ارتباطات ایفا می‌کنند.

حال نوبت به پایگاه‌داده (Database) می‌رسد.

اما پایگاه‌داده چیست؟

پایگاه‌داده یک سیستم مرتبط با ذخیره و مدیریت داده‌ها است که توسط برنامه‌ها و کاربران برای دسترسی، افزودن، بروزرسانی و حذف داده‌ها استفاده می‌شود. وجود پایگاه‌داده‌ها برای انجام عملیات مرتبط با داده‌ها در سایت‌ها و برنامه‌ها اساسی است.

چرا به پایگاه‌داده نیاز داریم؟

دلایل مختلفی برای نیاز برنامه نویسان به پایگاه‌داده وجود دارد که چند مورد از آنها را مطرح می‌کنیم:

- ذخیره داده‌ها: یکی از نیازهای اصلی در توسعه برنامه‌ها، ذخیره داده‌ها است. پایگاه‌داده‌ها به عنوان مخازن مرکزی برای ذخیره انواع داده‌ها از جمله اطلاعات کاربران، محتوا، تنظیمات و ... عمل می‌کنند.
- مدیریت داده‌ها: با استفاده از پایگاه‌داده، توسعه‌دهندگان به راحتی می‌توانند داده‌ها را دریافت، به‌روزرسانی، حذف و مدیریت کنند.
- امنیت داده‌ها: با استفاده از پایگاه‌داده‌های معتبر و امن، اطلاعات کاربران و اطلاعات حساس دیگر محفوظ می‌مانند. پایگاه‌داده‌ها امکانات مختلفی را برای ایجاد سطوح مختلف دسترسی و کنترل امنیتی ارائه می‌دهند که به توسعه‌دهندگان کمک می‌کند تا از نفوذها و حملات مخرب جلوگیری کنند.

به طور کلی پایگاه‌داده‌ها به دو دسته تقسیم می‌شوند:

پایگاه‌داده‌های رابطه‌ای (SQL):

در پایگاه‌داده‌های رابطه‌ای، جداول و رابطه‌هایی ذخیره می‌شوند که بین آنها ارتباطاتی تعریف شده است و PostgreSQL یکی از معروف‌ترین نمونه‌های پایگاه‌داده رابطه‌ای است که برای بسیاری از برنامه‌های وب و سیستم‌های تجاری استفاده می‌شود.

پایگاه‌داده‌های غیر رابطه‌ای (NoSQL):

در پایگاه‌داده‌های غیر رابطه‌ای یا NoSQL، داده‌ها به صورت اشیاء ذخیره می‌شوند و تنها به طریقی توسعه‌پذیر و فشرده می‌شوند. پایگاه‌داده MongoDB یکی از معروف‌ترین نمونه‌های پایگاه‌داده NoSQL است که برای برنامه‌هایی که نیاز به ذخیره داده‌های نامنظم و بزرگ دارند، مورد استفاده قرار می‌گیرد.

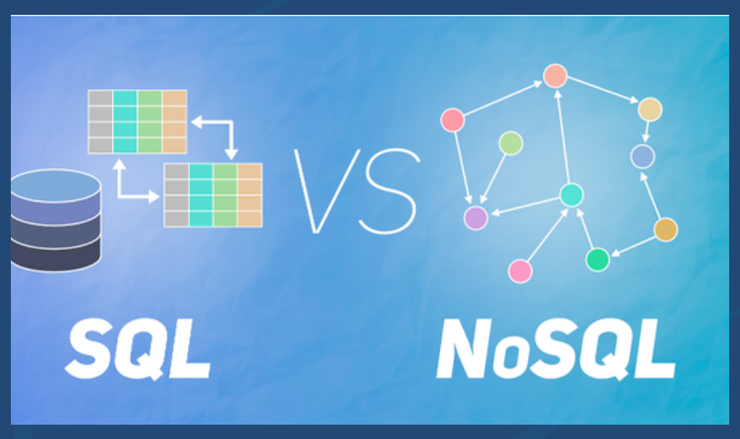
mongoDB

پس بپردازیم به اینکه طور کلی SQL و NoSQL در چه مواردی با هم تفاوت دارند؟

ساختار داده:

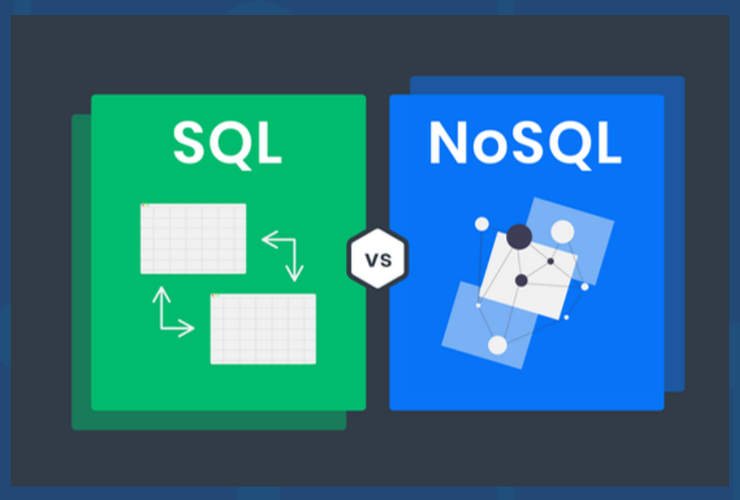
پایگاه‌داده‌های رابطه‌ای از جداول، ردیف‌ها و ستون‌ها تشکیل شده‌اند. داده‌ها در جداول ذخیره می‌شوند و بین جداول رابطه‌های معین وجود دارد.

در پایگاه‌داده‌های غیررابطه‌ای، داده‌ها به صورت اشیاء (Document)، کلید-مقدار (Key-Value) یا ستونی-واحد (Column-Family) ذخیره می‌شوند. این انواع پایگاه‌داده‌ها ساختار انعطاف‌پذیری دارند که به توسعه‌دهندگان اجازه می‌دهد داده‌ها را به شیوه دلخواه ذخیره کنند.



انعطاف‌پذیری در طراحی:

پایگاه‌داده‌های رابطه‌ای برای تعریف ساختار داده‌ها نیازمند تعریف الگوی مشخص (Schema) هستند که باید پیش از استفاده مشخص شود. در پایگاه‌داده‌های غیررابطه‌ای، انعطاف‌پذیری بالاتری در مورد طراحی و تغییرات در ساختار داده وجود دارد. بسیاری از این پایگاه‌داده‌ها از ساختار بدون الگو (Schemaless) پشتیبانی می‌کنند که به توسعه‌دهندگان امکان می‌دهد بدون نیاز به تعریف الگو، داده‌ها را ذخیره کنند.



قابلیت افزایش مقیاس‌پذیری:

پایگاه‌داده‌های رابطه‌ای معمولاً مقیاس‌پذیری عمودی را پشتیبانی می‌کنند، به این معنی که برای افزایش ظرفیت، باید منابع سخت‌افزاری موجود را ارتقاء داد.
پایگاه‌داده‌های غیررابطه‌ای معمولاً مقیاس‌پذیری افقی را پشتیبانی می‌کنند، که به این معنی است که می‌توان با افزایش تعداد سرورها و توزیع بار، به طور افزایشی ظرفیت پایگاه‌داده را افزایش داد.

کاربرد های متفاوت:

پایگاه‌داده‌های رابطه‌ای معمولاً برای سیستم‌هایی با ساختار داده ثابت و استفاده‌هایی مانند سامانه‌های مالی، سیستم‌های مدیریت محتوا و سیستم‌های حسابداری مناسب هستند. پایگاه‌داده‌های غیررابطه‌ای معمولاً برای برنامه‌هایی با داده‌های نامنظم و بزرگ مانند شبکه‌های اجتماعی، سیستم‌های تجارت الکترونیک، اپلیکیشن‌های موبایل و سیستم‌های IOT (که در هفته نامه های بعدی به آن خواهیم پرداخت) مناسب هستند.

حال بیشتر در مورد تکنیک‌های برنامه نویسی موجود در بک‌اند صحبت کنیم:

اولین چیزی که مایل بودیم برای شما در مورد آن صحبت کنیم ORM ها هستند.

اما ORM چیست و چرا به آن نیاز داریم؟

شاید کار کردن با پایگاه‌داده و یادگیری زبان SQL برای عده‌ای طاقت‌فرسا به نظر برسد.ORM یا Object Relational Mapping در واقع یک لایه انتزاعی برای ارتباط برقرار کردن با پایگاه‌داده بوده و تا زمانی که از پایگاه‌داده‌های رابطه‌ای استفاده کنید، شما را از نوشتن کدهای SQL بی‌نیاز می‌کند.

یک مثال ساده و قابل درک :

یک مثال ساده از کاربرد ORM در توسعه یک دفترچه تلفن خواهد بود. در این حالت شما یک کلاس برای دفترچه تلفن مورد نظرتان ایجاد کرده و سپس مشخصات هر فردی که قصد وارد کردن به این دفترچه تلفن را دارید به صورت یک سری ویژگی (Property) ایجاد می‌کنید. مطمئناً هر ویژگی حاوی یک سری نوع داده خاص خواهد بود که ORM در فرایند تعریف این نوع‌ها به شما کمک خواهد کرد.در واقع شما صرفا با شی ساخته شده کار می‌کنید در حالی که هر تغییری در آن شی ایجاد شود به طور خودکار اعمال می‌شود بدون اینکه نیاز به نوشتن query خاصی در پایگاه داده باشیم.

سوال این است که چگونه می‌توانیم از ORM استفاده کنیم؟

هر زبان برنامه‌نویسی شئ‌گرا یکسری کتابخانه منحصر به فرد در اختیار شما قرار خواهد داد که با استفاده از آن می‌توانید به قابلیت‌های ORM دسترسی داشته و از آن استفاده کنید. برای مثال در زبان برنامه نویسی پایتون ما قابلیت استفاده از ORM را از طریق کتابخانه‌هایی مانند SQLAlchemy، Peewee ORM Django و موارد دیگری را خواهیم داشت. در زبان برنامه‌نویسی جاوا اسکریپت نیز مواردی مانند Prisma، TypeORM، Sequelize و... وجود دارند.

مفهوم بعدی که مایل هستیم در مورد آن صحبت کنیم مفهوم تراکنش (Transaction) است!

می‌دانیم که هنگام ذخیره‌سازی اطلاعات در یک پایگاه‌داده ممکن است چند اتفاق به صورت موازی رخ دهد. به طور مثال ذخیره‌سازی انتقال وجه از حساب بانکی A به حساب بانکی B شامل یک عمل برای کسر پول از حساب مبدا و یک عمل برای اضافه کردن پول به حساب مقصد را در نظر بگیرید. این یک مثال هستش و به هر تعداد عمل مورد نظر که به طورمتوالی انجام می‌شوند و وابستگی به یکدیگر دارند می‌توان مفهوم را تعمیم داد.

اگر به هر دلیلی نتوانیم تمامی اطلاعات را ذخیره کنیم دچار مشکل خواهیم شد! برای مثال پس از کسر از حساب مبدا پول به حساب مقصد نشیند. در این صورت باید پول به حساب مبدا برگشت داده شود. تراکنش (Transaction) در واقع مجموعه ای از چند دستور است که یا با هم با موفقیت اجرا می‌شوند و یا هیچکدام از آنها اجرا نخواهند شد.

حال باید در مورد ACID صحبت کنیم. در واقع ACID یکی از اصلی‌ترین ویژگی یک تراکنش است. توجه داشته باشیم که ACID به معنی Atomicity Consistency Isolation Durability می‌باشد.

- اتمی بودن (Atomicity): نشان می‌دهد که بخشی از تراکنش به تنهایی قابل اجرا نمی‌باشد.
- ثبات (Consistency): تضمین می‌کند که پس از اجرای موفقیت آمیز تراکنش و یا شکست آن داده‌ها در حالت سازگار و معتبر قرار می‌گیرند.
- ایزوله بودن (isolation): تضمین می‌کند که داده‌های درگیر در تراکنش، توسط سایر پروسه‌ها غیر قابل تغییر هستند.
- دوام (Durability): تضمین می‌کند که عوامل ناخواسته مثل رفتن برق باعث حذف تأثیر تراکنش نهایی شده (Commit) نشوند.

“

If you look at history, innovation doesn't come just from giving people incentives; it comes from creating environments where their ideas can connect.

“

Steven Johnson (born 1968), Science author & media theorist

code tale

شماره: 3

تهیه شده توسط: سارا اکبرزاده، آرمان چم‌حیدری، امیرحسین صدر، امیرقائمی و آراس ولی‌زاده

هفته نامه code tale بهونه ای برای بیشتر دونستن و نگاه همه جانبه به موضوعات رشته کامپیوتر هستش .
دستیاران آموزشی درس برنامه سازی پیشرفته هدف والای خودشون رو بدین شکل می‌دونن که دانشجویان درس با مفاهیم روز رشته خود آشنا باشند و بتونن سریعتر علاقه خودشون رو پیدا کنن .
امیدواریم از خوندن این قسمت لذت برده باشین .