# Vague Prompt:

Crafting a prompt that is too vague or broad can often lead to responses from ChatGPT that are off-topic or not useful for the game. For example, if you ask something like "What happens next?" without providing enough context, the model might produce a response that doesn't fit the narrative. To overcome this, it's essential to provide sufficient context and details in the prompt. This could involve specifying the characters involved, their current situation, their surroundings, or their intended goals.

# ChatGPT Has a Lying Problem

It's hard to believe that we've only been living with ChatGPT for a matter of months. In that time, I'm sure you've heard how everyone's favorite LLM (large language model) will shred the fabric of modern life — by destroying white-collar jobs, making human creativity redundant, and giving teens the tools to abolish homework forever.

But beneath all the hype and occasional silliness, there are some serious problems that LLMs are about to unleash on our world. The most likely scenario is that the information space of the future — web content, comments, social media, and everywhere else that text exists — will be massively polluted by people misusing AI tools.

<https://medium.com/young-coder/chatgpt-has-a-lying-problem-c88f2ba2b010>

# Simplify Relational Database + Elasticsearch architecture with TiDB

As a developer, you know that a relational database (RDS) such as MySQL, or PostgreSQL is great for storing structured data and handling online transactional workloads. However, when it comes to handling analytics, and indexing large amounts of data, a different storage engine such as Elasticsearch is often needed. It is a popular choice for online services with hybrid workloads.

Developers need to handle two types of API: SQL and Elasticsearch API. This is not only about different wired protocols, but also different behaviors while processing data. Developers need to build two modules to interact with both sides. This would introduce complexity to the code.

TiDB provides a highly available, horizontally scalable, and fault-tolerant architecture that allows it to handle high write and read loads. It also has built-in support for distributed indexes, which eliminates the need for separate indexing and analytics engines like Elasticsearch. Additionally, it's compatible with the MySQL protocol, making it easy for developers to use and integrate with their existing applications.

<https://medium.com/@shenli3514/simplify-relational-database-elasticsearch-architecture-with-tidb-c19c330b7f30>

# gRPC with microservice

I will talk about the work that we have done to improve communication performance between services, in a project that was developed with microservice architecture by the TDesk team in Trendyol.

TDesk is an application that was located behind an API gateway and has so many domain services that developed different languages. The communication is provided with REST architecture and HTTP way between API gateway and services.

After some investigation to improve communication performance, we decided to make a POC about gRPC and we would like to share our experience about this process.

<https://medium.com/trendyol-tech/using-grpc-in-microservices-communication-and-performance-results-between-different-languages-4bd13890002a>

# Event-Driven Architectures

Event-driven architecture is a popular software design pattern that enables a system to react to various events that occur in the application. In this pattern, there are two types of objects: event emitters and event listeners. Event emitters trigger events, while event listeners listen and react to those events.

Event-driven applications are common for use cases including IoT, fraud detection, payment processing, website monitoring, and real-time marketing. Event-driven applications often treat data as immutable, or unchangeable, making it easy to look up the values of data at previous points in time.

It can handle heavy traffic with a large number of events with low latency. Message broker makes it an ideal choice for applications that need to process a large number of messages in real-time, such as financial trading systems, social media applications, and gaming platforms.

<https://medium.com/scb-abacus/introduction-to-event-driven-architecture-in-node-js-typescript-354302bc055c>

<https://medium.com/@vraghani/event-driven-architecture-386c5da5ebfc>

# Caching Strategies to Remember while designing Cache System

The goal of Caching is to reduce the number of times data needs to be fetched from its original source, which can result in faster processing and reduced latency.

Caching can be implemented at various levels of architecture, including in-memory caching, disk caching, database caching, and CDN caching.

Data can be cached using different techniques, each with its own benefits. In-memory caching stores data in the computer's primary memory, enabling fast access compared to disk storage.

Disk caching, on the other hand, stores data on the hard disk, which is slower than primary memory but faster than accessing data from a remote source.

With database caching, frequently accessed data is stored within the database, reducing the need to retrieve data from external storage.

Lastly, CDN caching involves storing data on a distributed network of servers, reducing latency when accessing data from remote locations.

<https://javascript.plainenglish.io/6-caching-strategies-to-remember-while-designing-cache-system-da058a3757cf>

# Saying Goodbye to Ingress: Embracing the Future of Kubernetes Traffic Management with Gateway API and Cilium

In the dynamic world of Kubernetes, managing ingress traffic efficiently and securely has always been a crucial challenge.

However, a groundbreaking solution has emerged, promising to transform the way we handle traffic routing within Kubernetes clusters. Meet the Gateway API: a long-term replacement for Kubernetes Ingress that brings a wealth of benefits to operators, including role-based access control, portability, and extensibility.

Gateway API is an open-source project managed by the SIG-NETWORK community. It is a collection of resources that model service networking in Kubernetes. These resources — GatewayClass, Gateway, HTTPRoute, TCPRoute, Service, etc - aim to evolve Kubernetes service networking through expressive, extensible, and role-oriented interfaces that are implemented by many vendors and have broad industry support.

<https://itnext.io/saying-goodbye-to-ingress-embracing-the-future-of-kubernetes-traffic-management-with-gateway-api-6584b7b8f913>

# My Developer Friend's 2-step Way to Become a Multi-millionaire

That's a good and a bad thing. Why good? Obviously… because it's good money. Money trumps most other benefits from a job, and development wins here.

Passion is good. Today we will talk about how you can become a millionaire only with coding and how my friend used it to become a millionaire.

There are many ways to look for opportunities. But the best way is to consciously look for opportunities and give more time to them.

Being a millionaire can never be achieved by books or a specific set of rules because everyone's situation is different. Everyone is different. But one of my favorite motivational lines is from a Jeff Bezos conference:

Life is too short of making all the mistakes

So, you gotta learn from other people's mistakes and other people's successes. So that's the whole purpose of today's article. Hope that it will help at least one developer who just read it.

# So Long “Prompt Engineering,” We Hardly Knew Ya

It's all about how we humans identify, analyze, and define problems. When we can illustrate the problem, AI will provide the most efficient way to counter it. And there are four ways to examine and define problems.

Diagnosis is discovering the problem that AI can solve. This is the human part of knowing that a problem exists. Learning to ask the right questions, look at the different ways that the problem can be seen.

Decomposition is about splitting the big problems into bite-sized ones. Take the problem apart, examine it, and let AI help you determine your findings since it handles data so well. Instead of tackling the biggest problem, take it apart and work on the smaller parts to achieve small successes.

Reframing is about shifting your perspective and seeking new interpretations. Extrapolating and recombining the parts of the problem in order to identify the meta components. Perhaps a new way of looking at the problem may find a solution hidden in plain sight.

Constraint design is about setting boundaries for the solution. Knowing what to accomplish, and when to know it is done. Setting the length, style, and description of the audience can help AI understand its mission. But we have to know that first in order to instruct.

# Why FastAPI is a Future of Python Web Development

If you are a Python Back-end developer or at least somehow connected to the field you must have heard about FastAPI.

FastAPI is a modern, fast (hence the name) web framework for building APIs with Python. It is built on top of the Starlette framework and relies heavily on Python's type annotations to provide fast and efficient validation, serialization, and documentation of API requests and responses.

FastAPI is a powerful and efficient Python web framework that has been gaining popularity in recent years. With its impressive speed, simple API, and built-in documentation, FastAPI is an excellent choice for building high-performance APIs. Its use of Python's type hints for automatic validation and serialization eliminates boilerplate code and makes development faster and easier. Additionally, FastAPI's support for ASGI and dependency injection further enhances its capabilities, making it a top choice for both beginner and experienced developers. Overall, if you're looking for a modern and reliable Python web framework, FastAPI is definitely worth considering.

# Understanding Websockets In depth

WebSockets are a powerful communication protocol that enables bidirectional, real-time communication between a client and a server over a single TCP connection. In this blog, we'll cover everything that you need to know about web sockets.

Web sockets allow us to create "real-time" applications which are faster and require less overhead then TRADITIONAL API PROTOCOL

Web socket uses a DUPLEX protocol for communication between client and server.

Websockets are reliable and easy to use and are act as a boon in the systems where real time nature is requested. Let say you are working on an application, where you need to display data in real-time that is changing continuously.

Websocket works by making a connection between client and server. In order to start the connection, clients sends a http GET request with a upgrade header, so that the server know that this is a upgrade request and it responds with status 101 if the server supports the upgrade properties and return error code if not.

# Are Kubernetes days numbered?

As a sequel to my earlier "Are Terraform's days numbered?" post, today I want to address a different question that Julio Ortega posed in the comments on that post. Are Kubernetes days numbered? For those who aren't familiar with Kubernetes, it has become the leading container orchestration system. If you want further explanation I quite like this tongue-in-cheek Kubernetes for kids video:

For those who are familiar with VMware, the analogy I tend to like is that Kubernetes works quite like vCenter, controlling the placement of workloads (containers rather than VMs) across a distributed cluster of nodes (or hosts in VMware parlance).

Kubernetes has multiple components that form its control plane along with small node-based components which coordinate networking and local monitoring of 'pods'. I won't go into more detail than that but needless to say, there are a fair few components that potentially require management. Indeed, if you have been in the Kubernetes ecosystem for a while you have probably heard or even tried Kelsey Hightower's 'Kubernetes the Hard Way' repository, which gives step-by-step instructions on how to bootstrap Kubernetes manually.

# 99% of Users Are Using ChatGPT the Wrong Way: Use Prompt Engineering and Be Ahead of the Game!

We all know that Artificial intelligence is a field that is continuously progressing, with new breakthroughs emerging on a daily basis.

The development of the GPT-3 and GPT-4 models, which are revolutionizing natural language processing, is one of OpenAI's most remarkable accomplishments. Here are a few of the instances.

In addition, Playground, a platform to interact with the GPT models and learn about their potential, has been developed by OpenAI.

# Prompt Engineering

Prompt engineering is a fascinating concept that is gaining traction in the AI industry.

At its core, it involves asking an AI model for specific details rather than vague generalities. For instance, instead of saying "I want to create a website," a prompt engineer would say something like "I want to create a finance tracker website using React.js and Redux."

This specificity allows the AI model to generate a more precise response that is tailored to the user's needs. OpenAI has even developed Playground, a tool that allows users to experiment with various prompts and see how the AI responds.

# Explaining Vector Databases

Vector databases have been getting a lot of attention recently, with many vector database startups raising millions in funding.

A vector database is a type of database that stores and manages unstructured data, such as text, images, or audio, in vector embeddings (high-dimensional vectors) to make it easy to find and retrieve similar objects quickly.

The main difference between relational databases and vector databases lies in the type of data they store. While relational databases are designed for structured data that fits into tables, vector databases are intended for unstructured data, such as text or images.

The type of data that is stored also influences how the data is retrieved: In relational databases, query results are based on matches for specific keywords. In vector databases, query results are based on similarity.

And this is one limitation of relational databases: You must add all the information you think someone might need to find that specific item. But how do you know which information and how much of it to add? Adding all this information is time-consuming and does not guarantee completeness.

Indexing — For this, a vector database indexes the vector embeddings. This step maps the vectors to a data structure that will enable faster searching.

Similarity Measures — To find the nearest neighbors to the query from the indexed vectors, a vector database applies a similarity measure. Common similarity measures include cosine similarity, dot product, Euclidean distance, Manhattan distance, and Hamming distance.

# How to Build a SaaS on AWS: a Deep Dive into the Architecture of a SaaS Product

Building a SaaS product is a complex task. There are many things to consider, from the business model to the technology stack. In this article, we will focus on the technology stack I use and learn how I build my SaaS product on AWS. I'll describe the various services and how they fit together to provide a scalable, secure, and reliable SaaS product.

It took me 5 months to build my first SaaS product. During this time, I had to learn a lot about AWS and gain a ton of experience. This article is an opportunity for me to share my knowledge, and I also hope it will inspire you to build your own SaaS product.

If you are currently building or planning to build your own SaaS product, you should be interested in my AWS SaaS Boilerplate. It's a React SaaS boilerplate that includes everything you need in a SaaS product. Fully hosted on AWS, you can deploy it in a few minutes with the same stack described in this post. So, you can use it as a starting point to build your own SaaS product and earn your 1st MRR.

# Microservices with NestJS

Nest (NestJS) is a NodeJS framework suitable for building efficient, scalable Node.js server-side applications. Nest offers a good structure and CLI tools for beginners to get started with the project. It allows you to easily scaffold a new feature with a single command from the CLI.

Nest offers an easy way to convert your application to the microservice architectural style of development. The benefits of it are increased scalability, better resilience and increased productivity. Nest supports several built-in transport layer implementations, called transporters, which are responsible for transmitting messages between different microservice instances. One of the benefits of this is that NestJS abstracts the implementation details and makes it easier to switch between different transporters. We can pick between direct form of communication or utilize a message broker in the process for asynchronous communication.

# Why do you need caching?

Any web application that has some success will eventually run into bottlenecks. The most common bottleneck is usually related to how information is fetched from a primary database, like Postgres or MySQL.

Indeed, as the number of users grows, so does the number of HTTP requests made to the server. This results in the same data being fetched all over again and again. Optimizing your application for speed and efficiency is important by caching frequently requested data.

Since most relational databases involve structured data, they are optimized for reliability and not for speed. That means the data they store on a disk is many times slower than the RAM. Using a NoSQL database does not bring any tremendous performance gains either.

The solution is to use an in-memory cache-store.

In this tutorial, we will implement caching in NestJs and ultimately scale it with Redis, a fast in-memory database that is perfect for this use case.

# What is Vector Embedding

In the realm of AI chatbot development, vector embedding plays a pivotal role in capturing the essence of textual information. At its core, vector embedding refers to the process of representing words, sentences, or even entire documents as dense, low-dimensional vectors in a mathematical space. Unlike traditional methods that rely on sparse representations like one-hot encoding, vector embeddings encapsulate the semantic relationships between words and enable algorithms to comprehend their contextual meaning.

The importance of vector embedding lies in its ability to transform the raw text into a numerical representation that algorithms can comprehend and reason with. This transformative process not only facilitates various natural language processing (NLP) tasks but also serves as a fundamental building block for large language models. Vector embeddings empower these models to leverage the rich semantic information embedded within the textual data, enabling them to generate more coherent and contextually appropriate responses.

# Tailwind vs Modern CSS

Tailwind is a beloved CSS framework, it has many loved features for creating user interfaces. In this article we're not taking a deep dive into Tailwind, instead we're going to isolate some of the concerns with the use of inline-classes in Tailwind as well as touching on new native CSS features that revolutionize how we approach those concerns.

Tailwind is massive, I believe Tailwind to CSS is bigger than React to JS/TS. So inevitably there will be many thought-leaders and mobs to sweet talk much of the concerns raised in this article away.

CTFO, nobody is telling you to stop using Tailwind. As a developer, it's more crucial to cheer-lead your own abilities and knowledge than the tools you use. This is just another source of knowledge like any other.

# Message Brokers: Queue-based vs Log-based

Message brokers are one of the essential features of modern distributed applications architecture. They provide an ability to asynchronously exchange messages for different parts of the system. Asynchronous messaging becomes more and more important, and so do message brokers. Today there are a lot of options to choose from, not only Rabbit and Kafka, but many more, including cloud-native solutions, provided by the biggest cloud providers like AWS or Azure. However, most of them can be divided into two groups: queue-based and log-based. Let's see how they work.

Queue-based

Queue-based systems are based on the Queue data structure that should be familiar to everyone from the first steps in Computer Science. Queue is a simple data structure working by the FIFO (First In — First Out) principle. In its simplest form, the system consists of three components. There is the producer, sending messages to the queue, the consumer receiving them from the queue, and the queue itself.

Log-based

As it follows from the name, the main difference of log-based message brokers is the usage of the log as a store for messages. Log is persistent storage and therefore several consumers can read from it in parallel.

# Getting Started with MongoDB Atlas for Semantic Search

Vector search is an advanced technique used to perform semantic searches, where data is searched based on its meaning rather than the data itself.

This search method utilizes Machine Learning models to effectively search unstructured data, including text, audio, video, and images. It allows finding items that are similar or related to the search item. It is used for several use cases, like recommendation systems, chatbots, or search engines.

When dealing with text data, vector search makes finding words or phrases of similar meaning possible, even if the exact query words are not in the searched sentences.

# Pydantic 2.0 just released! An overview of the most popular data validation python package

Chances are you have used an application that made use of Pydantic, the world’s most popular data validation framework, without even knowing it. In this post, we will cover the basics of Pydantic, what it is used for, and what has changed in the recent 2.0 release. If you are already a Pydantic user, you can skip the intro sections and head to the latter discussion of what has changed.

For full transparency, the main reason I became aware Pydantic even existed is because function calling in the OpenAI ChatCompletions API takes in JSON Schema which Pydantic outputs.

The basic use case for Pydantic is that it allows you to validate data by defining the model. This can be particularly useful if your program expects some specific structured input or output.

# Automating Web Scraping with ChatGPT Code Interpreter

In previous articles, I showed you a workaround to scrape websites with ChatGPT and ChatGPT's Plugin "Scraper." They're good but have some limitations and drawbacks.

But ChatGPT Code Interpreter takes web scraping to a different level.

In this article, we'll see how to automate web scraping with ChatGPT Code Interpreter.

To scrape websites with ChatGPT Code Interpreter, we have to upload an HTML file of the target website.

Then Upload the HTML file and use this prompt. Code Interpreter is only available to ChatGPT Plus subscribers. In case you don't have it enabled yet, follow the steps in this guide to activate it.

# Node.js for Real-Time Communication

In today's digital world, real-time communication is essential. Whether it's a chat application or a live sports update, real-time communication is necessary to keep users engaged. Node.js is a popular tool for developing real-time applications because of its speed, scalability, and reliability. In this article, we'll explore why Node.js is the ideal choice for real-time communication and how to implement it.

Node.js is built on top of Google's V8 JavaScript engine, which is known for its high-performance capabilities. This makes Node.js a perfect tool for building real-time communication applications that require speed and scalability. Node.js is also event-driven, which means it can handle multiple connections simultaneously, making it perfect for building real-time applications.

Node.js provides various ways to implement real-time data communication. Some of the popular ways of real-time data communication in Node.js are: Socket.IO, WebSockets, Server-Sent Events, WebRTC, MQTT or etc.

<https://medium.com/@ben.dev.io/node-js-for-real-time-communication-cf71f985f983>

# Asynchronous Code Execution with Google Cloud Tasks

No application is an island — every non-trivial application has dependencies and exists alongside other services, sharing resources and data. Over the years, messaging has evolved from static, synchronous point-to-point models to asynchronous mechanisms.

In this blogpost, we'll learn what Google Cloud Tasks, an asynchronous point-to-point queuing system for code execution, where you could benefit from using Cloud Tasks, and how to get started.

What is Cloud Tasks?

Cloud Tasks is a queuing system service for Google Cloud Platform for managing the execution, dispatch, and delivery of a large number of distributed tasks.

With Cloud Tasks, you can add millions of "Tasks" in a queue. You may also create many queues. The queue processes Tasks in the queue, sending an HTTP request to the target URL or executing an App Engine application handler, depending on your setup.

# Saving Big on AWS: Best Practices for Cost Optimization and Efficiency

Looking to optimize your AWS costs? In this blog, we'll share the best practices for AWS cost optimization, including tips and strategies for achieving optimal cost savings and improving your efficiency. Whether you're a new AWS user or an experienced one, these insights will help you reduce your cloud expenses while leveraging the best features on the cloud.

✅ Use AWS EC2 On-Demand, Reserved, or Spot instances

Instead of always relying on Dedicated instances, try these cost-effective options as well. On-demand instances provide flexibility and no upfront costs, spot instances offer potential cost savings for non-critical workloads, and reserved instances are ideal for applications with predictable or steady-state workloads.

# What is Redis and How to Use it with Nest.JS

Today we will talk about Redis. A very powerful and easy database to use. It is very popular for caching.

We at our company are building a property management solution where we have to maintain many users and they can have various permissions based on their role.

So, after each successful login client makes a request to get the user profile details along with the permissions for them. As the query is very complex which requires the joining of multiple tables it took a long time to load.

And the users have to wait a long time before seeing anything on their screen which is a very bad user experience.

So, after digging through some possible solutions, we found that Caching is the answer for us. Because of 2 reasons mainly

- This API is called very frequently.

- Users permissions don't change so frequently

So, after some digging, it was clear that Redis might be the best way to go for us. And Redis is a good choice for caching.

NestJS makes it really easy to use Redis.

# Express vs Fastify: Hello world use case

Express is undoubtedly the king of Node.js's web frameworks. Express was first released in 2010 as one of the initial packages on NPM. Express is as strong as ever, even after 12 years. The NPM downloads (~23M weekly) is the proof:

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy. Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Fastify is a relatively new framework for Node.js. It is yet another framework that claims itself to be fast. Launched in 2016, fastify is about 6 years old now. As the name indicates, the purpose of Fastify is to offer a feature rich framework that's fast.

Fastify is a web framework highly focused on providing the best developer experience with the least overhead and a powerful plugin architecture, inspired by Hapi and Express. As far as we know, it is one of the fastest web frameworks in town.

# Stop Being a Junior

Naturally the company likely trusts the seniors with more than they trust you and you don't want to overstep, but volunteer to participate in the more complicated tasks. Even if it's just "hey, can I come sit in on that meeting?" Take notes for yourself of anything you're unfamiliar with and ask about those things later. In future meetings you'll be able to contribute more and more with the knowledge you start to accumulate. Study things out a bit after the meeting and make suggestions.

Beware of being over-bearing. Even though you can accumulate a lot of modern knowledge really fast, you should be sure to acknowledge that your experience limits the usefulness of that knowledge.

Use that fast-paced technology to your advantage and learn about what's new and improved that developers at your company can take advantage of today.

Make sure your manager understands your goals and intentions as a developer. If you're hoping to ask for a promotion in the meeting where your manager tells you about whether you get a pay raise, you're asking way too late. They should know your goals much earlier than that and you can ask them what they expect you to be able to do to get to that level.

# Routers

Routers are modules that take in a user query and a set of "choices" (defined by metadata), and returns one or more selected choices.

They can be used on their own (as "selector modules"), or used as a query engine or retriever (e.g. on top of other query engines/retrievers).

They are simple but powerful modules that use LLMs for decision making capabilities.

They can be used for the following use cases and more:

Selecting the right data source among a diverse range of data sources

Deciding whether to do summarization (e.g. using list index query engine) or semantic search (e.g. using vector index query engine)

Deciding whether to "try" out a bunch of choices at once and combine the results (using multi-routing capabilities).

# Use Nginx as an API Gateway

In this article, we will utilize Nginx as the API gateway to manage and direct incoming requests to three different microservices: user-service, product-service, and order-service. The primary role of the API gateway (Nginx) is to act as an intermediary between clients (e.g., web or mobile applications) and these microservices. Instead of the clients directly accessing each microservice, they will send their requests to the API gateway, which will then intelligently route them to the appropriate microservice based on the request's content and destination.

In simple words, Nginx is software that works as a reverse proxy and can be used as a Load balancer to distribute requests to different servers. Nginx helps you to build a web server.

Here, we would use Nginx as the central traffic controller, handling all incoming requests and making sure they reach the relevant microservice to process the requested actions or data. This approach allows for better management, security, and scalability in a microservices-based architecture.

Leveraging Nginx as an API gateway streamlines interactions between clients and backend services, enabling developers to build scalable and efficient microservices applications with ease.

# Lazy Loading In JavaScript

Web users have high expectations regarding website loading times and performance. Websites that are slow to load can increase bounce rates and dissatisfied users. To address this challenge, developers are constantly seeking different techniques to improve the speed and overall user experience, and one such approach is "Lazy Loading." To achieve lazy loading, developers use JavaScript. Using JavaScript, web developers can control when and how specific elements are fetched from the server and rendered on the user's screen. This article will look into the benefits of lazy loading, ways to implement it, its impacts on web performance, challenges, and best practices.

Lazy loading in JavaScript can be achieved using different methods. Still, two widely used techniques are using the Intersection Observer API to lazy load images and implementing lazy loading of content on the scroll event.

# Backend Projects for Transitioning to a Senior-Level Developer

Transitioning from a mid-level to a senior-level developer requires honing your skills and gaining hands-on experience in building complex backend systems. As a backend developer, you play a crucial role in designing and implementing the server-side logic that powers applications. In this article, we will explore a range of project ideas that can help you level up your backend development skills, demonstrate your proficiency in handling advanced concepts, and pave the way for your transition to a senior-level developer role.

Table of Contents:

1. RESTful API with Authentication and Authorization

2. Task Scheduling and Background Jobs with a Job Queue

3. Real-time Chat Application with WebSockets

4. Microservices Architecture with Communication Protocols

5. Scalable and Fault-Tolerant Distributed Systems

6. Building a Content Delivery Network (CDN)

7. Implementing a Caching Layer for Performance Optimization

8. Designing and Optimizing Relational Database Schemas

9. Building a Web Scraping and Data Processing Service

10. Creating an Analytics Dashboard with Data Visualization

Each project idea will allow you to explore different aspects of backend development, including API design, authentication, data processing, scalability, and performance optimization. Let's delve into the details of each project:

# LlamaIndex: Use the Power of LLMs on Your Data

Everyone has been talking about Large Language Models (LLMs). ChatGPT shows the power of this phenomenal technology in an impressive way. LLMs are pre-trained on a large amount of data. We can use it for our data.

But how we best augment LLMs with our data?

In this case, we need a toolkit to help us. LlamaIndex (previous GPT-Index) is well fit for this. It's a simple and flexible data framework that helps you to connect your data to LLMs. You can connect your unstructured, structured, or semi-structured data sources.

We'll discuss the following points in this article:

1. Ingesting your data

2.Indexing your data

3. Query over your data

With LlamaIndex, you can connect your custom data sources to LLMs. The framework is very user-friendly. There are a high-level API for beginners and a low-level API for the advanced user. In this tutorial, we'll focus on the high-level API. In addition, the framework allows you to build powerful end-user applications. You can integrate LlamaIndex easily into your existing tech stack (e.g., LangChain, Flask, Docker, or anything else).

# Yes, here are 4 stacks to build MVPs faster in 2023

Whether you are a full-time Indie Hacker, a Freelancer or just trying to make a side project, you need to ship and launch your product faster.

I've been building personal projects since 2011 and, the more I tried to do things in a complicated way, the less I shipped and launched my projects.

The key to getting shit done is to build an MVP (minimum viable product) or an MLP (minimum lovable product), and build it fast.

The faster you launch, the faster you can test your idea with users.

But building fast doesn't mean being sloppy.

It just means using the best stack for your idea and context.

I've built and launched all types of products:

1. Marketplaces & custom e-commerce

2. Social networks & community-based apps

3. Chrome extensions, plugins & mobile apps

4. Frontend only web apps (yes, that's possible)

Today, in 2023, I can say without a doubt that I've found the 4 best stacks you can use to build your MVP or MLP faster.

1. Directus CMS

2. Firebase and Firestore

3. Supabase

4. Laravel + Livewire (or Laravel as an API)

# Web Architecture

1. DNS

DNS stands for "Domain Name System" and it's a backbone technology that makes the world wide web possible. At the most basic level DNS provides a key/value lookup from a domain name (e.g., google.com) to an IP address (e.g., 85.129.83.120), which is required in order for your computer to route a request to the appropriate server.

2. Load Balancer

Before diving into details on load balancing, we need to take a step back to discuss horizontal vs. vertical application scaling. What are they and what's the difference? Very simply put in this StackOverflow post, horizontal scaling means that you scale by adding more machines into your pool of resources whereas "vertical" scaling means that you scale by adding more power (e.g., CPU, RAM) to an existing machine.

3. Web Application Servers

At a high level web application servers are relatively simple to describe. They execute the core business logic that handles a user's request and sends back HTML to the user's browser.

4. Database Servers

Every modern web application leverages one or more databases to store information. Databases provide ways of defining your data structures, inserting new data, finding existing data, updating or deleting existing data, performing computations across the data, and more.

5. Caching Service

A caching service provides a simple key/value data store that makes it possible to save and lookup information in close to O(1) time. Applications typically leverage caching services to save the results of expensive computations so that it's possible to retrieve the results from the cache instead of recomputing them the next time they're needed.

6. Job Queue & Servers

Most web applications need to do some work asynchronously behind the scenes that's not directly associated with responding to a user's request. For instance, Google needs to crawl and index the entire internet in order to return search results. It does not do this every time you search. Instead, it crawls the web asynchronously, updating the search indexes along the way.

7. Full-text Search Service

Many if not most web apps support some sort of search feature where a user provides a text input (often called a "query") and the app returns the most "relevant" results. The technology powering this functionality is typically referred to as "full-text search", which leverages an inverted index to quickly look up documents that contain the query keywords.

8. Services

Once an app reaches a certain scale, there will likely be certain "services" that are carved out to run as separate applications. They're not exposed to the external world but the app and other services interact with them.

9. Data

Today, companies live and die based on how well they harness data. Almost every app these days, once it reaches a certain scale, leverages a data pipeline to ensure that data can be collected, stored, and analyzed.

10. Cloud storage

"Cloud storage is a simple and scalable way to store, access, and share data over the Internet" according to AWS. You can use it to store and access more or less anything you'd store on a local file system with the benefits of being able to interact with it via a RESTful API over HTTP.

11. CDN

CDN stands for "Content Delivery Network" and the technology provides a way of serving assets such as static HTML, CSS, Javascript, and images over the web much faster than serving them from a single origin server. It works by distributing the content across many "edge" servers around the world so that users end up downloading assets from the "edge" servers instead of the origin server.

# Why You Need a Knowledge Graph, And How to Build It

A knowledge graph organizes events, people, resources, and documents in a graph database for advanced analysis. This article will explain the purpose of a knowledge graph and show you the basics of how to translate a relational data model into a graph model, load the data into a graph database, and write some sample graph queries.

Why a Knowledge Graph?

Relational databases are great for creating lists, but terrible for managing networks of diverse entities. Have you ever tried to do any of these tasks with a relational database?

- analyze a healthcare episode of care when a patient interacted with dozens of people, places and procedures

- find patterns in financial fraud with a web of vendors, customers and transaction types involved

- optimize the dependencies and interconnected elements of a supply chain

In the future, we should expect to see enterprise data groups adopting a combination of relational databases for isolated analysis on one business function, and knowledge graphs for complex, networked processes that span functions.

A knowledge graph, based in graph database technology, is built to handle a diverse network of processes and entities. In a knowledge graph, you have nodes that represent people, events, places, resources, documents, etc. And you have relationships (edges) that represent links between the nodes. The relationships are physically stored in the database with a name and direction. Not every graph database is a knowledge graph.

# Angular is getting New Template Syntax

New Template Syntax, Built-In Control Flow, a farewell to structural directives? Not a lot has been changing in recent years. Angular has been stable for some devs and stagnant for others. Now it's moving forward at light speed. But where exactly is it headed?

Angular proposes a transition from the current structural directives (NgIf, NgForOf, NgSwitch) to a new built-in syntax. If you're not already following the RFCs (Request for Comments), please do. In meantime I am going to help you understand what's coming.

Revamping Control Flow

Angular team aims to replace the existing structural directives (NgIf, NgForOf, and NgSwitch) with a more modern, macro-like syntax. While the structural directives are not going away completely, because the concept is going to stay — this new way of writing your template is going to be the preferred one.

If-Else Blocks

If you found the original way of working with ng-container & ng-template alongside \*ngIf structural directive not intuitive enough — New Control Flow might have an answer for your troubles.

Switch Block

The 'switch' block takes place of \*ngSwitch which. New way of writing is said to bring in substantial benefits like enhanced template type-checking and no need for container elements to hold condition expressions.

What About Migration?

Migration to the new syntax is promised be relatively smooth. Angular team is working on an automated migration schematic to convert from the old to the new syntax.

Future Opportunities

The Angular team envisions extending this new syntax to accommodate more JS loop flavors, including async iteration, and for-in loops. Potential future improvements also include virtual scrolling and destructuring support.

# How to Handle a Million Requests in a Day Using Django

Django is a popular web framework for building scalable and maintainable web applications. However, as the userbase of your application grows, you may find that it is struggling to handle a large number of requests. This was the case for us at Dixiapp, a Slack-based application for automating asynchronous meetings and standups. As our userbase scaled up, we started getting a lot of traffic and needed to find ways to handle the increased load.

In this blog post, we will explore some of the techniques that we used to handle a million requests in a day using Django. We will cover how to choose a suitable database, use database partitioning, caching, indexes, and database sharding to optimize our application's performance and efficiently store and retrieve data. By implementing these techniques, we were able to improve our application's speed, reliability, and scalability, and ensure that we could handle the increased traffic from our growing userbase.

Choosing a suitable database

Choosing the right database is an important factor in ensuring that your Django application can handle a large number of requests. There are many types of databases available, such as SQL, NoSQL, and graph databases, each with their own strengths and weaknesses. When choosing a database, it's important to consider factors such as data structure, scalability, and performance.

Using database partitioning

Database partitioning is a technique used to split a large database table into smaller, more manageable pieces. By partitioning a table, we can distribute the load across multiple servers and improve the performance of our application.

Caching

Caching is another technique that we can use to improve the performance of our Django application. By caching frequently accessed data, we can reduce the load on the database and improve the response time of our application.

Indexing

Indexing is a database technique used to improve the performance of queries by creating an index on one or more columns in a table. An index allows the database to quickly locate and retrieve the data that matches a query.

Database Sharding

Database sharding is a technique used to horizontally partition a database across multiple servers. By sharding a database, we can distribute the load across multiple servers and improve the performance of our application.

We can use partitioning to distribute the load across multiple servers, caching to reduce the load on the database, indexing to improve the performance of queries, and database sharding to horizontally partition the database across multiple servers.

# Job Queue & Servers

Most web applications need to do some work asynchronously behind the scenes that's not directly associated with responding to a user's request. For instance, Google needs to crawl and index the entire internet in order to return search results. It does not do this every time you search. Instead, it crawls the web asynchronously, updating the search indexes along the way.

While there are different architectures that enable asynchronous work to be done, the most ubiquitous is what I'll call the "job queue" architecture. It consists of two components: a queue of "jobs" that need to be run and one or more job servers (often called "workers") that run the jobs in the queue.

Job queues store a list of jobs that need to be run asynchronously. The simplest are first-in-first-out (FIFO) queues though most applications end up needing some sort of priority queuing system. Whenever the app needs a job to be run, either on some sort of regular schedule or as determined by user actions, it simply adds the appropriate job to the queue.

Storyblocks, for instance, leverages a job queue to power a lot of the behind-the-scenes work required to support our marketplaces.

We run jobs to encode videos and photos, process CSVs for metadata tagging, aggregate user statistics, send password reset emails, and more.

We started with a simple FIFO queue though we upgraded to a priority queue to ensure that time-sensitive operations like sending password reset emails were completed ASAP. Job servers process jobs. They poll the job queue to determine if there's work to do and if there is, they pop a job off the queue and execute it. The underlying languages and frameworks choices are as numerous as for web servers so I won't dive into detail in this article.

# Building a Real-Time Database Notification Service with Change Tracking in C#

We will explore the implementation of a powerful database notification service in C#, capable of providing real-time updates on data changes in a SQL Server table. By leveraging SQL Server's Change Tracking feature, we can efficiently monitor and respond to modifications without the need for continuous polling, resulting in a more responsive and resource-efficient application.

SQL Table Change Notification is required for real-time data synchronization and event-driven applications. It provides a mechanism for applications to be notified when changes occur in a specific SQL database table, allowing them to take appropriate actions based on those changes. There are several reasons why SQL Table Change Notification is essential:

In distributed systems or scenarios where multiple instances of an application are running,

- Modern applications require real-time data updates to provide timely and accurate information to users.

- Keeping data in sync across all nodes can be challenging.

- Caching is used to improve application performance, but it can lead to stale data if not managed properly.

- Where multiple users can access and modify the same data to resolve conflicts .

- Where applications offer subscription-based services in delivering timely and relevant information to subscribers.

Overall, SQL Table Change Notification enhances application responsiveness and enables efficient data synchronization, making it a valuable feature for various types of applications that rely on real-time data updates and event-driven behavior.

# PHP 8.2: A Glimpse into the Future of Web Development

As the digital landscape continues to evolve, PHP, one of the most popular programming languages for web development, stays ahead of the curve with its latest version — PHP 8.2. Packed with exciting features and enhancements, PHP 8.2 takes the language to new heights, empowering developers to build faster, more secure, and feature-rich web applications. In this article, we'll explore some of the most captivating features of PHP 8.2 and how they can revolutionize the way we code.

PHP 8.2 brings a bouquet of exciting features and enhancements, making it an indispensable tool for modern web development. With improvements in FFI, Tracing JIT, and Fibers, PHP is now more performant, scalable, and capable of handling sophisticated applications. The enhancements to date and time functions, along with the introduction of enumerations, improve code quality and readability, fostering a more enjoyable development experience. As PHP 8.2 gains momentum, developers are empowered to create cutting-edge web applications that deliver unparalleled performance and user experiences. Embrace PHP 8.2 and embark on a journey into the future of web development!

# How to Chat With Any PDFs and Image Files Using Large Language Models — With Code

So much valuable information is trapped in PDF and image files. Luckily, we have these powerful brains capable of processing those files to find specific information, which in fact is great.

That is the whole purpose of this article. I will explain step-by-step how to build a system that can chat with any PDFs and image files.

General Workflow of the project

It's always good to have a clear understanding of the main components of the system being built. So let's get started.

- First, the user submits the document to be processed, which can be in PDF or image format.

- A second module is used to detect the format of the file so that the relevant content extraction function is applied.

- The content of the document is then split into multiple chunks using the Data Splitter module.

- Those chunks are finally transformed into embeddings using the Chunk Transformer before they are stored in the vector store.

- At the end of the process, the user's query is used to find relevant chunks containing the answer to that query, and the result is returned as a JSON to the user.

# Building a RAG-based Conversational Chatbot with Langflow and Streamlit

According to its creator, LogSpace, a software company that provides customized Machine Learning services, Langflow is a web-based UI for LangChain designed with react-flow. Langflow provides an effortless way to experiment and prototype flows. LangChain, created by Harrison Chase, is a wildly popular framework for developing applications powered by large language models (LLMs).

Langflow allows users to quickly develop simple to complex LangChain prototypes without requiring coding skills, truly democratizing LLM access. According to the project's documentation, "Creating flows with Langflow is easy. Simply drag sidebar components onto the canvas and connect them to create your pipeline. Langflow provides a range of LangChain components, including LLMs, prompt serializers, agents, and chains." I recently published the blog, Turn Your Langflow Prototype into a Streamlit Chatbot Application, which explains how to use Langflow and turn prototypes into Streamlit applications.

In this post, I will walk you through how to build a conversational chatbot to answer wine-related questions, a "Virtual Sommelier" if you will, using Retrieval Augmented Generation (RAG), in 20 minutes or less with almost little to no coding required.

# Using LLaMA 2.0, FAISS and LangChain for Question-Answering on Your Own Data

Over the past few weeks, I have been playing around with several large language models (LLMs) and exploring their potential with all sorts of methods available on the internet, but now it's time for me to share what I have learned so far!

I was super excited to know that Meta released the next generation of its open-source large language model, LLaMA 2 (on 18th July 2023) and the most interesting part of the release was, they made it available free of charge for commercial use to the public. Therefore, I decided to try it out and see how its performs.

In this article, I'm going share on how I performed Question-Answering (QA) like a chatbot using Llama-2–7b-chat model with LangChain framework and FAISS library over the documents which I fetched online from Databricks documentation website.

LLaMA 2 model is pretrained and fine-tuned with 2 Trillion 🚀 tokens and 7 to 70 Billion parameters which makes it one of the powerful open source models. It comes in three different model sizes (i.e. 7B, 13B and 70B) with significant improvements over the Llama 1 models, including being trained on 40% more tokens, having a much longer context length (4k tokens 🤯), and using grouped-query attention for fast inference of the 70B model 🔥. It outperforms other open source LLMs on many external benchmarks, including reasoning, coding, proficiency, and knowledge tests.

# REST API vs. GraphQL: Unraveling the Web of APIs

In the vast landscape of web development, two superheroes have emerged to battle it out for the crown of API supremacy: REST and GraphQL. 🦸‍♂️💥 But fear not, fellow developers, for in this blog, we'll break down the battle between these two contenders and help you decide which one suits your needs best. Let's dive into the showdown of REST API vs. GraphQL! 🥊🤖

REST API (Representational State Transfer) 📡

REST has been the reigning champ for years. It relies on HTTP methods to perform actions on resources using simple and predictable endpoints. Each endpoint represents a resource, and CRUD operations are mapped to HTTP verbs like GET, POST, PUT, and DELETE.

GraphQL 🕸️

GraphQL, the challenger, arrived with a fresh perspective. It allows clients to request exactly what data they need, and nothing more. Instead of multiple endpoints, GraphQL offers a single endpoint for querying and mutating data. This flexible and efficient approach has gained popularity rapidly.

In the epic showdown between REST API and GraphQL, there's no definitive winner. The choice between the two depends on your project's requirements, complexity, and performance considerations. REST API remains a solid choice for simpler applications, while GraphQL shines in scenarios where flexibility and efficiency are paramount.

So, whether you're cheering for the RESTful veteran or the GraphQL upstart, remember that both have their strengths and weaknesses. Choose wisely, and may your APIs be ever efficient, and your endpoints be ever RESTful or GraphQLish! 🥇🚀

# Manage Serverless APIs With API Gateway in GCP

In Google Cloud Next 2020, new API management service has been introduced. Google Cloud API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor and secure APIs. It acts as "Front Door" for an application deployed on backend service like Cloud Functions, Cloud Run and App Engine, Compute Engine and Google Kubernetes Engine.

In this tutorial, we are going to deploy Employee APIs endpoints in Cloud Function, Cloud Run and App Engine Standard environment with front end proxy by API Gateway. These APIs will be secured with API Keys and Oauth2 Bearer Token.

Architecture

We are aiming to create an app with 3 different services using the following components.

Cloud Functions: GET an Employee from Firestore using API Key.

Cloud Run: Add an Employee into Firestore using Bearer Token.

App Engine: DELETE an Employee from Firestore using Bearer Token.

API Gateway: Proxy service to control the requests.

Firestore: Store Employee data.

# Why I definitively switched from Cloud Functions to Cloud Run

Serverless tools are really easy to use for any developers and you can easily build applications and scale them from 0 to planet scale solution!

In that journey, the project management, deployment, automation and all that industrialisation topics must be efficient to keep the advantage and to continue to grow smoothly.

On Google Cloud, Cloud Functions is (was?) one of the most popular services to deploy a simple piece of code in HTTP mode or in Background mode (answer to events, like PubSub messages or Cloud Storage events).

However, the introduction of Cloud Run in 2019 and the very fast and very good feature additions to the product lead me to reconsider the use of Cloud Functions and, slowly, lead me to abandon it.

However, something better, stronger, easier and more portable exists now. It's simply obvious to use it.

More and more developers are familiar with containers and it's not an additional cost to use them now.Using a single one product, lead to other advantages: You have to train your team on a single product, and you no longer have the question to choose between Cloud Functions or Cloud Run. Finally, processes and best practices are fewer (because of only one product) and you increase your overall efficiency.

Like the Darwin evolution model, it's not the weaker that disappear, it's only the most adapted to their environment that dominate and extinct the other species.

# Introduction to Abstract Syntax Trees in Python

When working with Python, it is essential to have a clear understanding of the code's structure and how it works. This is where Abstract Syntax Trees (AST) come into play. An AST is a tree-like structure that represents the syntactic structure of a program, making it a powerful tool for Python programmers who want to take their coding skills to the next level. By understanding how Python code is parsed, developers can efficiently analyze their code leading to more effective and efficient programming.

An Abstract Syntax Tree (AST) is a hierarchical, tree-like data structure that represents the syntactic structure of a program. It serves as an intermediate representation of the code and is generated by a compiler or interpreter. The AST captures the essential elements of the program's syntax, providing a structured representation of operators, functions, statements, and expressions. The AST is constructed by parsing the source code of a program. During the parsing process, the code is analyzed and broken down into its constituent parts, such as keywords, identifiers, literals, and operators. These parts are then organized hierarchically in the form of a tree, with the main program as the root node and the various code elements as its child nodes. The AST effectively captures the structure of the code by representing relationships between different code elements. For example, function calls are represented as child nodes of the corresponding function definitions, and expressions within statements are represented as child nodes of those statements. This hierarchical representation allows for easy traversal and analysis of the code's structure. To obtain an AST from source code, a parser is used. The parser takes the source code as input and performs lexical analysis and syntactic parsing to identify the different elements and their relationships. The parser follows the grammar rules of the programming language to generate the corresponding AST.

# Simplifying your Kubernetes infrastructure with cdk8s

CDK for Kubernetes, or cdk8s is an open-source CNCF project that helps represent Kubernetes resources and application as code (not YAML!)

Working on Kubernetes?

Many of you work on Kubernetes, perhaps in different capacities — Architect, DevOps engineer, SRE, Platform engineer, Software engineer etc. No matter what we tell our friends, family or even recruiters (we need to convince them how hard it is to work across the entire CNCF landscape, isn't it? 😉 )

… YAML everywhere

It has become the "lingua franca" of the Kubernetes world. Every tool set that we use including Helm, kustomize, GitOps, Kubernetes operators etc. involves using (and debugging) YAML.

From Infra-as-code to Infra-Is-code

You may have used Terraform or AWS CloudFormation. These fall into the Infrastructure as Code (IaC) category, where you write configuration to define your resources — be it YAML or JSON or a custom templating language.

Hello cdk8s!

And that's where cdk8s comes into the picture since it embraces the Infrastructure Is code paradigm for Kubernetes applications. I am a Go developer and that's what I used for the demos and code samples, but cdk8s supports Python, Java and Javascript.

# The S.O.L.I.D Principles in Pictures

There are so many great articles online about SOLID but I rarely see any examples with pictures. This makes it a bit difficult for visual learners like me to learn while staying engaged.

You see, some of these principles may look similar but they are not targeting the same goal. It is possible to satisfy one principle while violating the other, even though they are alike.

S — Single Responsibility

If a Class has many responsibilities, it increases the possibility of bugs because making changes to one of its responsibilities, could affect the other ones without you knowing.

O — Open-Closed

Changing the current behaviour of a Class will affect all the systems using that Class.

If you want the Class to perform more functions, the ideal approach is to add to the functions that already exist NOT change them.

L — Liskov Substitution

If you have a Class and create another Class from it, it becomes a parent and the new Class becomes a child. The child Class should be able to do everything the parent Class can do. This process is called Inheritance.

I — Interface Segregation

When a Class is required to perform actions that are not useful, it is wasteful and may produce unexpected bugs if the Class does not have the ability to perform those actions.

D — Dependency Inversion

This principle aims at reducing the dependency of a high-level Class on the low-level Class by introducing an interface.

# Secure database connection in Nodejs application

Securing database connections is an essential part of building secure Node.js applications. Here are some best practices for handling secure database connections in Node.js

1. Use environment variables

Store sensitive database connection information (such as usernames, passwords, and connection strings) in environment variables rather than hard-coding them in your application code. This helps protect your application from accidental exposure of sensitive information as well as giving flexibility in changing the credentials depending on the environment that we are using (dev, staging, or production).

2. Use encrypted connections

When you transmit data over an unencrypted connection, the data can be intercepted by attackers who can then read, modify, or steal it. This is especially important when sensitive data, such as usernames and passwords, financial data, or personal information, is being transmitted.

Using encrypted connections can help protect your data from these threats. Encrypted connections use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols to encrypt the data in transit, making it more difficult for attackers to intercept and access the data.

3. Use connection pooling

Connection pooling is a technique used to optimize the performance of applications that use a database. When an application needs to connect to a database, it creates a new connection to the database server.

Creating a new connection is a relatively expensive operation in terms of time and resources, so connection pooling aims to minimize the number of connections that are created by reusing existing connections whenever possible. Libraries like pg and mysql2 for PostgreSQL and MySQL, respectively, provide built-in connection pooling.

4. Limit access to your database server

Configure your database server to only accept connections from authorized IP addresses, and use firewalls and other security measures to restrict access to your database server

# Event Storming

Traditionally, when we designed software systems, we would begin by focussing on a data model and then build an app around it. Unfortunately, as software engineers, this is something that we are inherently programmed to do. We would often get attached to these data models and bend our apps and even our requirements around them to make them fit.

Let's save some time and skip to the outcome: we end up building something that does not match our requirements and is not fit for purpose.

This old way of doing things does not work (most of the time). This is why techniques such as domain-driven design, code-first data modelling, and Event Storming came along to the rescue! First, these techniques focus on the problem space and desired behaviours of our system, then look at how we model this in a data/database model at the very end of the design process. Rather than 'finding a problem for a solution,' we are designing our system around the specific requirements for the problem we are trying to solve.

# Understanding the Crucial Difference: await vs. return await

Asynchronous programming is an essential aspect of modern JavaScript development, allowing us to handle time-consuming operations without blocking the execution of other tasks. When working with async functions, we encounter three keywords that play crucial roles: await, return, and return await. In this article, we'll explore the differences between these keywords and discuss when to use each one.

Before diving into the specifics, let's clarify the purpose of async functions. An async function is a special type of function that enables the use of the await keyword. It allows us to write asynchronous code in a more synchronous and readable manner, making it easier to handle promises and perform non-blocking operations. When an async function is invoked, it returns a promise that resolves to the function's eventual result.

# Real-Time Bidding Engine Using NestJS and Socket IO

"Goin once, Goin twice, sold for this price", yes You heard it right we are talking about bidding. We all have heard about this in movies or live auctions, with people competing against each other in auctions to win the ultimate title. Several platforms are offering real-time bidding for online auctions. In this article we will be talking about the mechanism behind real-time bidding, how it works and the development of it using Nest.js and Socket IO.

What is Socket.IO?

Socket.io is a popular JavaScript library that allows us to create real-time, bi-directional communication between web browsers and a server. It is a highly performant and reliable library designed to process a large volume of data. It follows the WebSocket protocol and provides better functionalities, which enables us to build efficient real-time applications.

Publish-Subscribe Design Pattern

Publish-Subscribe also known as pub/sub is a commonly used design pattern for real-time communication between the client and the server. It allows the server to send messages to clients through a channel/medium in real time. The senders of these messages (publishers) do not explicitly identify the targeted recipients. Instead, the messages are sent out on a channel on which any number of recipients (subscribers) can be waiting for them.

Developing a JSON Web Token (JWT)Authentication using Spring Security.  
In this tutorial, you will learn how to implement JWT (JSON Web Token) authentication system using Spring Boot and Spring security. First I hope to explain some basics theories regarding to this tutorial. Then I'll present in detail how to develop the application step by step and at the end I'll also provide my Git Hub repository link.

What is authentication ?

Authentication is the process of identifying users that request access to a system, network, or device. Access control often determines user identity according to credentials like username and password.

What is JWT (JSON Web Token) ?

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. It is commonly used for authentication and authorization purposes in web applications and APIs. JWTs are often used in scenarios where information needs to be verified and trusted between different parties, typically involving a client (such as a user's browser) and a server.

# Mastering Database Replication: An Essential Guide for 2023

In our interconnected world, data is the heart of many systems. Whether you're browsing your favorite online store, transferring money via a mobile app, or liking a friend's post on social media, you're interacting with systems that rely heavily on data. But what happens if that data becomes unavailable? Enter: replication.

The Growing Need for Replication

Imagine you're running an e-commerce site. Your servers are based in New York, and you've just started getting a significant number of customers from Australia. The time it takes for a user in Sydney to retrieve product details from your New York server might be longer than you'd want. Slow access can turn potential buyers away.

Understanding the Primary-Replica Relationship

In the world of replication, terms like "primary" and "replica" are often thrown around. Let's demystify them.

Replication Defined

At its core, replication is the act of duplicating data from a main source (often called the "primary" database) to one or more secondary sources (the "replica" databases). Think of it as taking a book and creating multiple copies of it. If the original book gets damaged or lost, the information isn't gone; you still have the copies.

Key Benefits of Replication

The world of databases might seem convoluted, but the principles guiding them often stem from real-world needs. Replication, for instance, didn't come into existence on a whim; it solves specific challenges. Let's shed light on the core advantages of implementing replication in your system.

At its core, replication is akin to a well-choreographed dance. Different servers (or dancers) move in harmony, ensuring the data (or the dance narrative) remains consistent, available, and impactful. As businesses, technologists, or even curious learners, understanding this dance — its moves, its challenges, and its future — is instrumental. As we step into a world even more intertwined with data, here's to mastering this dance and making the most of the opportunities it unfurls!

# How I became a Google Developer Expert in Flutter

Google has a program for experts in its technologies. To get there, an expert must speak, teach, mentor, or develop open-source software. The program is called "Google Developer Experts" and is the most privileged community for developers that I know of.

My path with Flutter

In December 2020, I was working on a startup and needed the cheapest possible frontend to build mobile apps and web from a single code base because I was the only developer.

Of all frontend things, I was most experienced in jQuery. My last mobile app was a Tetris in JavaME made in 2006.

That was far from a single code base, so I had to learn something new. I recalled my friend saying Flutter was the next big thing. That was even before Flutter became the most popular cross-platform solution.

I was hoping to only get a part-time job to continue my startup, so I hesitated. But then my friends said I would be nuts to not take that. So I took it. The post was for seniors, but they hired me as a middle Flutter developer because I never learned the letters in "SOLID".

If you doubt whether you qualify with your statistics, ask an existing GDE.

If your statistics for the rolling window of a year are declining, I suggest you do not waste your time. Otherwise, you may miss an opportunity that may take a long time to present again.

# Our backend strategy to handle massive traffic

Compared to 2018, Coupang has quadrupled in revenue with fast growth in active users. In our never-ending mission to "Wow the Customer", we are constantly adding new services such as Rocket Delivery and Rocket Fresh. Although such services give us an edge over our competitors, they require us to develop and maintain an increasingly complex network of data and data systems to serve our 18 million customers.

In this post, we will examine how we serve data from our databases to the Coupang e-commerce application with low latency and high availability through our core serving layer. To read about the lessons we learned from operating our core serving layer, read part 2 of this series.

Background and challenges

In addition to being a marketplace, Coupang purchases items at wholesale and sells them to customers through Rocket Delivery, our own delivery service. The concept is simple in theory, but the business and data logic involved in such services is far more complicated than that of the average e-commerce platform.

As seen in Figure 1, each product contains various types of data, each demarcated with a separate-colored box. Each data type is managed through a discrete microservice architecture in the backend. For instance, the product image and title data is managed by the Catalog microservice and the guaranteed delivery date data by the Stock and Fulfillment microservice.

Furthermore, each data point is personalized to the customer and can change in real time. For example, if the daily outbound capacity of a fulfillment center (FC) is met, the FC cannot take any new customer orders, and the relevant inventory and guaranteed delivery date information must be updated on the frontend for affected customers.

If all of Coupang application's pages each directly called data from the microservices, the microservices would always have to secure high availability, and commonly used business logic code would be duplicated on the frontend without centralized management. For such reasons, we needed a single microservice to handle the commonly used business logic and data.

The core serving layer was our solution to providing a unified and systemic approach to serving data from multiple microservice systems to the Coupang application pages at high availability, high throughput, and low latency. To ensure reusability and standardization across domains, we also distributed a core serving layer template that provides the data serving foundation and unified business logic.

# API Gateway

What is API

API or Application Programming Interface is a software intermediary by which two software components talk to each other. Nowadays APIs are being used everywhere from — making an online transaction — to chatting with friends on WhatsApp — or posting your favorite picture on Instagram — or watching your favorite movie on Netflix and whatnot.

API Management

Due to this enormous use of APIs and the fact that organizations are moving fast from large legacy monolithic applications to small individual microservices, API management has become a need of the hour.

API Gateway is one of the critical API management components of modern application architecture that acts as a single point entry in modern API-enabled applications.

What is API Gateway?

API Gateway is a server that acts as a single point of entry for all client requests. It receives the requests, processes them, and routes them to the appropriate backend services. It can also handle several cross-cutting concerns such as authentication, authorization, protocol translation, rate limiting, caching, load balancing, logging, monitoring etc.

The architecture of API Gateway

1. Client

The client is the entity that sends requests to the API Gateway. It can be a web application, mobile application, Other APIs, IOT devices as described in the left side of the API Gateway in Image 1, or any other application that communicates with the API Gateway.

2. API Gateway

API Gateway is the central component of the architecture. It receives the requests from the client and routes them to the appropriate backend services. It also performs various tasks such as authentication, rate limiting, caching, and monitoring.

3. Backend Services

Backend services are the services that provide the functionality requested by the client. They can be microservices, databases, or any other service that provides the required functionality.

4. Service Registry

Service Registry is a database that contains information about the backend services. It includes the service name, version, and the location of the service.

5. Load Balancer

Load Balancer is a component that distributes the requests to the backend services. It ensures that the requests are evenly distributed among the backend services.

6. Security Layer

The security Layer is a component that provides security to the API Gateway. It includes various security mechanisms such as authentication, authorization, and encryption.

7. Monitoring Layer

Monitoring Layer is a component that monitors the API Gateway and the backend services. It collects various metrics such as response time, error rate, and throughput.

# Multithreading and Concurrency Concepts I Wish I Knew Before the Interview

What is Multithreading?

Imagine you are sitting in your office and you have multiple tasks to do — reading emails, making phone calls, writing reports, meeting colleagues, and so on. Now, if you were to do these tasks one by one, it would take a good amount of time, right? But what if we could do several of these tasks at once? Like reading an email during a dull meeting (we've all been there!), or making a phone call while going through the report. This simultaneous execution of tasks not only makes us more productive but also saves precious time. This is, in the simplest terms, the essence of multithreading.

In the realm of computing, a thread is the smallest sequence of programmed instructions that can be managed independently by a scheduler. Multithreading is the ability of a central processing unit (CPU) (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system.

What is Concurrency?

Now that we've touched upon multithreading, let's introduce another intriguing concept — Concurrency.

Imagine you're cooking a meal. You chop the vegetables, heat the oil, sauté the veggies, and then let them cook. While the veggies are cooking, you start kneading the dough for bread. Here, even though the tasks are related and dependent, you're not waiting for one to finish completely before starting the next. This ability to overlap tasks is called concurrency.

In the world of computing, concurrency is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads running in parallel. The running process threads always communicate with each other through shared memory or message passing. Concurrency results in sharing of resources, in turn making the system efficient.

Why is Multithreading and Concurrency Important?

In today's world, where time is of the essence and efficiency is key, concepts like multithreading and concurrency have gained a lot of traction. They not only help in improving the speed and performance of tasks but also improve the overall system efficiency.

Multithreading helps to achieve parallelism in the programs. We can use it to keep the CPU busy and exploit its full potential. For instance, in a game, one thread could handle the game's graphics, another could handle the user inputs, and yet another could handle the game logic.

# ASP.NET Core: Flexible and Versatile

Core Features and Advantages

ASP.NET Core has evolved into a comprehensive, modern, and cross-platform framework for building web applications, APIs, and cloud-based services. Here are the core features and advantages that make it an attractive choice for developers:

Cross-Platform: One of the major strengths of ASP.NET Core is its ability to run on multiple platforms, including Windows, macOS, and Linux. This versatility offers developers flexibility and widens the scope of application deployment.

Performance: ASP.NET Core is fast. It's optimized for high-performance applications, offering features such as asynchronous programming, which is ideal for tasks like database access, web requests, and file I/O, where operations can run concurrently, enhancing the overall application performance.

Unified MVC & Web API frameworks: ASP.NET Core combines the best of MVC (Model-View-Controller) and Web API frameworks into a single programming model, simplifying the development process for creating web pages and services.

Dependency Injection: Dependency Injection is built into ASP.NET Core, promoting loosely coupled and testable code. It makes applications easier to change and configure.

Middleware Components: Middleware in ASP.NET Core defines how an application responds to HTTP requests. Developers can configure it to add functionalities such as exception handling, logging, and static file serving, among others.

# Flutter vs React Native: Which Framework Wins in 2023?

Flutter or React Native: Usage, Popularity and Examples

To begin, let's determine which framework is better suited for your needs. Additionally, we will provide examples of apps built with both frameworks to help you evaluate their capabilities and potential.

Usage of Flutter and React Native

To determine the best framework for your app, consider the desired features, target devices, and number of screens.

Overall, Flutter is the more suitable option for most cases, but for AR elements, native app development may be a better choice. By the way, you can still create VR apps on Flutter, using our VR Player plugin.

Popularity of Flutter and React Native

Many popular apps are built using either Flutter or React Native, and you likely use them on a daily basis without realizing it. Below are some examples of well-known apps developed using these frameworks.

The exact number of apps built with both frameworks is unknown. However, according to Google statistics, over 500,000 apps have been built with Flutter. React Native, being an older framework, likely has a higher number of built apps.

It is difficult to determine which framework is more popular among businesses. However, Google Trends data illustrates that interest in Flutter is consistently increasing.

Design, UI and Performance Comparison

Flutter's core concept is building user interfaces using widgets. These widgets not only include buttons, text and cards, but also animations, layout options, and touch interactions. The widgets describe their appearance based on their configuration and state. When a widget's state changes, it rebuilds its description and updates its appearance.

Performance Comparison

When it comes to performance, Flutter has an advantage due to its architecture. Unlike React Native, which requires a JavaScript communication bridge to interact with native components, Flutter doesn't need this bridge. Instead, it utilizes the powerful Skia rendering engine. The bridge in React Native can cause a slowdown in performance.

Moreover, Google's experimental Impeller tool in Flutter 3.0 enhances animation and rendering (currently available for iOS only). Tests conducted by our team show considerable performance improvements when using Flutter.

# Stop using AWS Lambda for everything!

Serverless computing has revolutionized how we build and deploy applications in the cloud. It offers scalability, reduced infrastructure management, and cost-efficiency.

Previously, I wrote an article about Serverless does not mean Lambda or FaaS!, but in this article, we will focus on different options for serverless computing in AWS and compare the benefits and pitfalls of using them.

Amazon Web Services (AWS) provides various options for serverless computing and orchestrating these resources. This includes but is not limited to AWS Step functions to potentially handle some logic and orchestrate AWS Lambda, AWS Fargate to manage ECS or EKS responsible for orchestrating containers, and AWS Batch to orchestrate batch jobs. Note that we will not cover AWS Batch in this article.

AWS Lambda with AWS Step Functions

AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers. It is event-driven and scales automatically based on incoming requests. Lambda functions can integrate well with various AWS services, providing a powerful way to coordinate workflows.

AWS ECS with Fargate

AWS Elastic Container Service (ECS) with Fargate is a fully managed container orchestration service that allows you to run containers without operating the underlying infrastructure. Fargate abstracts away the need to provision and manage EC2 instances, providing a simplified experience for deploying and scaling containerized applications.

AWS EKS with Fargate

AWS Elastic Kubernetes Service (EKS) with Fargate is a fully managed Kubernetes service that runs containers without operating underlying infrastructure. EKS provides a scalable and highly available platform for deploying containerized applications using Kubernetes, while Fargate abstracts away the need to provision and manage EC2 instances.

*When choosing between AWS Lambda with/without Step Functions and other serverless options, for example, container orchestrators like AWS ECS managed by Fargate or AWS EKS managed by Fargate, it is essential to consider the specific requirements of your application and the trade-offs associated with each option.*

*Lambda and Step Functions offer simplicity for small workflows but can become quite complex for large applications or lead to the trap of “building your frameworks”. Also, by adopting event-driven architecture, you can quickly achieve scalability but remember that complexity never dies and, most of the time, moves to another layer and grows there. Bear in mind that lambda functions are suitable only for small and short-lived computations, which can lead to too many communications over the network, which is one of the slowest components.*

*On the other hand, ECS with Fargate or EKS with Fargate can provide more customization options, compatibility with existing containerized workloads, and the ability to manage long-running tasks.*

# Microservices Checklist — Cover All Aspects of Microservices Architecture

In this article, we are going to learn our Microservices Checklist and cover All Aspects with exploring vertical topics of Microservices Architecture.

Before design any architecture, we should evaluate and verify some vertical topics and question to our architecture in order to make sure that we have calculated every tradeoffs and risks at acceptable level for your project.

Software Architecture Design

Software architecture is always a balancing act between trade-offs and compromises. There is no single design or architecture that is perfect for every situation. Instead, software architects must carefully consider the trade-offs and make decisions based on the specific requirements, constraints, and goals of the software system.

The key point is that there is no single "best" solution for software architecture, and software architects must constantly make trade-offs and compromises based on the specific requirements, constraints, and goals of the system. The goal is to find the right balance that meets the needs of the system and the stakeholders.

Microservices Building Blocks — Sub Components

Now we can go one more level to identify and check sub components of building blocks. Here you can reach the sub components of these Microservices Building Blocks:

Conclusion

These are created from our team when we design and iterate microservices into our cloud-native enterprise applications and projects. So if you have additional consideration feel free to extend this list with adding new considerations regarding to microservices.

# 7 Useful .NET Libraries You Should Use in Your Next Project

In the digital era, libraries are the levers for developers and .NET is the ideal fulcrum. They take us out of coding boring stuff and allow us to focus on creating unique features.

We all know the power of .NET: It's like a Swiss Army development knife, suitable for everything from fun games to rugged IoT devices. But just as a master craftsman knows the value of his tools, the best .NET developers appreciate top-notch libraries that can turn a good project into a masterpiece.

1. Bogus

As a developer, we use fake data to generate test data for applications or to create sample data for demos and prototypes. This library will help you with this task. It includes support for generating fake data in various formats, including names, addresses, phone numbers, and more. It also makes it easy to generate large amounts of fake data quickly and easily. This library has more than 7.5k stars on GitHub.

2. DbUp

This library will help you to deploy changes to SQL Server databases. It tracks which SQL scripts have been run already, and runs the change scripts that are needed to get your database up to date. It has more than 2k stars on GitHub.

# How to secure your web applications (Part 1) — CPAS 3

If you have a live application used by other users, then there is a high chance that they will be attacked, if not daily, at least once a week.

From my own experience,I have noticed that almost all the applications I monitor are attacked daily, either with an automatic script for finding vulnerabilities or manually.

Almost all the attacks, both manually and automated ones are harmless. They are done by curious users who I presume that do them mostly for fun or to see how the application is made. Most of them look like it follows:

The problem is with the other attacks, that are done by hackers who rather analyze the architecture manually, than with some automatic scripts only for certain languages, and build their attacks according to the vulnerability they discover.

With this in mind I'll show some best practices that you can apply right now in order to build a bulletproof security system for your application. These principles can be applied in any language and environment.

For demo purposes I'll choose tools from the stack I most use: Angular, NestJS (with express), MongoDB, a VPS with Ubuntu, and Nginx.

# Node.js Security

In the digital age, software development places high importance on security. JavaScript remains a top choice for web development, and its server-side counterpart, Node.js, has gained immense popularity due to its flexibility and powerful features. However, when working with Node.js, it is crucial to prioritize security. This post guides the best security practices for Node.js in 2023.

Always Keep Your Environment Up-to-Date

The first line of defense in Node.js security is keeping your environment updated. Node.js and its packages are regularly updated to patch vulnerabilities, optimize performance, and add new features. Utilize npm's npm outdated command to identify outdated packages and npm update to upgrade them.

Limit the Use of Third-Party Packages

Node.js's vast ecosystem is one of its significant strengths, but it can also be a weakness if not properly managed. Third-party packages can introduce vulnerabilities into your application. Use them sparingly and only from trustworthy sources. Tools like the Node Security Platform or npm audit can help identify packages with known vulnerabilities.

\*npm audit\*

Use Appropriate HTTP Headers

Adding HTTP headers like Content Security Policy (CSP), HTTP Strict Transport Security (HSTS), and X-XSS-Protection can make your application more secure. To simplify things, you can use libraries like Helmet.js to help set these headers and protect your application from common web vulnerabilities.

Secure Your MongoDB Database

If you use MongoDB as your database, safeguard it against NoSQL injection attacks by verifying and cleansing user inputs. Keep MongoDB updated to the latest version, enable encryption while storing data, and employ MongoDB's built-in RBAC (Role-Based Access Control) for precise access control.

Secure Your Express Apps

When using the popular Node.js framework Express.js, it's essential to utilize its built-in security measures. It's crucial to avoid exposing errors to users, as this could potentially reveal sensitive information or app architecture details to attackers. To prevent brute-force attacks, implement the express-rate-limit middleware, and for protection against Cross-Site Request Forgery (CSRF), use the csurf middleware.

CSRF protection:

Cross-Site Request Forgery (CSRF) is a security vulnerability in web applications. It tricks the victim into submitting a malicious request. It uses the identity and privileges of the victim to perform an undesired function on their be

Let's consider a hypothetical situation. Imagine that you are currently signed into your bank's website, and a fraudulent website attempts to transfer funds from your account to the attacker's. If the bank's website lacks CSRF safeguards, it may accept the request as genuine since it originates from a logged-in user (you) and proceeds with the transaction.

Implement Error Handling

Correct error handling helps with debugging and plays a crucial role in security. Unhandled errors can crash your application, making it a target for DoS attacks. Use 'try-catch' blocks and domain-specific error handlers to catch unhandled errors.

Use TLS (Transport Layer Security)

Always use HTTPS instead of HTTP to encrypt data in transit between the client and server. This can be achieved by implementing TLS. Certificates can be acquired for free from Let's Encrypt.

Use JWT for User Authentication

JSON Web Tokens (JWT) is a secure and stateless way to manage user authentication and can also be used for secure information exchange. However, remember to store tokens securely and set an expiration time.

Secure Against XSS and CSRF

Cross-site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are common attack vectors in web applications. Use output encoding libraries like OWASP's ESAPI to defend against XSS and anti-CSRF tokens for CSRF protection.

# Real-Time Exchange Information With Microservices and NodeJs

Today we will talk about real-time information used in all areas of social life. Speed ​​is the most important thing in the new technology century. In this article, we will try to informed customers who buy the stock papers when the price changed. It should be as fast and accurate as possible.

You can listen to the radio to get information about the exchange news once in a day like above the yellow taxi radio. But if you want to learn something in a real-time, you have to do more then this. You have to use technology. We will use "Socket.IO" in this example.

But speed is not enough. Performance should also be. What is the performance of programming? Completing the job within a required time period. The user experiences are no interrupted or slowdown. How is this accomplished?

How could we improve performance? If we separate some long processes, we keep customers waiting less while working. We will use Microservices for performance. Informing customers about the exchange of stock data is a very intense process. We will add to a Queue all inserted and updated stock data. We will use "RabbitMQ" for this application. We will write consumer microservice and get data from the queue and finally notify all customers with Socket.IO.

With this article, we learned how to handle loads by dividing them into small pieces. We call them Microservices and they work asynchronously. But Microservices is not a silver bullet. There a lot of disadvantages besides the advantages of Microservices. We saw the combined use of RabbitMQ, SocketIO, NodeJs, Consumer, and Angular technologies. You have to automate your test, deployment, management, replica, and more. So if you want to use Microservices, think twice and do once.

# Kafka Streams — How to build an advanced stateful data stream processing

Given a data stream of transactions in real-time, how do you calculate the latest account balance?

The traditional way could materialize the data stream into a data table and run a calculation logic by a job which polls for new transaction records. The output of this solution depends on how often the job polls on the database. It can at most achieve just a "near real-time" update.

Real time data processing is not possible without data stream programming.

Fast and processing data in real-time is the unparalleled advantage of data streaming. It is critical to many modern systems which are handling huge amounts of data. Delay of just several minutes is not tolerated. Hence, real data stream processing has gradually become the backbone of many enterprise systems.

System logic in the data stream is driven by new transactions and the account balance is calculated and sent to the output data stream in a real time.

A typical and simple data stream handles messages one by one in sequence. It could be data filtering, transformation and some other data operations. Data processing are generally stateless.

For example, the diagram below illustrates data flow of input with "A" and "B". The filter discards "B" and then "A" is transformed to "S". The filter and transform logic are stateless. In other words, the logic is triggered by the arrival of a message and processes the data without any knowledge of the previous messages.

Stateless data processing is not sufficient enough for account balance calculation. The latest account balance is calculated based on the previous balance and the new transaction. For example, the balance calculation for a $10 deposit transaction is the previous balance plus $10. We need a place to keep the previous balance in the data flow in order to implement the balance calculation logic.

Stateful data processing is an advanced topic. Data store is introduced to the data stream processor.

# Event Streaming v.s Event Sourcing

Event streaming and event sourcing are two related but distinct concepts in the world of event-driven architecture.

Event streaming is the process of continuously capturing and storing events as they occur in a system. These events can then be processed and analyzed in real-time or stored for later analysis. Event streaming is often used in systems that require real-time processing of large volumes of data, such as financial trading systems or social media platforms.

Event sourcing, on the other hand, is a pattern for building systems that store all changes to the state of an application as a sequence of events. These events can then be used to reconstruct the state of the application at any point in time. Event sourcing is often used in systems that require auditability, traceability, or compliance, such as financial systems or healthcare systems.

Event sourcing is a method for logging every alteration made to an aggregate by appending it to a continuous stream. To reconstruct the ultimate state of an aggregate, it's necessary to read these events in sequence and then apply them to the aggregate. This stands in contrast to the immediate modifications performed in a create, read, update, and delete (CRUD) system. In a CRUD system, any changes to a record's state are stored in a database, effectively overwriting the previous version of the same aggregate.

For effective event sourcing implementations, it's advisable to utilize event stores that offer robust consistency assurances and employ optimistic concurrency control. In practice, this means that when multiple alterations occur concurrently, only the initial modification can append events to the stream. Subsequent modifications may need to be retried or may fail outright.

# Implementing Role-Based Access Control (RBAC) in Node.js

Role-Based Access Control (RBAC) is a crucial aspect of application security. It provides a structured approach to manage and restrict access to resources based on the roles of users within an organization or application. In this comprehensive guide, we will explore the concept of RBAC, discuss its benefits, and walk you through the process of implementing RBAC in a Node.js application. By the end of this article, you will have a clear understanding of how to design and implement RBAC for your own projects.

1. Introduction to Role-Based Access Control (RBAC)

What is RBAC?

Role-Based Access Control (RBAC) is a security concept that restricts system access to authorized users. In RBAC, access rights are associated with roles, and users are assigned one or more roles. These roles define what actions or operations a user can perform within the system.

RBAC simplifies access control by centralizing permissions, allowing administrators to manage user access at a high level. It enhances security by ensuring that users only have the permissions they need to perform their roles, reducing the risk of unauthorized actions.

Benefits of RBAC

Implementing RBAC offers numerous benefits, including:

Security: RBAC minimizes the risk of unauthorized access or actions within an application, reducing security vulnerabilities.

Simplicity: It simplifies user access management by grouping permissions into roles, making administration more straightforward.

Scalability: RBAC is highly scalable, making it suitable for both small and large applications.

Compliance: Many regulatory frameworks, such as GDPR and HIPAA, require robust access control mechanisms like RBAC.

Auditability: RBAC allows you to track and audit user actions, which can be essential for identifying security breaches.

Now that we understand the concept and advantages of RBAC, let's proceed to implement it in a Node.js application.

We've explored the concept of Role-Based Access Control (RBAC) and demonstrated how to implement it in a Node.js application. We've covered role and permission definitions, user authentication, role assignment, role-based middleware, and created a sample Task Management System to showcase RBAC in action.

By implementing RBAC, you can significantly enhance the security of your Node.js applications, control user access effectively, and reduce the risk of security breaches. Additionally, following best practices and considering security considerations ensures the robustness of your RBAC system.

# The Best Strategies to Slim Docker Images: How to Reduce Docker Image Size

You need Docker images to build and ship Docker containers. An image is the base of your container. Thus, keeping images slim and light speeds up the build and deployment of containers. Optimizing Docker images should be practiced in your containerization workflow. Your Docker image size matters for reasons such as:

-Faster deployments time to download, transfer and load into the container runtime, and improved team productivity and application performance.

-Better storage utilization on your machine.

-Reduced network bandwidth when transferring between hosts and container orchestration environments.

-Reducing image size and removing unnecessary files eliminate vulnerable components that expose images to security issues.

-Build and portability efficiency speeds up the build processes and improves resource usage.

Best strategies to slim Docker images

Multistage Builds

Multistage builds allow you to slim Docker images. This allows you to define multiple stages in your Dockerfile to represent the different stages of the build process, using multiple FROM commands. Each represents a different stage of the Docker image build process. In this case, every stage you create has a different base image that executes different commands to build and package your application.

Choosing base images

A Docker base image creates the foundation for structuring your Docker images. They provide pre-built images with the tools and libraries required to run your applications in containers. Docker provides different variants of such base images optimized for specific use cases.

Docker image layers

A Docker image is sectioned into layers. Layers are created based on how you write your Dockerfile: every Dockerfile command or line creates an image layer of its own. These layers are the Docker image file system.

Add .dockerignore

Not every file and folder needs to be copied to the Docker container. Using commands such as COPY . . instructs Docker to copy all files and folders from your local directory to the Docker container.

Leverage compression tools

Combining all the above practices, is guaranteed to help reduce Docker image size, but we can do even more.

Slimming and reducing the size of your Docker images improves your containers' overall builds and deployments. Using the practices and strategies discussed in this post, you can create slim and trim Docker images that will provide faster build times and lower storage and transfer costs. You can use all the above strategies in every image to achieve the optimal size.

# JSON is incredibly slow: Here’s What’s Faster!

Yes, you heard that right! JSON, the ubiquitous format for data interchange in web development, might be slowing down your applications. In a world where speed and responsiveness are paramount, it's crucial to examine the performance implications of JSON, a technology we often take for granted. In this blog, we'll dive deep into the reasons why JSON can be a bottleneck in your applications and explore faster alternatives and optimization techniques to keep your apps running at their best.

What JSON Is and Why You Should Care?

JSON, short for JavaScript Object Notation, is a lightweight data interchange format that has become the go-to choice for transmitting and storing data in web applications. Its simplicity and human-readable format make it easy for both humans and machines to work with. But why should you care about JSON in the context of your web development projects?

JSON is the glue that holds together the data in your applications. It's the language in which data is communicated between servers and clients, and it's the format in which data is stored in databases and configuration files. In essence, JSON plays a pivotal role in modern web development.

The popularity of JSON and why people use it…

JSON's popularity in the world of web development can't be overstated. It has emerged as the de facto standard for data interchange for several compelling reasons:

Given these advantages, it's no wonder that developers across the globe rely on JSON for their data interchange needs. However, as we explore deeper into the blog, we'll uncover the potential performance challenges associated with JSON and how to address them effectively.

The Importance of Application Speed and Responsiveness

In today's fast-paced digital landscape, application speed and responsiveness are non-negotiable. Users expect instant access to information, swift interactions, and seamless experiences across web and mobile applications. This demand for speed is driven by several factors:

In the world of development, JSON stands as a versatile and indispensable tool for data interchange. Its human-readable format and cross-language compatibility have made it a cornerstone of modern applications. However, as we've explored in this guide, JSON's widespread adoption doesn't exempt it from performance challenges.

# Building a Load Balancer using Node JS + Express

In this article, we will see what is a load balancer, why we should use it, and when we should use a load balancer.

First of all, let's see when one should use a load balancer

you don't need a load balancer if your application or website doesn't get a lot of traffic or requests, but if it does and become more popular, your underlying server might not be able to manage the increased demand, because a single Node JS server is not flexible enough to handle very large amount of traffic.

This problem can be fixed by adding more machines. but a load balancer is required to split traffic among all of your application servers.

What is a Load Balancer?

A load balancer is a software component that sits between the client and server and distributes incoming requests across multiple servers. It monitors the health of each server and routes the incoming requests to the most available and healthy server.

Load balancers can be implemented in different ways, such as hardware appliances, software appliances, or cloud-based services. In this post, we will focus on how to implement a software load balancer in Node.js and build a balancer using Node js.