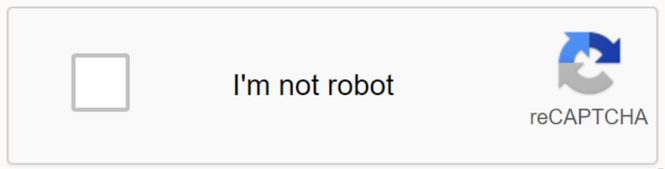

Continue

Software architecture pdf

FormerLy Microsoft, formerly Facebook. Co-founder of Educative.ioSo embarked on the entrepreneurial quest to build your own web application. You have the idea in place, but getting the architecture right is very important. In this post we'll go through these key areas: What is software architecture That is important Software architecture The difference between software architecture and software designDepress software architectureHow to decide the number of levels your app needs to be from a hurricane or vertical scale... What's right for my app? Monolith or microservice? When should I use NoSQL or SQL? Choosing the right technology for your workHow to become a software architect Where to go from here The purpose of this post is to give you a solid understanding of web architecture, the concepts involved, and how to choose the right architecture and technology when designing your app. So by the end of this post, when you're going to design an app from the exposed bones you won't be sitting in the dark anymore. If you are looking for a full course on web application and software architecture, I recommend taking out web application and software architecture 101. This is a useful course for anyone looking to strengthen their overall knowledge of software architecture. Let's dive in! What is software architecture? System software architecture describes its main components, their relationships, and how they interact with each other. It's basically used as a blueprint. It provides abstraction to manage system complexity and establish communication and coordination between components. Here are some key points: Architecture helps define a solution to meet all technical and operational requirements, with the common goal of optimizing performance and security. The design of the architecture involves the intersection of the needs of the organization, as well as the needs of the development team. Each decision can have a considerable impact on quality, retention, performance, etc. One of my favorite definitions of software architecture came from Ralph Johnson, co-author of Design Patterns: Elements of Object-Oriented Software for Repetitive Industrialization. He stated: These are the decisions you would like to make early in the project. So however he said, let's move on to why software architecture is important. Why is software architecture important? The key element in creating everything successfully is getting the base right. Now whether it's building a building or making pizza. If we don't get the base right, we need to start over; There's no other way. Building a web application is no different. The architecture is its foundation and should be thought carefully to avoid major design changes - code satisfaction at a later point in time. Many engineers will tell you you don't want to delve into the redesign It eats up your time like a black hole. It has the potential to push your shipping date further down the calendar in months, if not more. And that doesn't even raise the waste of engineering and financial resources caused by it. It also depends on what stage of the development process we encountered at an impasse due to the hasty sea decisions during the initial design stages. So, before we even touch the code and get our hands dirty, we need to do the basic architecture right. Although software development is an iterative and evolutionary process, we don't always get the perfect things at first to go. Still, that can't be an excuse for not doing our homework. The difference between software architecture and software design there is often confusion between software design and architecture so we will dismantle it. The software architecture is used to define the skeleton and components at a high level of system, and how they will all work together. For example, would you need a serverless architecture that divides the application into two components: BaaS (back end as a service) and FaaS (functions as a service)? Or do you need something like a microservics architecture where the different features/tasks are divided into separate modules/code bases respectively? Choosing architecture will determine how you deal with performance, fault tolerance, scalability, and reliability. The software design is responsible for designing the level of code such as, what each module does, the scope of the class, and the purposes of the functions, etc. When used strategically, they can make a programmer significantly more efficient by allowing them to avoid reinventing the wheel, instead using methods already distilled by others. They also provide a useful common language to perception of problems and repeated solutions when discussing with others or managing code in larger groups. Here is a good article on understanding the importance of software design and the tried and correct patterns that developers often use: 7 most important software design patterns. Software architecture templatesClient-server servers The architecture runs on a request-response model. The client sends the request to the server for information - the server responds with it. Every website you browse, whether it's a WordPress blog or a web app like Facebook, Twitter or your banking app is built on the client server architecture. A Peer-to-peerA P2P network is a network where computers also known as ZIM can communicate with each other without the need for a central server. The absence of a central server negates the possibility of a single failure point. All computers on the network have equal rights. A node acts as a extract and a knock at the same time. Therefore, even if some computers/adids are downloading, the network and communications are still running. P2P is the basis of blockchain (MVC) What are you doing? The MVC architecture is an architectural template of software in which application logic is divided into three components based on functionality. These components are called: models - represent how data is stored in database views - the components that are visible to the user, such as output or GUI controllers - the components that serve as an interface between models and views MVC architecture is used not only for desktop applications but also for mobile and Web applications. Microservices In microservices architecture, different features/tasks are divided into separate modules/code bases respectively, which work in conjunction with each other and create a large service as a whole. This particular architecture allows for easier, cleaner maintenance of applications, development, testing and deployment compared to monolithic architecture. Event-driven architecture That is uneventfully stored is also known as reactive or event-driven architecture. Event-driven architectures are quite popular in developing modern web apps. They can handle a large number of simultaneous connections with minimal resource consumption. Modern applications need a completely asynchronous model to expand. These modern Web frames provide more reliable behavior in a distributed environment. LayersThen this template can be used to structure programs that can be decrypted into groups of sub-tasks, each of which is at a certain level of abstraction. Each layer provides services for the next high layer. Here are the most common layers:Presentation layerA business logic access layerThe hex data access layerThe architecture consists of three components:The focus of this architecture is to make different components of the application independent, loosely and shared and easy to test. The architectural pattern holds the field at its core, that's business logic. From the outside, the outer layer has ports and adapters. Ports work like an API, as an interface. All input to the app goes through the interface. How to decide the number of levels your app should be a singlePros layer application:No network latches Data quickly and easily available Data is not transmitted over a network that ensures data safetyCons:Little control over the application; It is difficult to implement new features or code changes after it is sentCheck should be very thorough with minimal room for errors that different level applications are exposed to be navigated or reverse engineered Two-tier applicationPros:Fewer network calls since the code and user interface are on the same MachineDatabase server and business logic is physically close, which offers higher performance. Cons: Because the client has most of the application logic, problems are experiencing problems controlling the software version and redistributing new versions. Scalable as it only supports a limited number of users. When a customer is multiple Increased, application performance can slow down due to the fact that clients require separate connections and CPU memory to continue. Because the application's logon is combined with the client, it is difficult to re-use the logon. Three-tier ApplicationPros:Data damage through client applications can be undone as the data transferred at the middle level for database updates ensures that the validity of business logic on a central server makes the data more secureDue to distributed deployment of application servers, system scalability is improved because a separate connection from each client is not required and which connections from application servers are few enough. Cons:More effort should usually be enforced when creating three-tier applications as communication points increase (client from middle tier to server, instead of client directly to server) and increased performance by tools such as Visual Basic, PowerBuilder, Delphi will be reduced. N-Tier applicationPros:All advantages of three-tier architectureSize performance due to loading from database level and client level, allowing it to fit medium to high industriesCons:Due to components of the levels, The complex structure is difficult to implement or maintain conclusion on you to choose a level one architecture when you don't want any network latch Select a two-tier application when you need to minimize network latch and you need more control of data within your application you need to choose a three-tier architecture when you need control over your application's code/business logic & want it to be secure, and you need control over data in your application. You must select an N-layer architecture when you need an application to scale and manipulate large amounts of data. Scaling Horizontally or Vertically... What's right for my app? If your app is a utility or tool that's expected to receive minimal consistent traffic, it might not be mission critical. For example, an internal organization tool or something similar. Why bother hosting him in a decentralized environment? A single server is enough to manage traffic so you can go with vertical scaling when you know that traffic congestion will not increase significantly. If your app is a public-facing social app like a social network, fitness app or something similar, traffic is likely to surge exponentially in the near future. In this case, high availability and horizontal scalability are important to you. Build to deploy it to the cloud and there's always horizontal scalability in mind right from the start. Here's a good website for learning more about scalaness. Monolith or microservice? Let's investigate when you have to choose one over the other. When to use monolithic architecture Monolithic applications are best suited for use cases where the requirements are Simply, the app is expected to handle a limited amount of traffic. One example of this is implementing an internal tax calculation of a similar open public organization or tool. These are the use cases where the business is confident that there will be no exponential growth in user base and traffic over time. There are also cases where development teams decide to start with monolithic architecture and later expand into distributed microservics architecture. This helps them deal with the complexity of the application step by step as needed. That's exactly what LinkedIn did. When to use microservics architecture The microservics architecture is best suited for complex use cases and apps that expect traffic to grow exponentially in the future like a luxurious social networking application. A typical social networking app has various components such as messaging, real-time chat, live video streaming, image uploads, Like, Share feature, etc. In this scenario, I propose to develop each component individually while maintaining one responsibility and the principle of separating concerns. Writing each feature at the base of one code won't take time to turn it into a mess. So, until now, in the context of monolithic and microservics, we've gone through three approaches: choosing a monolithic architecture choosing microservics architecture start with monolithic architecture and then scale to microservic architecture. Choosing a monolithic or microservics architecture depends heavily on our use case. I suggest, keep things simple, have a thorough understanding of the requirements. Get the layer of soil, build something only when you need it - continue to evolve the code iteratively. It's the right way. When should I use NoSQL or SQL? When should I select an SQL database? If you write stock trading, banking or a financing-based app or need to store a lot of relationships, for example, when writing a social networking app like Facebook, then you need to choose a relative database. Here's why:Transactions & data consistency if you write software which has something to do with money or numbers, that makes deals, acid, consistency and super data important to you. Relative DBs shine when it comes to transactions & data consistency. They've been complying with the acid rule, been around for years and tested in battle. Store relationships if your data has a lot of relationships like which of your friends live in a particular city? Which one of your girlfriends already ate at the restaurant you're planning to visit today? Etc. There is nothing better than a relational database to store this type of data. Relative databases are built to store relationships. They have been tried and tested - used by big guns in the industry like Facebook as the main user-facing database. Popular Relative Databases:MySQLMicrosoft SQL To select a NoSQL database, here are several reasons why you want to select a NoSQL database:Handling a large number of write-to-read operationsIn the direction of NoSQL databases when you need to expand quickly. For example, when there are a large number of read-write operations on your Web site and while dealing with a large amount of data, NoSQL databases best match these scenarios. Because they have the ability to add in-flight hinges, they can handle more movement at the same time and large amounts of data with the minimum delay. Running Data Analytics DatabasesNoSQL is also suitable for data analysis use cases where we have to deal with the flow of massive amounts of data. Popular databases NoSQL:MongoDBRedisCassandraHBASEPicpicing the right technology for working in real-time interaction if you are building an app that needs to:interact with the contrast server in real time, such as a messaging application, or audio video streaming app like Spotify, Netflix etc. A constant connection between the client and the server and technology that does not block at the back end. Then some of the popular technologies that allow you to write these apps are NodeJS, and the popular python frame known as Tornado. If you work in the Java ecosystem you can look into spring reactor, play, Akka.io.Peer-to-peer web application if you intend to build a peer-to-peer web application, for example, a P2P distributed search engine or p2P Live TV radio service, something similar to LiveStation by Microsoft, so you want to look into JavaScript, protocols like DAT, IPFS. FreedomJS Coupe, this is a framework for building P2P web applications that run in modern web browsers. Regular CRUD-based applications If you have simple use cases such as a standard CRUD-based app, then some of the technologies you can use are: Spring MVC, Django Python, Ruby on Rails, PHP Laravel, ASP .NET MVC. Simple, small-scale apps If you plan to write an app that doesn't involve as much complexity as a blog, a simple online form, simple apps that integrate with social media that are infused within the portal's IFrame, then you can choose PHP. You can also consider other web frames like Spring boot, Ruby on Rails, which tweezers the text, configuration, and development time by slots - facilitating rapid development. But PHP hosting will cost a lot less compared to hosting other technologies. It is ideal for very simple use cases. Processor-intensive and memory-intensive applications Do you need to run intensive, memory-intensive computational tasks, heavy on the end, such as big data processing, parallel processing, running monitoring, and analysis on a fairly large amount of data? Standard Web frames and script languages are not designed to overestimate numbers. Technology commonly used in the industry to write performance systems, scalable and distributed is C++. It has features that facilitate low-level memory manipulation, More control over developer memory when writing distributed systems. Most cryptocurrencies are written in this language. Rust is a programming language similar to C++. It's built for high performance and safe performance bouquets. It's been gaining a lot of popularity lately among the developer circles. Java, Scala and Arlang are also a good choice. Most large-scale enterprise systems are written in Java.Go and is a programming language of Google to write apps for multi-core machines and handling a large amount of data. Julia is a dynamically programmed language built for high performance and running calculations and numerical analysis. How to become a software architect? If this all sounds interesting, then you can aspire to be a software architect. But where do you start? Well, it's very rare for someone to start as a software architect, so most software engineers work for a few years before they take on architecture design. One of the best ways to get to know software architecture is by designing your own web applications. This will force you to think through all the different aspects of your application, from load balancing, message queue, stream processing, grocery storage and more. Once you start figuring out how these concepts fit into your app, you'll be on your way to becoming a software architect. As an ambitious software architect, you need to constantly expand your knowledge and stay on top of the latest industry trends. You can start by learning one or more programming languages, working as a software developer, and gradually in the character. Although you may not get a degree in software architect in college, there are other courses you may find useful. Web Application and Software Architecture 101 is a great place to start learning the best practices for designing and implementing web applications. Where do I go from here? While there was a lot covered in this post, we just touched on the surface on this issue. We haven't toured the REST API yet, high availability and CAP tournament. If your designation dives deep into the software architecture, I highly recommend web application and software architecture 101. It goes you step by step through various components & mixed concepts when designing the architecture of a web application. You will learn about different architectural styles such as client server, peer-to-peer distributed architecture, microservices, the basics of data flow in a web application, different mixed layers, concepts like scalability, high availability and much more. In addition, you will pass the techniques of choosing the right architecture and stack technology to implement your use case. I will walk you through various use cases which will help you gain insight into what technology & architecture best fits for a particular use case when writing a web You'll understand the technological transactions involved. If you're just starting your career in software development, this course will help you a lot. It will also help you with software engineering interviews, especially for full stack key jobs. Happy learning! Previously posted level your reading game by joining a hacker at noon now! Now!