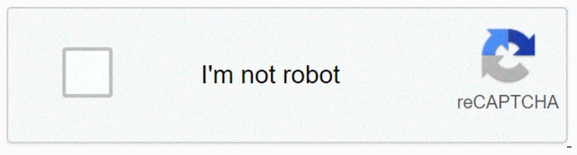

Continue

Naval architecture software free

By ExtremeTech Staff on February 5, 2002 at 9:53 am This site can earn affiliate commissions from links on this page. Terms of use. Itanium programmers, this is the guide for you. This white paper defines the common software conventions for the processor and tells you how to compile, link, and execute applications on Itanium-based operating systems. Fleets around the world engage in surface and sub-surface military operations at sea. Learn more about how aircraft carriers, aircraft carrier battle groups, and next-generation destroyers work. Ad ad ad ad ad Ex-Microsoft, Ex-Facebook. Co-founder at Educative.ioSo you have initiated the entrepreneurial journey to build your own web application. You've got the idea in place, but the importance of getting the architecture right is extremely important. In this post we will go through these key areas: What is software architectureWhere is software architecture importantDiffer between software architecture and software designSoftware architecture patternsHow to decide how many levels your app should haveHorizontal or vertical scaling ... What's right for my app? Monolith or Microservice? When do you need NoSQL or SQL? Choose the right technology for the jobHow to become a software architectWhere to go from here The goal 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 at the end of this post, when you go to design an application from the bare bones, you won't sit in the dark anymore. If you are looking for a complete course on web application and software architecture, I would recommend checking out Web Application and Software Architecture 101. This is a useful course for anyone who wants to strengthen their overall knowledge of software architecture. Let's dive in! What is software architecture? Software architecture in a system describes its key components, their relationships, and how they interact with each other. The essentials serve as a plan. It provides an abstraction to control the complexity of the system and establish communication and coordination between components. Here are some key points: The architecture helps define a solution that meets all the technical and operational requirements, with the common goal of optimizing for performance and safety. Designing the architecture involves the intersection of the organization's needs and the needs of the development team. Each decision can have a significant impact on quality, maintenance, performance, etc. One of my favorite definitions of software architecture came from Ralph Johnson, co-author of Design Patterns: Elements of Reusable Object Oriented Software. He stated that: These are the decisions you wish you could make quite early in a project. So with that said, let's move on to why software architecture is important. Why is Important? The key element in the successful creation of something to get the base to the right. Now whether it's building a building or making a pizza. If we don't get the base right, we have to start over; there's no other way around. Building a web application is no different. The architecture is its base and must be carefully thought out to avoid major design changes & code refactoring at a later date. Many engineers will tell you that you don't want to delve into re-designing things. It eats your time like a black hole. It has the potential to push your shipping date further down the calendar after months, if not longer. And it's not even bringing up the waste of engineering and financial resources that are caused because of this. It also depends on what stage of the development process we hit a stalemate due to the hasty decisions that were made in the initial design phases. So before we even touch the code and get dirty hands, we need to make the underlying architecture right. Although software development is an iterative and evolutionary process, we don't always get things perfect at the first time. But that can't be an excuse not to do our homework. The difference between software architecture and software designThere is often confusion between software design and architecture, so we will break this down. Software architecture is used to define the skeleton and the high-ranking components of a system and how they will all work together. For example, do you need serverless architecture that divides the application into two components: BaaS (backend-as-a-service) and Functions-as-a-service (FaaS)? Or do you need something like a microservice architecture where the various functions/tasks are divided into separate respective modules/code bases? Choosing an architecture determines how you handle performance, fault tolerance, scalability, and reliability. Software design is responsible for code level design such as what each module does, the classes scope, and the features purpose, etc. When used strategically, they can make a programmer significantly more efficient by allowing them to avoid reinventing the wheel, instead using methods refined by others already. They also provide a useful common language to conceptualize repetitive problems and solutions when discussing with others or managing code in larger teams. Here is a good article on understanding the importance of software design and the tried and true patterns that developers often use: The 7 main software design patterns. Software architecture patternsClient serverArchive works on a query-response model. The client sends the request to the server for information & the server responds with it. Every website you surf, be it a Wordpress blog or a web application like Facebook, Twitter or your banking app is built on client-server architecture. Peer-to-peerA P2P network is a network where computers, also called nodes, can communicate each other without the use of a central server. The absence of central server excludes the possibility of a single error point. All computers in the network have equal rights. A node acts as a soror and a leecher at the same time. So even if some of the computers/nodes go down, the network & communication is still up. P2P is the foundation of blockchain technology. The MVC architecture (Model-View-Controller) MVC architecture is a software architectural pattern in which application logic is divided into three components based on functionality. These components are called: Models - represents how data is stored in the Views database - the components that are visible to the user, such as an output or a GUI Controllers - the components that act as an interface between models and the viewsMVC architecture are used not only for desktop applications, but also for mobile and web applications. MicroservicesIn a microservice architecture, different functions/tasks are divided into separate respective modules/code bases that work with each other and form a large service as a whole. This particular architecture facilitates easier & cleaner app maintenance, feature development, testing & implementation compared to a monolithic architecture. Event drivenNon-blocking architecture is also known as Reactive or Event-driven architecture. Event-driven architectures are pretty popular in modern web application development. They are able to handle a large number of simultaneous connections with minimal resource consumption. Modern applications need a fully asynchronous model to scale. These modern web frameworks provide more reliable behavior in a distributed environment. LayeredThis pattern can be used to structure programs that can be decoction into groups of subtasks, each of which is at a specific abstraction level. Each layer provides services to the next higher layer. Here are the most common layers:Presentation layerApplication layerBusiness logic layerData access layerHexagonalArchitecture consists of three components:The focus of this architecture is to make various components of the application independent, loosely coupled & easy to test. The architectural pattern keeps the domain at the center, it's business logic. On the outside, the outer layer has Ports & Adapters. Ports acts as an API as an interface. All input to the app goes through the interface. How to decide how many levels your app should haveSingle applicationPros:No network latencyData is fast and easily accessibleData is not transmitted over a network that ensures data securityCons:Poor control of the application; difficult to implement new features or code changes when it is shippedTesting must be extremely thorough with minimal space for errorSing level applications are vulnerable to being tweaked or reverse engineeringTwo-tier applicationPros: Fewer network calls since the code and UI are in the same machineDatabase server and business logic is physically close, giving higher the client keeps most of the program logic, problems arise in controlling the software version and re-distributing new versions. Lack scalability because it only supports a limited number of users. As multiple client requests increase, application performance may slow down because clients require separate connections and CPU memory to continue. Because the application logic is connected to the client, it is difficult to reuse logic. Three-layer applicationPros: Data corruption through client applications can be eliminated as data transferred in the middle level of database updations ensures its validityLocation of business logic on a centralized server makes data more secureDue for distributed installation of application servers, scalability of the system is improved since a separate connection from each client is not required, while connections from few application servers are sufficient. Cons: Usually a greater effort should be enforced when creating 3-tier applications as communication points are increased (client to mid level to server, instead of direct client to server) and performance increased by tools like Visual Basic, PowerBuilder, Delphi will be reduced. N-Tier applicationPros: All benefits of three-tier architectureEaeture increased due to off-load from database level and client level to suit medium to large industryCons: Due to the componentization of the levels, the complex structure is difficult to implement or maintainConclusionYou must choose an architecture at a single level, when you don't want a network latencyPrevent a two-tier program to minimize network latency and you need more control over data in your applicationYou need to choose a three-level architecture when you need control over the code/business logic of your application & want it to be secure and you need control over data in your application. You must select an N-level architecture when you need your application to scale and handle large amounts of data. Horizontal or vertical scaling... What's right for my app? If your app is a utility or tool that is expected to receive minimal consistent traffic, it may not be mission critical. For example, an internal tool in an organization or something similar. Why bother hosting it in a distributed environment? A single server is enough to control traffic, so you can go with vertical scaling when you know that the traffic load wouldn't increase significantly. If your app is a public-facing social app like a social network, a fitness app or something similar, then traffic is expected to spike exponentially in the near future. In this case, both high availability and horizontal scalability are important to you. Build to implement it on the cloud & always have horizontal scalability in mind right from the start. Here is a good website to learn more about scalability. Monolith or Microservice? Let's explore, you must select one over Other. When to use monolithic architectureMonolitic applications best suit use cases where the requirements are pretty simple, the app is expected to handle a limited amount of traffic. An example of this is an internal tax calculation app for an organization or a similar open public tool. These are the use cases where the company is confident that there will not be an exponential growth in the user base and traffic over time. There are also cases where the development teams decide to start with a monolithic architecture and later scale out to a distributed microservice architecture. This helps them manage the complexity of the application step by step when and if necessary. This is exactly what LinkedIn did.When using microservice architectureMicroservice architecture best suits complex use cases and for apps that expect traffic to increase exponentially in the future as a fancy social networking application. A typical social networking application has various components such as messaging, real-time chat, LIVE video streaming, image uploads, Like, Share function, etc. In this scenario, I would suggest developing each component that takes into account the principle of shared responsibility and the principle of separation of concerns. Writing each function into a single code base would take some time to become a mess. So now, in the context of monolithic and microservices, we've gone through three approaches: Picking a monolithic architecture Picking a microservice architectureStart with a monolithic architecture and then later scaling out a microservice architecture. Choosing a monolithic or a microservice architecture largely depends on our use case. I would suggest keeping things simple, having a thorough understanding of the requirements. Few lay off the ground, build something when you need it & keep developing the code iteratively. That's the right way to go. When do you need NoSQL or SQL? When should you select an SQL database? If you're writing a stock trading, bank, or a Finance-based app, or you need to save a lot of relationships, for example, when writing a social networking app like Facebook, then you need to choose a relational database. Here's why:Transactions & data consistencyIf you write software that has something to do with money or numbers that make transactions, ACID, data consistency super important to you. Relational DBs shines when it comes to transactions & data consistency. They adhere to the ACID rule, have been around for years & are battle-tested. Saving relationshipsIf your data has a lot of relationships like which of your friends live in a particular city? Which friend already ate at the restaurant you plan to visit today? Etc. There is nothing better than a relational database for storing this kind of data. Relationship databases are built to store relationships. They have been tested & tested & used by big guns in the industry like Facebook as the the database. Popular relational databases:MySQLMicrosoft databases:MySQLMicrosoft ServerPostgreSQLMariaDBWough to select a NoSQL databaseHere are a few reasons why you want to select a NoSQL database: Handling a large number of read write operationsLook against NoSQL databases when you need to scale quickly. For example, when there are a large number of read-write operations on your site, and when dealing with a large amount of data, NoSQL databases fit best in these scenarios. Because they have the ability to add nodes on the fly, they can handle more simultaneous traffic and large amounts of data with minimal latency. Running data analyticsNoSQL databases also best suit data analytics use cases where we have to deal with an influx of massive amounts of data. Popular NoSQL databases: MongoDBRedisCassandraHBASEPicking the right technology for jobsReal time data interactionIf you build an app that needs:To interact with the backend server in real time, such as a messaging application, or an audio-video streaming app like Spotify, Netflix, etc. A persistent connection between the client and the server and a non-blocking technology in the back-end page. So some of the popular technologies that allow you to write these apps are NodeJS, and the popular Python framework known as Tornado. If you work in the Java ecosystem, you can look at Spring Reactor, Play, Akka.io.Peer-to-peer web appIf you intend to build a peer-to-peer web app, for example a P2P distributed search engine or a P2P Live TV radio service, something similar to The LiveStation of Microsoft, then you will want to look at JavaScript, protocols like DAT, IPFS. Checkout FreedomJS, it is a framework for building P2P web apps that work in modern web browsers. CRUD-based regular applicationsIf you have simple use cases such as a regular CRUD-based app, then some of the technologies that you can use are: Spring MVC, Python Django, Ruby on Rails, PHP Laravel, ASP .NET MVC.Simple, small applicationsIf you intend to write an app that doesn't involve much complexity like a blog, a simple online form, simple apps that integrate with social media that runs within the IFrame of the portal, then you can choose PHP. You can also consider other web frames like Spring boot, Ruby on Rails, which cuts down verbosity, configuration, development time by notch & facilitate the rapid development. But PHP hosting will cost much less compared to hosting other technologies. It is ideal for very simple use cases. CPU- and memory-intensive applicationsIt you need to run CPU-intensive, memory-intensive, heavy compute tasks on backend like Big Data Processing, Parallel Processing, Running Monitoring & Analytics on quite a large amount of data? Regular web frameworks & scripting languages are not intended for talk-nude. Tech commonly used in the industry to write performant, scalable, distributed systems are C++. It has features that facilitate low-level memory manipulation, giving more control over to developers when you write distributed systems. Systems. cryptocurrencies are written using this language. Rust is a programming language similar to C++. It is built for high performance and safe concurrency. It's gaining a lot of popularity lately among developer circles. Java, Scala & Erlang is also a good choice. Most of the major enterprise systems are written on Java.Go is a programming language of Google to write apps for multi-core machines & handling a large amount of data. Julia is a dynamically programmed language built for high-performance & running calculations & numeric analysis. How to become a software architect? If all this sounds interesting, then you can strive to be a software architect. But where do you start? Well, it's extremely uncommon for someone to start out as a software architect, so most software engineers work for a few years before taking on designing architecture. One of the best ways to familiarize yourself with software architecture is to design your own web applications. This will force you to review all the different aspects of your application, from load balancing, message queue, power processing, caching, and more. Once you start to understand how these concepts fit into your app, you'll be well on your way to becoming a software architect. As an aspiring software architect, you must constantly expand your knowledge and keep track of the latest industry trends. You can start by learning one or more programming languages, work as a software developer and gradually make your way. Although you can't get a software architect's degree in college, there are other courses that you may find useful. Web Application and Software Architecture 101 is a great place to start learning best practices for designing and implementing web applications. Where are we going from here? While there was a lot covered in this post, we simply touched the surface on this topic. We still have yet to explore REST APIs, high availability, and CAP phrase. If you would like a deep dive into software architecture, I highly recommend Web Application and Software Architecture 101. It takes you step by step through various components & concepts involved when designing the architecture of a web application. You will learn about different architectural styles such as client server, peer-to-peer decentralized architecture, microservices, the basics of data flow in a web application, different layers involved, concepts like scalability, high availability and more. In addition, you will go through the techniques of choosing the right architecture and technology stack to implement your use case. I will walk you through various use cases that will help you get an insight into what technology & architecture best suits a particular use case when writing a web application. You will get to understand the technology trade-offs involved. If you are a beginner just beginning your career in software development, this help you a lot. It will also help you technical interviews, especially for the full stack of developer positions. Happy learning! Previously published on Hacker Noon Create your free account to unlock your custom reading experience. Experience.