Engineering and Technology Journal e-ISSN: 2456-3358

Volume 09 Issue 06 June-2024, Page No.- 4276-4280

DOI: 10.47191/etj/v9i06.14, I.F. – 8.227

Abstracting the Architecture Design System to Create New Applications

Arjun Sudhanva Naik

Tech Lead, Commercial Advantage First Citizens Bank Virginia, USA

ABSTRACT: In software engineering, application design and architecture play an important function of their improvement, maintenance and scalability. This paper explores the idea of abstracting architectural design approaches to facilitate the advent of latest packages. By decoupling structure from unique application contexts, builders have the ability to test new technologies, adapt to evolving wishes, and scale their programs. Finally, this paper recommends to adopt abstract architecture design systems as a method for software improvement. By offering a based framework for constructing new packages, the abstraction of architectural design structures empowers developers to innovate, iterate and optimize in a dynamic business surroundings, and truly fosters a tradition of continuous improvement and excellence in the application development exercise.

KEYWORDS: architecture design, end to end architecture, development practice, evolving needs, scaling, abstract architecture design systems, software engineering, scalability, new technology, architecture optimization

I. INTRODUCTION

In the ever-evolving landscape of software engineering, the design and structure of applications stand as foundational pillars that underpin their fulfillment, durability, and adaptability. As technology continues to improve and consumer expectations evolve, developers face the perennial mission of making programs that are not only functional but also efficient, flexible and scalable to fulfill changing demands. In reaction to this venture, the concept of abstracting structure design structures has emerged as a effective paradigm for steering the improvement of latest applications.

This advent sets the level for a comprehensive exploration of abstracting structure design systems and its implications for modern-day software program development. We embark on an adventure to elucidate the standards, methodologies, and advantages of abstracting architecture design structures, dropping light on how this approach can empower builders to create modern, resilient, and maintainable applications.

The creation of abstracting structure layout structures represents a paradigm shift in software engineering, emphasizing the significance of modularity, reusability, and abstraction within the layout system. By encapsulating not unusual architectural styles, design concepts, and satisfactory practices into reusable frameworks, builders can streamline the improvement method, lessen redundancy, and foster a subculture of code reusability.

Moreover, abstracting structure design structures offer a pathway to agility and innovation in application development. By decoupling architecture layout from precise implementation information, builders advantage the flexibility to test with new technology, adapt to evolving necessities, and iterate on their designs with extra ease.

In this paper, we delve into the multifaceted dimensions of abstracting structure layout systems, exploring its theoretical underpinnings, practical programs, and real-global

implications. Abstracting the design procedure will help offer a comprehensive expertise of this paradigm and its ability to revolutionize software program development practices.

II. THE BLACK BOX SOLUTION

Black Box is a term that is used to point towards a conceptual model that will abstract the underlying components. Without entering into the details, we can analyze the model to derive a simple solution for visual understanding. The actual term ‘black box’ originates from the idea of a container where the user can only observe the inputs and outputs of the system without being able to see or understand what lies underneath. From the user's perspective, the system behaves like a ‘black box’ whose internal workings are opaque or invisible.

In software development, black box approaches are often used to encapsulate complexity, improve modularity, and promote abstraction. By hiding implementation details, developers can create components or systems that are easier to understand, maintain, and integrate into larger software architectures. Additionally, black box approaches can enhance security by limiting access to sensitive information or functionality.

Using this concept of black box, we have created an abstract model that be followed to easily solve complex questions. The dotted boxes indicate optional elements that can be added or removed as per use case. The main elements are indicated by

solid lines and need to be considered for solutions from start as they are key pieces on the board. The optional boxes useful in some cases and can be omitted in some cases. A good architect will determine when to use them and when to skip them.

The diagram below should be a good starting point for all architects envisioning a solution for a complex application build. Of course, there are more elements that can be added and this diagram is not your one stop solution for all problems but it gives you the starting point from which you can begin the process of creating a diagram to present an overarching structure.

This method will help developers ace any interview design problems put forth as well as help them understand a general solutioning method. Using this a developer can orchestrate an explanation to further invoke a series of thoughts to generate an answer to the problem put forth.

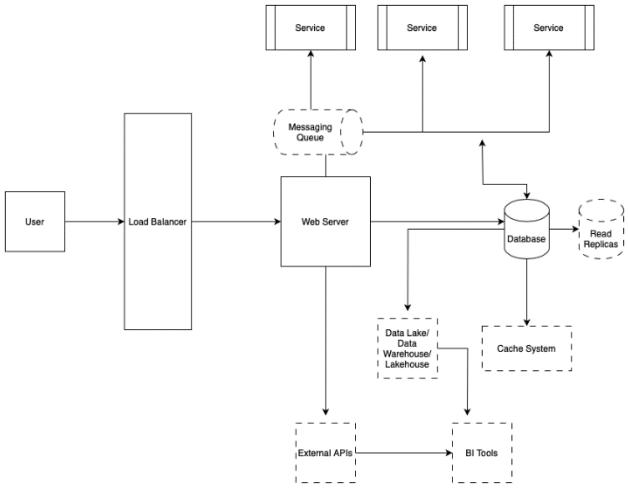

Figure 1: A black box representation of a basic architecture diagram

III. INDIVIDUAL COMPONENTS

Here we begin to deep dive into individual components shown in Figure 1.

A. User

A user or end user is the recipient of the application. This can be an individual or an institution. The box denotes the end client. This can be on any devices- mobile, computer, tablet or even embedded point of sale device. This can vary as per the requirement and the developer has to understand exactly what the user base needs and use this box accordingly. Sometimes multiple types of devices can be used by the end user to access the application and this must be evaluated during the design phase.

B. Load Balancer

A load balancer is a crucial element in distributed computing structures, performing as the role of a site visitors manager that efficiently distributes incoming network traffic throughout multiple servers or assets. Its number one feature is to optimize aid usage, improve responsiveness, and make sure high availability and reliability of packages and offerings. By distributing incoming requests equally, load balancers prevent any single server from becoming overwhelmed, thereby enhancing device performance and scalability. Load balancers can utilize numerous algorithms to determine how to distribute traffic, inclusive of round-robin, least connections, or weighted distribution based on server capacity. Additionally, load balancers often incorporate health tests to display the utilization of servers and dynamically alter traffic routing to avoid sending requests to bad or overloaded servers. In essence, load balancers play an important function in optimizing the performance, reliability, and scalability of current disbursed systems.

As a load balancer stands between incoming traffic redistributing the load, it can be divided into various types depending upon the requirements. Use cases will determine what time of a load balancer is needed- hardware or software load balancers, layer 4 (transport layer) or layer 7 (application layer) or DNS load balancer and so on. This also varies if you have an on-premise solution or a cloud solution, creating various options. With a black box representation, we can state its presence adding to the dynamic elements of the ever-growing architecture designs.

C. Web Server

A web server is a software application or a hardware device that serves content to users over the World Wide Web or as we fondly know it , the Internet. It receives requests from clients/user(as indicated in the black box) and delivers web pages, files, or related content in response. Web servers may be deployed on various operating systems, consisting of Linux, Windows, and macOS. Popular web server software program includes Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS), and LiteSpeed. Web servers make up for the foundation of the internet infrastructure, permitting the shipping of web content and the website hosting of web sites, internet programs, and other online offerings. They play an important function in facilitating conversation between client and servers and making sure the supply and accessibility of internet assets.

As we abstract the Web server as a black box, we can use the various options available to us to ensure that this aspect of the architecture is well thought of. The webserver can be a full-fledged server or can be a replaced by a serverless computing service for smaller applications.

D. Services

As seen in Figure 1, the webserver is connected to various services that help it to transfer data to the client. These services are nothing but internal components built for specific tasks. These are written in isolation to the main backend service to ensure that data can be synchronously or asynchronously worked on without overloading the server with a lot of threads. These services will take in inputs or triggers and render output that will be routed back to the client.

Services are usually defined with a request response system and have underlying authentication to ensure that the access is

limited and authorized. They enable modularity and scalability in the system. Data exchange and performance is optimized with the help of services. These services will vary with the industry but are an important part of the structure and need to be always considered while architecting a solution.

E. Database

Selecting the right database is an essential choice which can extensively impact its performance, scalability, and upkeep. Several factors ought to be considered when making this choice. Firstly, the characteristics of the records itself plays an essential position. A few questions need to be answered - Is the data structured or unstructured? Is it relational or non-relational? Understanding the data model requirements is important in figuring out whether a SQL (relational) or NoSQL (non-relational) database suits the overall needs.

Scalability is a pivotal element to decide the technology for the database selection. One important question is whether the database needs to handle a huge set of data or an excessive variety of concurrent customers? Scalability necessities can range significantly depending on elements like projected growth, peak utilization instances, and geographical distribution of users. NoSQL databases like MongoDB or Cassandra are frequently desired for horizontal scalability, while conventional SQL databases like MySQL or PostgreSQL may additionally require more complex scaling techniques.

Moreover, the ecosystem surrounding the database must not be well thought of. One must consider elements like support from the user community, availability of third-party tools and their integrations, in addition to risks associated with vendor lock-in. Open-source databases frequently offer a great community support and a big selection of resources, while proprietary solutions may additionally provide extra comprehensive aid alternatives however come with licensing expenses. Ultimately, the process to select a database must contain thorough evaluation and consideration of these elements to make sure the selected answer aligns with the project needs and overall future goals.

F. Messaging Queue(Optional)

Messaging queues play a pivotal role in present day software program architectures, facilitating asynchronous conversation between distinctive components or services within a system. They act as intermediaries that decouple producers of messages from clients, allowing more efficient and scalable communication patterns. One of the primary positives of messaging queues is their capacity to deal with bursts of traffic and clean out load spikes by storing messages temporarily until they can be processed. This asynchronous nature complements resilience and fault tolerance, as it allows components to perform independently without having to wait for instant responses from other elements of the system.

Messaging queues are utilized in various events across various domains, together with distributed systems, microservices architectures, and event-driven packages. They permit free coupling between components, making it simpler to add new functionalities or scale existing ones without disrupting the system overall. Additionally, messaging queues facilitate dependable message delivery through features like acknowledgments, retries, and dead-letter queues, making sure that messages are processed reliably even within the face of network failures or other network issues. Popular messaging queue systems encompass Apache Kafka, RabbitMQ, Amazon SQS, and Redis Pub/Sub, each offering unique capabilities and alternate-offs suitable for different use instances. In essence, messaging queues are crucial building blocks for growing resilient, scalable, and loosely coupled systems which can effectively cope with the complexities of modern-day software program improvement. They will not be utilized in all applications and hence are classified as optional in our abstract model.

G. Read Replicas(Optional)

Read replicas play an essential position in improving performance, scalability, and fault tolerance. These replicas are copies of a primary database or service which can be synchronized in close to actual-time, allowing for access control to data without affecting the primary system's performance. By distributing read requests throughout multiple replicas, systems can manage higher loads whilst decreasing latency for users. This is mainly beneficial in packages with heavy read traffic, which includes e-commerce platforms, social media networks, and content delivery networks.

Moreover, read replicas contribute to fault tolerance and disaster recovery strategies. In the event of a failure or outage within the primary system, read replicas can seamlessly take over read operations, making sure non-stop availability of data to users. Additionally, read replicas can serve as backups, permitting quick recovery and minimizing data loss. Through load balancing and redundancy, read replicas in computer architecture provide scalability, performance optimization, and robustness, essential for modern-day distributed structures working in dynamic and stressful environments.

H. External APIs(Optional)

External APIs (Application Programming Interfaces) function as gateways that permit applications to communicate and interact with external services, systems, or platforms. External APIs may not be a requirement for all applications but it is a god to know component. External APIs can be integrated with your application based on a need to basis. As the world is growing with open access applications, we can see that the integration of external APIs is increasing. With the added advantage of reducing the stress on your backend, external APIs give us the option to add more data that can be pulled in as necessary.

External APIs expose unique functionalities, facts, or offerings, enabling developers to integrate them seamlessly into their applications without needing to apprehend the underlying

complexities. External APIs empower developers to extend the abilities of their software, get right of entry to valuable sources along with payment gateways, mapping offerings, social media platforms, or records analytics tools, and create revolutionary answers via leveraging the capability provided by means of third party providers. With the proliferation of cloud computing and microservices structure, they have now become essential additives of modern day application development, facilitating interoperability, scalability, and agility in building sturdy and feature-rich applications.

I. Caching System(Optional)

Caching structures are essential components of contemporary computing infrastructure, designed to improve the efficiency and performance of software applications by way of quickly storing regularly accessed records or computations. By storing copies of records in a cache, commonly closer to the application or end user, caching systems lessen the need to fetch records from slower, far off information sources consisting of databases or web servers. This drastically reduces latency and improves reaction instances, leading to a smoother and greater responsive time. Caching structures are available in numerous forms, which include in-memory caches like Redis or Memcached, Content Delivery Networks (CDNs), and browser caches. These structures utilize state-of-the-art algorithms and cache eviction guidelines to manipulate memory correctly and make sure that the most applicable and frequently accessed data stays readily accessible. As a result, caching systems play a crucial role in optimizing overall performance, scalability, and reliability in dispensed computing environments, powering some of the applications and offerings we have interaction with daily on the internet.

J. BI Tools(Optional)

Business Intelligence (BI) tools are essential software solutions that permit agencies to analyze, visualize, and interpret information to make informed business selections. They offer a complete suite of functionalities for statistics extraction, transformation, and loading (ETL), statistics warehousing, information modeling, and reporting. BI tools empower customers to advantage insights into numerous factors in their enterprise operations, along with sales traits, customer behavior, market analysis, and financial performance, through interactive dashboards, charts, graphs, and reviews. By centralizing information from disparate assets and providing it in a user-friendly layout, BI gear facilitate data driven, decision-making, streamline operational procedures, perceive possibilities for evolution, and optimize resource allocation. Leading BI equipment like Tableau, Power BI, and Qlik provide advanced capabilities which includes predictive analytics, system studying integration, and natural language processing (NLP), making them quintessential assets for groups in search of to harness the full capability in their information assets in today's competitive panorama.

IV. PAINTING A PICTURE

Overall, all the components that have been laid out will contribute towards a full-fledged architecture diagram with modifications. The black box solution mentioned is a starting off point but will be applicable to most applications as we build or design a system. The black box solution mentioned above can be altered as necessary and will provide a solution that can fit most cases. To give an example, if we are designing a social media platform like twitter, we can use our black box design with all the components mentioned. The end user can be on various devices like a laptop, mobile device or a tablet. It will connect to the backend via load balancer to ensure that the incoming traffic is well dispersed and the users can access the application without any delays. A messaging queue like Apache Kakfa will be required to connect to various services that are created for specific needs- these can be advertising services, news feed services, announcement services and various other services. The database can be chosen as per the need of the application and we would choose a NoSQL database as we need highly available data. I this case we will choose MongoDB as a starter database. We can design this system with various read replicas to ensure that data is highly available. A caching system like Redis cache can be introduced for data that is frequently accessed and does not have a lot of modification to enhance the user's experience. Data Warehousing connected to external BI tools help business analyze the data and better the consumer's overall experience. Incoming API's can be used to ingest data like news feed and from other existing applications. Integrating with other applications can help users push content without having to access the application.

Overall, using the black box solution as a cheat sheet, a very initial architecture diagram can be created to help build a more detailed architecture by diving in further. An initial diagram would look as follows

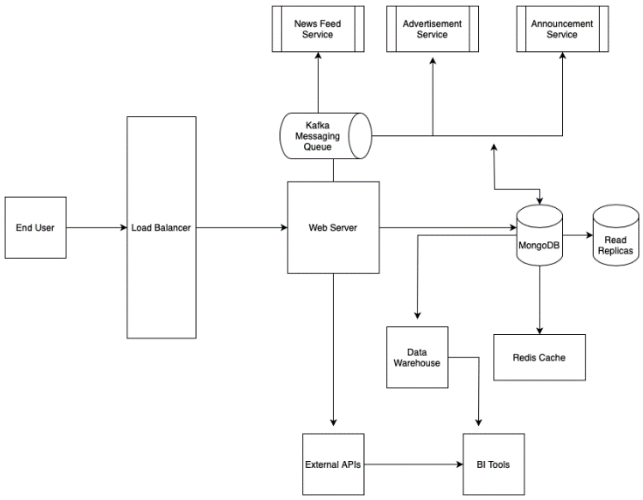

Figure 2: An architecture of twitter like social media platform derived from the black box architecture diagram.

V. CONCLUSION

The abstract model can be a starter system for various solutions. In conclusion, this research paper has delved into the

critical function of abstracting architectural design structures within the creation of new applications within the realm of software engineering. Throughout the paper, we have explored the significance of decoupling structure from precise software contexts, permitting developers the flexibility to innovate, adapt to evolving needs, and scale their applications correctly.

By adopting abstract architecture design techniques, developers are empowered to discover new technology, streamline development tactics, and optimize their packages for more advantageous performance and scalability. The structured framework furnished by using summary structure layout systems fosters a tradition of continuous development and excellence in software program improvement practices.

The paper has highlighted the advantages of abstract architecture layout structures, along with extended agility, decreased time-to-marketplace, and progressed maintainability. Moreover, by using presenting a basis for innovation and new release, abstract architecture design structures enable developers to navigate dynamic commercial enterprise environments with self-assurance and efficiency.

As we look to the future, the importance of abstracting architecture design systems will continue to grow. As technology advances and user demands evolve, developers should continue to be adaptable and aware of innovation in technology. Abstract structure layout systems provide a strategic technique to software program development, ensuring that packages are built on a strong foundation that can face up to the take a look at of time. In conclusion, abstracting architecture design systems represents a key strategy for creating new applications that are both resilient and scalable.

In conclusion, abstracting architecture design systems represents a key strategy for creating new applications that are both resilient and scalable. By embracing the principles outlined in this paper, developers can position themselves for success in an increasingly competitive and fast-paced digital landscape

5.

REFERENCES

1. Bass, L., Clements, P., & Kazman, R. (1997). Software Architecture in practice. https://www.researchgate.net/profile/Rick_Kazman/publication/224001127_Software_Architecture_In_Practice/links/02bfe510fef5da3230000000.pdf)
2. Naik, A. (2024). The Front-End Dilemma: How to Choose the Perfect Technology for your Application. Journal of Computer Science and Technology Studies, 6(1), 211–216. https://doi.org/10.32996/jcsts.2024.6.1.24
3. Cockburn, A. (2001). Agile software development. http://java.cz/dwn/1003/5386_AgileSoftwareDevelopment.pdf.
4. Hofmeister, C., Nord, R., & Soni, D. (2009). Applied software architecture (p. 397). https://dl.acm.org/citation.cfm?id=1611482