Understand Someone Else's Architecture

Team 35 - André Morais, Miguel Rodrigues, Paul Rodrigues

Part 1 - Review the Library architecture with the team assigned to you

Assessment Rubric

Assignment (which architecture): Library System Architecture

Which team are you reviewing: T25

What is your team: T35

Team:

Topic	Strengths and suggestions for the architects*
Diagrams	Fair
Clarity	The figure is clear; it is easy to understand what each class represents, except for the LibrarySystem class, which seems incomplete and needs additional information on what it is responsible for. Also, the connection between the LibrarySystem and the other classes connected to it is not clear enough to understand the relationship between them. Besides that, although there is no legend, it is not necessary to have one to fully understand the diagram.
Consistency	The shapes and lines are used in a consistent way, but there are a few remarks that need to be addressed. First, there appears to be a problem in the line between the Address and Inventory classes, since according to the UML diagram standard, an 'exactly 0' multiplicity does not exist, and it is hard to understand the connection between addresses and the library's inventories. Also, the line between the library system and the user does not need to have words next to the multiplicity, which generates more confusion than it helps and makes us wonder why it is there in the first place. Finally, there are some composition relationships between classes, in particular, the classes User and LibrarySystem and UserMedia and LibrarySystem. Compositions describe a relationship between a whole and its existential parts. And, in this case, saying that a library system consists of (or is a bigger part of) users and user media reservations is not clear enough to explain this relationship and is not consistent with the rest of the diagram or with the relationship between the library system and inventory classes.
Completeness	The diagram does not mention any class or way to implement a fine system for late returns of media by patrons. The diagram also does not mention how the notification system should work, how the library card is used, or many other features that should be presented in the architecture of the system. Also, the collaboration with other libraries appears to be addressed by just a share of other inventory libraries', but the UML itself is not clear enough to explain how that should be implemented.
Sufficient Level of detail	In general, since the objective of this work was that the diagrams and text could provide enough information for a team to implement the system, I do

	not believe a UML diagram is enough for that; a complete system architecture involves various components beyond just class relationships. Including modules, layers, components, interfaces, and their interactions. Class diagrams alone cannot provide enough information to define these architectural elements and their relationships comprehensively. Although, just as a class diagram, the diagram presented provides a few important relationships to effectively build this system later on, like the database of the system, for example, or a few methods that may be part of the system, the library system component should be broken down even further to give enough detail to what it should address.
Text Description	Very good
Clarity	The text description of the architecture is clear and provides a good overview of the system components and interactions. It explains the relationships between classes and the flow of data effectively. The description outlines the functionalities such as user management, catalog management, notification system, inventory management, and collaboration with other libraries in a coherent manner. Furthermore, the text description elaborates on the roles of different users and the interactions they can perform within the system. It also highlights the necessity of inventory management to keep track of physical and electronic media items, as well as the significance of user notifications for late returns and upcoming due dates.
Consistency	The concepts and components mentioned in the text description align with the elements represented in the diagram, but they provide a lot more information than the diagram itself is able to provide. Many components mentioned are not presented in the diagram, for example, the payment gateway or notification system.
Sufficient?	The text gives enough information to understand the key components, functionalities, and interactions within the system. It covers essential aspects such as user management, catalog management, inventory management, user interactions, and collaboration with other libraries. Furthermore, the text provides insights into the system's behavior, such as how patrons check out books, search for items, and receive notifications about their activities. It also mentions the capabilities of staff members in managing the inventory and adding or deleting items from the catalog.
Correctness	Good
Anything missed?	A component for fine management should have been included so it could track overdue items, calculate fines, and manage the payment process for patrons who return items late. Also, a component for generating notifications could be included in the diagram. And a component for handling the integration with other libraries would be a good idea to incorporate in the diagram too.
Will it work?	Even though the system architecture is not complete, I believe that the system presented, accompanied by the text explanation provided, would be feasible to implement in addition to the remarks presented previously.
Presentation	Good
Summary	There is a reasonable level of confidence that this design could be implemented as described, since this architecture includes key components such as relationships between classes, user interactions, and system behavior, which can facilitate the implementation process. But the problem relies on the fact that the diagram, even accompanied by the text, does not provide a lot of the necessary information to implement this design. Class diagrams do

	not fully capture all the details of the system design, leading to ambiguity in understanding certain aspects of the system. A static UML class diagram may not accommodate changes in the design effectively, leading to a rigid design that is challenging to implement. The UML diagram itself can be interpreted in many ways, leading to inconsistency, and it does not specify the technologies, frameworks, or platforms to be used in an implementation.

*Qualitative scale: Excellent (19-20), Very good (17-18), Good (14-16), Fair (10-13), Poor (<10)

Part 2 - Select one of the architecture papers and learn it

Nginx's Architecture

Nginx is an open-source web server implementation written by Igor Sysoeve. Its conception emerges from the fact that Apache's web server implementation, the most used in the industry at the time, was not designed with the evolving web paradigm in mind, especially, given that clients started to open multiple connections to improve loading speeds.

When Apache was architected, in the 1990s, the Internet was a very different place from what is today. Back in the days, most of the content served was static and browsers used to open a single TCP connection with the server and client's low bandwidth was the main bottleneck. Given this, Apache's approach is quite simple – create a new worker thread per connection that handles everything related to a single incoming request and reply back to the client.

Naturally, from this architecture it is possible to anticipate several issues, namely, when dealing with high amounts of traffic. The most obvious one is blocking whenever it is necessary to wait for resources to be freed, which reduces both system throughput and availability. Furthermore, the constant context switching between threads and processes increases thrashing which degrades the system's performance over time.

The nginx's author opted for a completely different approach. One which is event-based, non-blocking, i.e., using asynchronous communication, and where each module has a single responsibility. This permits receiving thousands of requests simultaneously while CPU and RAM usage remain manageable.

At the high level, nginx employs the mediator design pattern. There is a master, i.e., a privileged process, responsible for reading and maintaining the cluster's configuration and launching workers as needed. These workers, with distinct roles based on executing modules, process incoming requests and reply back to clients. The communication between processes occurs through the OS kernel interprocess communication interfaces.

The glue that is responsible for the seamless coordination between the master and workers is the configuration. This is a rule based system whose all the details are under the management of the master process. After interpreting the configuration, all relevant

information is placed in a read-only shared memory region available to all the workers. This can be considered an instance of a shared repository.

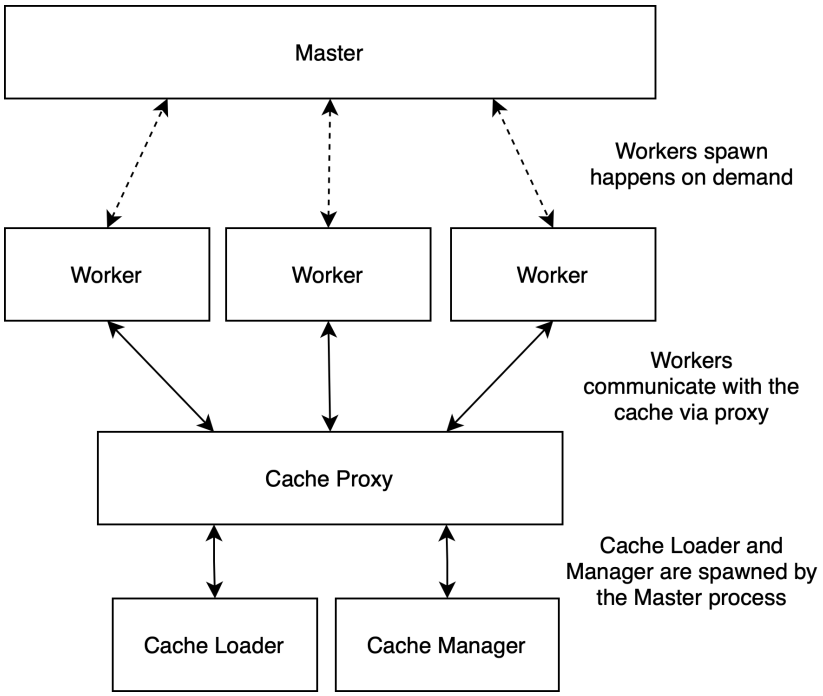

Figure 1 - Hierarchical process tree that constitutes nginx. The arrows denote the communication flow between different processes. The Master process, with privileged access to the system, is the one responsible for spawning all the other processes. As visible, each process has a clearly defined responsibility.

In Nginx's architecture, workers serve as intermediaries, similar to brokers, facilitating communication between clients and upstream servers. These workers manage the exchange of data, ensuring seamless communication flow within the server infrastructure and contributing to nginx's high performance, scalability, and reliability.

Contrary to the Apache's worker threads, nginx's workers are a set of predefined processes within a single-threaded environment that execute a run-loop which leverage the low-level APIs provided by the Operating System kernel. These APIs, namely, the asynchronous communication APIs make extensive use of queues. Such queues accept the incoming requests as they arrive and notifies the run-loop that they are pending, hence the non-blocking behavior. At each iteration of the run-loop, the requests present in the queue are processed by an appropriate callback. Furthermore, the fact that workers are single-threaded removes the burden of thread management which minimizes overhead.

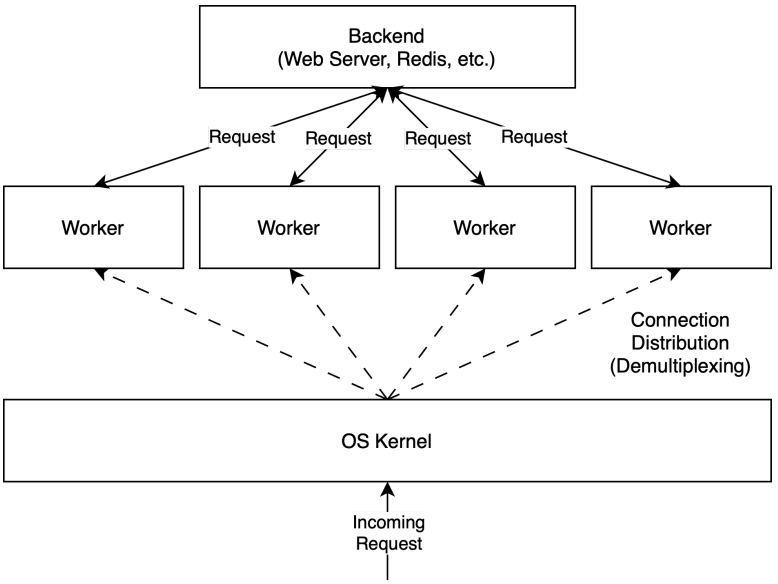

Figure 2 - Nginx architecture leverages I/O event multiplexing using OS kernel system calls, i.e., epoll and kqueue. Upon the reception of a request, the kernel assigns the incoming TCP connection, i.e., a file descriptor, to a particular worker process. After this, the request is forwarded to an appropriate server in the backend.

In nginx, workers are the composition of several modules. In nginx, functionality comes from modules. This means that communication protocols, load-balancing, logging, and filters, e.g., compression, image resizing, or charset modification, are self-contained units of code that can be invoked through callbacks. This is an example of a mixture of data flow styles: pipes-and-filters and batch sequential. In each module we can have either some computation or some data transformation.

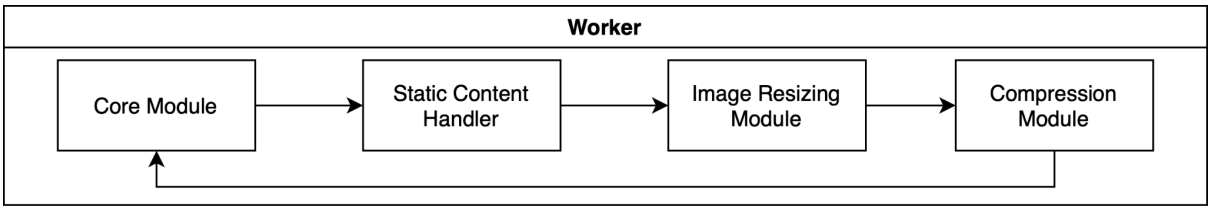

Figure 3 - Example of a worker's structure. In this case, the core module accepts and delivers the request by processing its header and body. The static content handler is responsible for retrieving the content from an upstream server. The following modules filter the request. This structure permits efficient computations because each module shares its output buffer with the input buffer from the following module.

Another aspect worth mentioning is the caching mechanism available to workers. It is a database that caches responses from upstream servers, thus reducing latency by avoiding the network communications. This database is under the belt of the cache loader and cache manager, the latter can be queried by the workers through a cache proxy that performs the advanced I/O operations.

Nginx’s Quality Attributes

A system such as nginx is relevant in the industry thanks to its quality attributes. The most prominent is its performance. Processing thousands of requests concurrently is all about processing a request in the shortest time possible. This is only possible by leveraging the properties of the hardware. Nginx makes an effort to provide hints to the OS kernel, which implements the network stack, and tries to avoid the overhead of switching between user and kernel space as much as possible.

Scalability may happen as a consequence of performance, since the less time it takes for a request to be processed, the more requests are processed per time unit. However, architecture plays a vital role here. Again, processing thousands of requests concurrently demands a non-blocking interface. This is intractable with Apache’s architecture, because no CPU has thousands of cores, thus blocking will inevitably occur under heavy load. On top of that there is thrashing that leads to further performance degradation.

Other important quality attributes worth noting are: flexibility, reliability, and configurability. The flexibility of nginx is inherited from its modular architecture, which permits workers to have very distinct roles according to modules that they have attached. Reliability comes from nginx’s robustness expressed by careful and conservative resource management. One last quality attribute is configurability, which gives the user the opportunity to express its intents in a declarative, concise, and understandable manner. It also hides the complex details from the user perspective.