CS352: Advanced SWE
Assignment 2

Faculty of Computers and Artificial Intelligence
Cairo University

CS352: Advanced Software Engineering
Fall 2024

Assignment 2

Course Instructor: Lamia Abo Zaid

Prepared by:

Name: Sherif Youssef Mahmoud Hamdy
ID: 20221081
Department: CS **Group:** 3CS-S1

What is Meant by Monolithic Software?

Monolithic software is an application whose components and functionalities are interwoven, interrelated, and developed, deployed, and managed as a single unit. In such a case, the architecture generally has one big codebase where all the components of a typical application, such as UI, business logic, and data access layers, are connected in one single executable. It means that in a monolithic architecture, all these independent components cannot function by themselves outside of an application, as they are meant to work integrally. When one partial or subsystem area changes, generally this implies the need for re-testing and redeployment of the entire application. Therefore, it increases the complexity of development and deployment processes.

In the past, monolithic applications were the norm for software development, particularly during times when computer power was limited and managing a distributed system was extremely difficult. For small and medium-sized applications, the monolithic architecture is typically simple to develop and implement. Because all the components of the application are centralized in a single code base, debugging and maintenance are made simpler. However, this kind of structure will inevitably become cumbersome and difficult to scale in larger systems, making the application insensitive to changes or at the very least more difficult to maintain over time.

Technically speaking, monolithic systems are frequently arranged into three primary layers: the data layer (database interactions), the application layer (business logic), and the presentation layer (user interface). These layers are distributed as a single artifact, like a Java.jar or .NET assembly, and collaborate closely. Because modifications to one layer may affect other layers, monolithic programs necessitate thorough dependency management. This makes codebase management crucial. The characteristic that sets monolithic programs apart from more modular, service-oriented designs is their interdependence.

When is it Advised to Build Monolithic Software?

In general, monolithic software is advised for small applications, particularly when the project's scope is well-defined and the development team has limited resources. Therefore, monolithic design could work very well for an application that requires very little complexity and has minimal room for expansion. For instance, because monolithic architecture is affordable and simple to use, it is frequently quite useful for startups or small enterprises with simple, uncomplicated applications. A monolithic approach would require less setup equipment and resources, allowing smaller teams to concentrate on features rather than the headache of administering several services.

When both high performance and low latency are required, the monolithic architecture may be helpful. Data does not need to pass via many services or systems because the various components of a monolithic application are deployed and run together. Because requests are handled more quickly within a single system, this could even lower latency. For high-speed applications, such as real-time analytics or gaming, monolithic architecture may therefore perform better than microservices because inter-service communication may cause delays. Here, a simpler and quicker processing pipeline is guaranteed by the tightly integrated architecture of monolithic software.

Lastly, there may be industries for which monolithic applications would be preferred, where, for regulatory or security reasons, it is beneficial to keep one codebase. In very regulated industries like healthcare or finance, where strict controls on data and processes are common, a monolithic structure indeed simplifies compliance by controlling data and application logic from one place. This makes version control easier, and security protocols may be more easily implemented, as well as system-wide testing to ensure compliance. Therefore, for simple applications or applications that have very high performance and security requirements, monolithic architecture is still an advisable solution.

What Are the Most Common Architecture Styles for Monolithic Systems and Their Advantages?

Architecture styles common for a monolithic system include the layered or n-tier architecture and modular monolith. A layered architecture will typically consist of well-defined layers, usually having a presentation layer, business logic layer, and data access layer. Each layer is responsible for a defined set of functions, and they work sequentially with requests passing from one layer to the other. This is an architectural style with a clear separation of concerns, and all layers can be managed and developed in parallel much more easily. Furthermore, it is easier to debug and test because a problem can be narrowed down to a single layer, making it reliable to use for well-defined domains. Besides, the layered architecture would improve the scalability factor by allowing the caching or load balancing at various layers, which can help a system stay tuned to higher loads and improve the performance for applications that have heavier read-write operations.

Another famous style is the modular monolith. It organizes your application into distinct modules within the same code base but with a stronger emphasis on separation between functionalities. It differs from the traditional monolithic architecture, where changes in one part of the code may affect other parts or anything else; in a modular monolith, it uses well-defined modules for organizing related code in which interaction among those modules is described by well-defined interfaces. Such an architecture allows teams to develop, maintain, and independently test individual modules while still deploying an entire application as a single unit. An advantage in this regard is flexibility: each module, being separate from the rest, can be updated and modified independently without disturbing the remainder of the system; work with the codebase will be easier in general while the application keeps growing. Besides, quite often such modular monoliths allow an easier transition to a microservices approach if necessary, and that's what makes them quite popular for businesses that believe their application complexity is going to grow over time.

Both the layered and modular monolith styles have significant benefits for monolithic applications. Layered architecture is best applied to teams that benefit from a simple, well-defined separation of concerns with easily understood, straightforward tiers. A modular monolith provides more flexibility for complex or ever-changing applications that may someday want to split into independent services. In both cases, scaling can be more effective, development more organized, and testing easier, all within a single, coherent deployment unit. They consequently remain popular choices for applications with special performance demands and teams seeking manageable, adaptable structures in general within a monolithic framework.

References

- Fowler, M. (2015). *Microservices vs. Monolithic Architecture*. martinfowler.com. Link: https://martinfowler.com/articles/microservices.html
- Richards, M. (2015). *Software Architecture Patterns*. O'Reilly Media.
- Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley Professional.
- Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). *Architectural Patterns for Microservices: A Systematic Mapping Study*. Springer.
- Geeksforgeeks: *Monolithic Architecture—System Design*. Link: https://www.geeksforgeeks.org/monolithic-architecture-system-design