Application and Database Coding Standards

Loic Niragire

TIM-8110 Programming Languages & Algorithms

Dr. Phillip Davis

Table of Contents

Abstract

We explore an architecture that can deliver efficient mobile and desktop applications. We aim to allow room to explore emerging languages by segregating critical business components into independent services. We also provide insights into other aspects of the entire development process. We discuss coding standards for database and application development suitable for a cloud-based platform.

1. Introduction

Successful software delivery is a team effort involving business stakeholders, developers, testers, and project managers. The overall objective is to deliver quality products and services derived from technical insights. One of the key ingredients required to achieve this objective is the speed of delivery. The entire process, from concept to delivered solution, needs to get faster and reliable continuously. This approach enriches a culture of innovation and attracts smart creatives ready to push technology boundaries. Therefore, it is paramount that our code design and implementation reflect these viewpoints. However, before addressing design and implementation, there needs to be a common agreed-upon communication building block detailing our technology vocabulary. Such a building block aims to facilitate a dynamic working environment where individual contributors can move across teams with a minimal learning curve. We call this common building block "Coding Standards." Figure one below depicts the foundation described above.

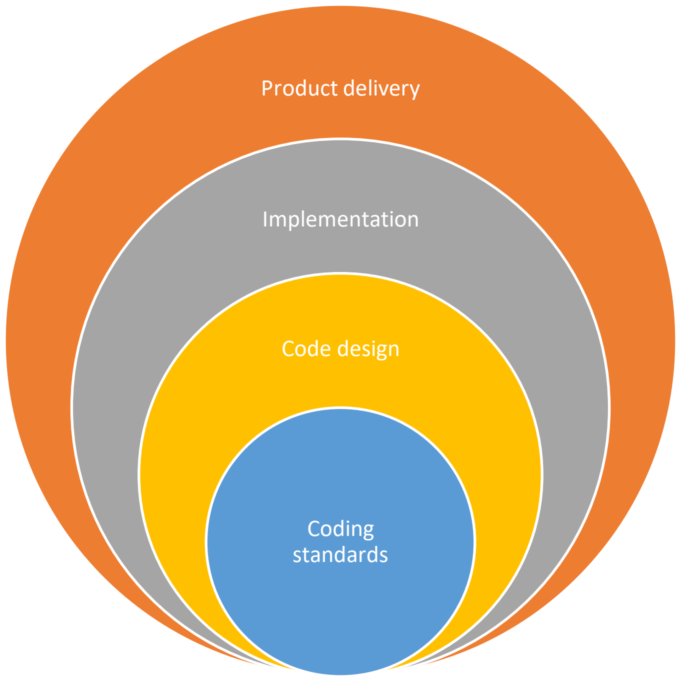

Figure 1 Common communication building block

2. Goal Statement

We aim to create a culture of technical innovation with products that revolutionize the marketplace. This document details our software development strategy to meet this objective. More specifically, it enumerates our standards for database, application design, and development. It is our firm belief that our prized possession is the people behind these products. Therefore, by providing a carefully thought-through minimal set of standards, we aim to facilitate better communication across teams. These standards are subject to evolve in service of our core objective.

3. Languages, Libraries, Frameworks, and Other Tools

Rather than providing an exhaustive list of approved tools, we emphasize a few guiding principles for selecting appropriate tools for the job at hand. By tools, we mean languages, libraries, and frameworks. Our principle is that the abstraction layer in question dictates tool selection. Each layer handles different concerns and carries specific non-functional requirements that inevitably filter our options. Figure two below provides a listing of non-functional requirements considered at various abstraction layers.

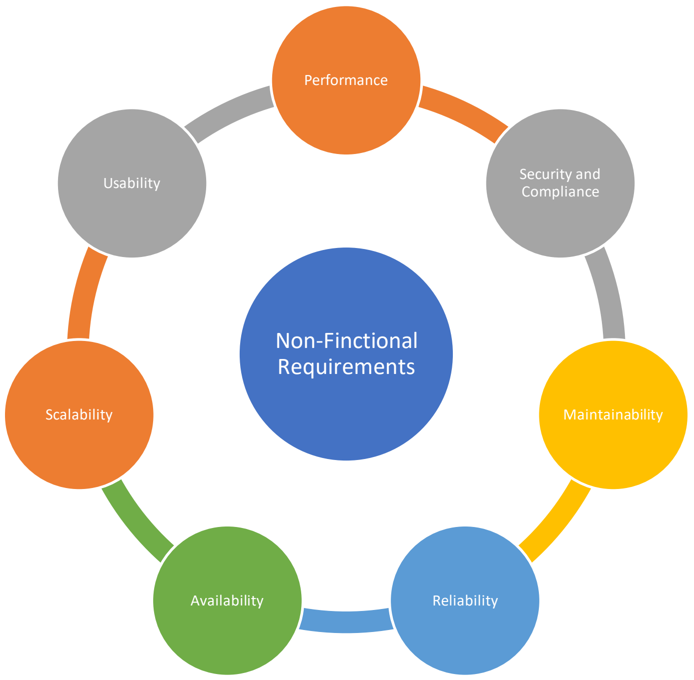

Figure 2 Non-Functional Requirements

We divide our applications into the five layers illustrated in figure three below. Except for the service layer, the degree of non-functional requirements imposed on the remaining layers fluctuates wildly based on the application. The presentation layer, for instance, may implement different JavaScript frameworks or libraries for mobile versus web applications. While scalability may be of some concern at this layer, it is not of great significance compared to the service layer. Therefore, more emphasis is placed on the structure and implementation of our business logic.

Guidelines for business logic language selection focus on statically typed object-oriented languages. The object-oriented programming paradigm is widely adopted across the industry and has been so for a long time. There the primary benefit of choosing this route is resource availability from novice to expert level. Also, they facilitate better abstractions through polymorphism. Lastly, there is an abundance of free open-source community projects providing quality tools that can improve our development process. The benefit, in this case, is that often the open-source community releases features much more frequently. It also gives us the option to contribute features to the code base.

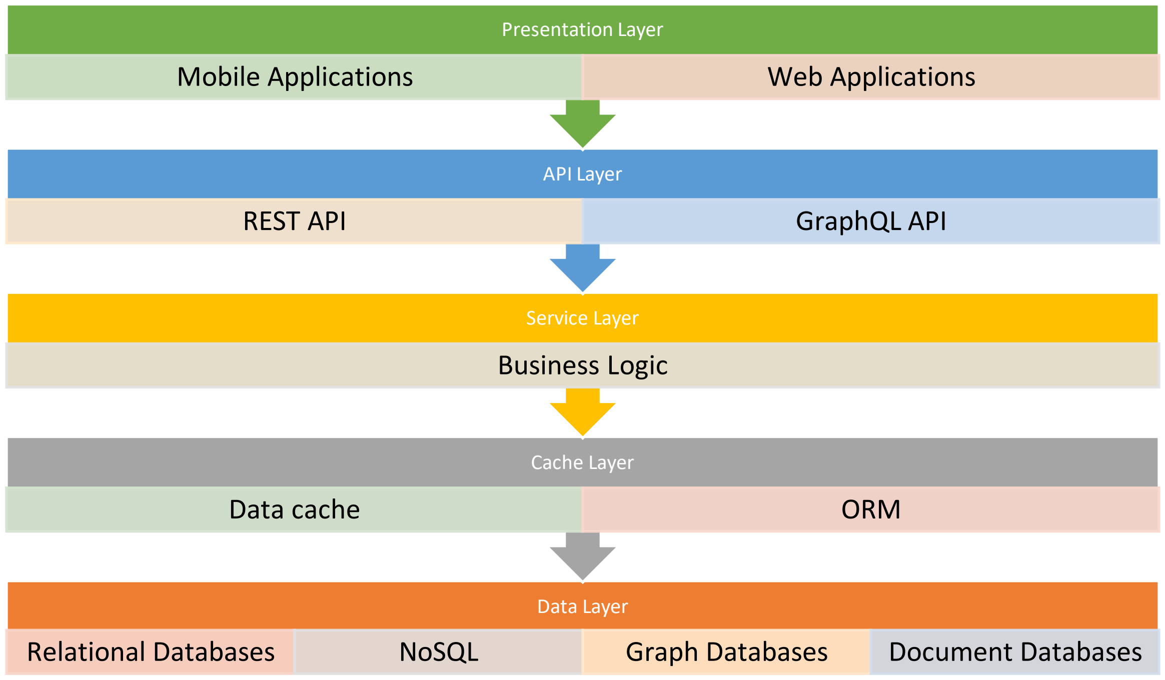

Figure 3 Application layers by functionality

Unless stated otherwise, C# is our de-facto language of choice for business logic implementation. Figure four below highlights C# key features. Additionally, we opt to leverage the .Net Core Framework to support operating system interoperability. Moreover, there are no licensing costs associated with the framework, language, compiler, libraries, and runtime (Microsoft, 2021). All base libraries implementation shall target .NET Standard to maximize portability across all .NET apps, i.e NET Framework, NET Core, or Xamarin. Finally, all unit tests will utilize xUnit Framework (Troelsen & Japikse, 2020)

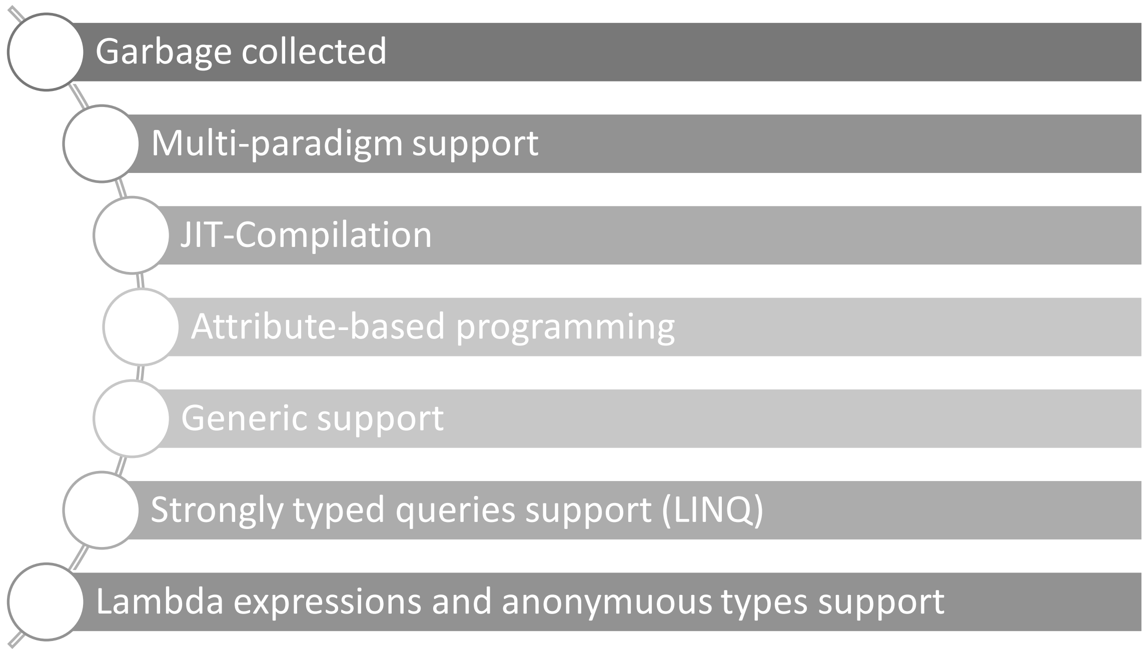
Garbage collected
Multi-paradigm support
JIT-Compilation
Attribute-based programming
Generic support
Strongly typed queries support (LINQ)
Lambda expressions and anonymuous types support

Figure 4 C# features sample

4. Data Management

The platform employs a multitude of data storage strategies including relational, non-relational, document-oriented and graph-oriented databases as highlighted in figure 5 below.

Relational databases	PostgreSQL
Non-relational databases	Cassandra
	Redis
Document databases	MongoDB
Graph databases	Neo4j

Figure 5 Data management solution

Our choice for what to use in any given scenario is influenced by the CAP theorem which was formally proved in 2002 (Hewitt & Carpenter, 2020). Table one below provides an explanation of the theorem, while figure 6 highlights where various databases appear on the CAP continuum.

Table 1 CAP Theorem

CAP Theorem

Consistency	All database clients will read the same value for the same query, even given concurrent updates
Availability	All database clients will always be able to read and write data
Partition tolerance	The database can be split into multiple machines; it can continue functioning in the face of network segmentation breaks

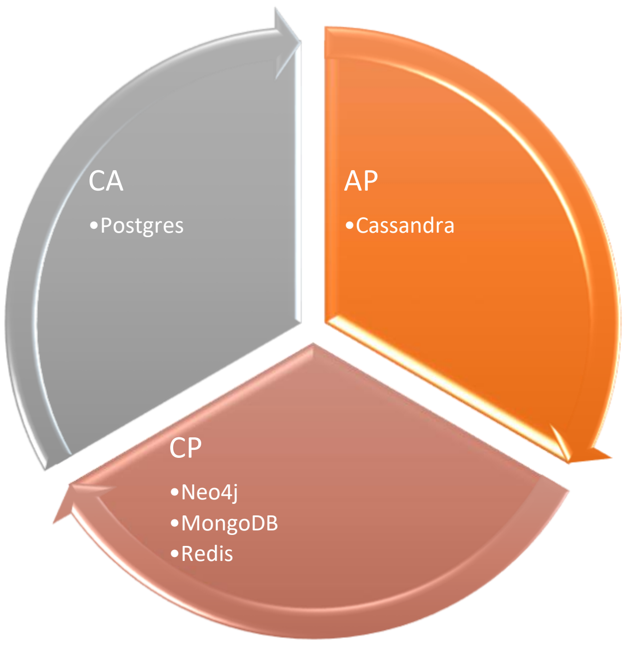

Figure 6 Databases on the CAP continuum

Implications behind the category of each database on the continuum above can be referenced in table 2 below.

Table 2 CAP support description

CAP Support	Description
CA	- Likely to use a two-phase commit for distributed transactions - It may be limited to a single data center cluster since the system will block when a network partition occurs. - Easy to manage
CP	- May require data shards to scale - Data may become unavailable if a node fails
AP	- The system may return inaccurate data but will always be available - Massively scalable, highly available, and partition tolerant.

4.1.PostgreSQL

PostgreSQL is a free and open-source relational database system with an active community. It runs on all major operating systems with powerful add-ons and has been ACID-compliant since 2001 (The PostgreSQL Global Development Group, 2021).

4.2.Cassandra

Cassandra is a decentralized distributed database with no simple point of failure. It features a peer-to-peer architecture and uses a gossip protocol to maintain and keep in sync a list of nodes (Hewitt & Carpenter, 2020).

4.3.Redis

The acronym Redis stands for “Remote Dictionary Server.” It is an advance key-value data store with consequential data types such as strings, hashes, lists, sets, sorted sets, bitmaps, and hyperLogLogs (Da Silva & Tavares, 2015).

4.4.MongoDB

MongoDB is an open-source document database that stores data in JSON-like documents. It features an expressive query language that supports filtering and sorting nested documents. It is a schema-less database that scales-out easily with the ability to index on any attribute (MongoDB, Inc, 2021)

4.5.Neo4j

Neo4j is an open-source native graph database that treats the relationships between data as equally important as the data itself. It supports ACID transactions and features efficient node access in constant-time, and can traverse millions of connections per second per core (Neo4j, 2021)

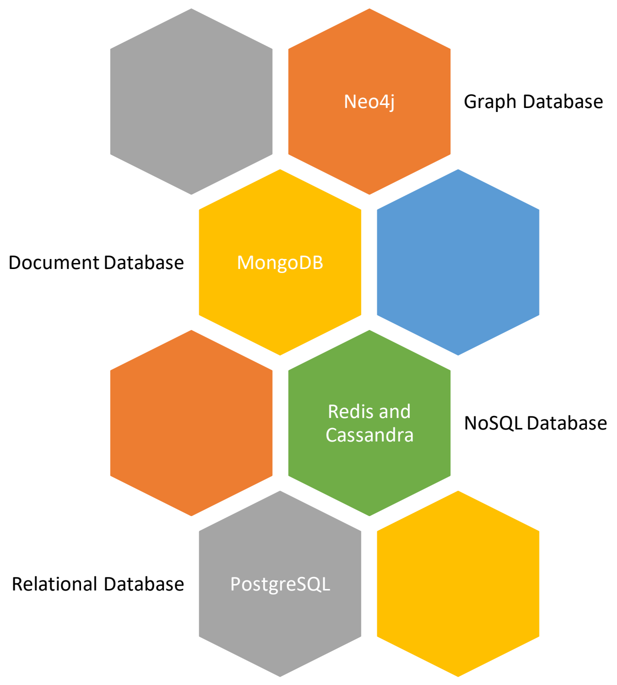

Figure 7 Data storage options

5. Coding Standards

Our coding standards are highlighted in five principles – depicted in figure seven below: functional, readability, testability, extensible and scalable. The first and foremost important standard is that a delivered solution has to fulfill all functional requirements as outlined in requirement analysis. It has to solve the problem it was intended to solve. Following that, the solution has to be readable and well structured. It is often the case that individuals from different teams have to maintain or update any given codebase. The next important factor is how testable the solution is. Briefly stated, it is expected that all business logic be fully unit tested and that these tests should be automated and repeatable. The last two attributes, namely scalability and extensibility, deal with ease of improvement. This is achieved through loosely coupled coding practices with well-defined modules and data flows.

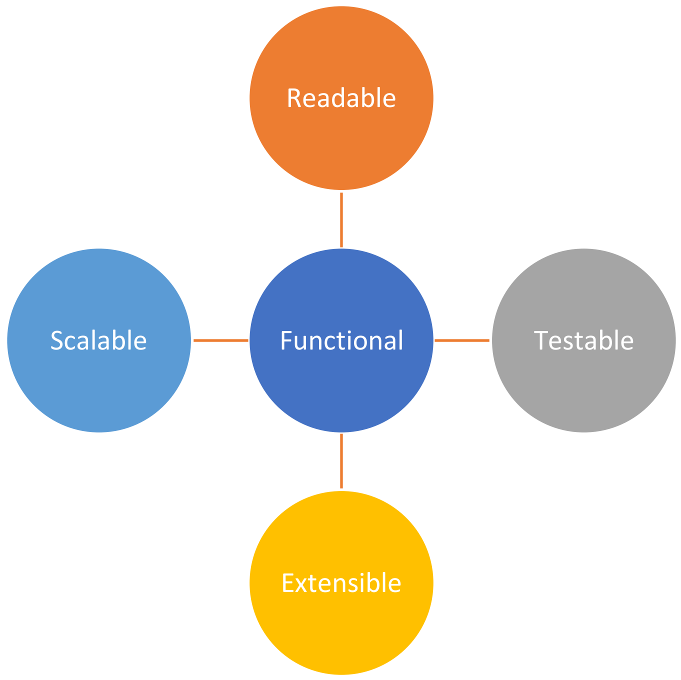

Figure 8 Coding standards

6. Conclusion

The standards outlined in this document facilitate a flexible architecture that can support mobile and desktop application development by leveraging the same set of APIs. We place more emphasis on our core business logic as illustrated in figure three and provide both REST and GraphQL endpoints through our API layer. Whether implementing a mobile or a desktop application, this essentially is a presentation layer concern. GraghQL endpoints are preferred for internal consumption because they offer optimal querying options to minimizes payloads, while REST endpoints are industry standard for external client integration (Banks & Porcello, 2018). While Go programming language shows many promises, it is still relatively new in comparison to C# and that makes it harder to find expert developers. Swift on the contrary can be leveraged at the presentation layer in our current architecture. For a complete and most up-to-date listing C# coding conventions, interested reader is directed to (Microsoft, 2021)

7. Reference List

Banks, A., & Porcello, E. (2018). Learning GraphQL. Sebastopol: O'Reilly.

Da Silva, M. D., & Tavares, H. L. (2015). Redis Essentials. Birmingham: Packt.

Hewitt, E., & Carpenter, J. (2020). Cassandra - The Definitive Guide. Sabastopol: O'Reilly Media.

Microsoft. (2021, March). C# Coding Conventions. Retrieved from docs.microsoft.com: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

Microsoft. (2021, March). Dotnet. Retrieved from dotnet.microsoft.com: https://dotnet.microsoft.com/platform/free

MongoDB, Inc. (2021, March). MongoDB. Retrieved from mongodb.com: https://www.mongodb.com

Neo4j. (2021, March). neo4j developer. Retrieved from neo4j.com: https://neo4j.com/developer/graph-database/

The PostgreSQL Global Development Group. (2021, March). About PostgreSQL. Retrieved from postgresql.org: https://www.postgresql.org/about/

Troelsen, A., & Japikse, P. (2020). Pro C# 8 with .NET Core 3. Minneapolis: Apress.