Enterprise software doesn't mean big/scale

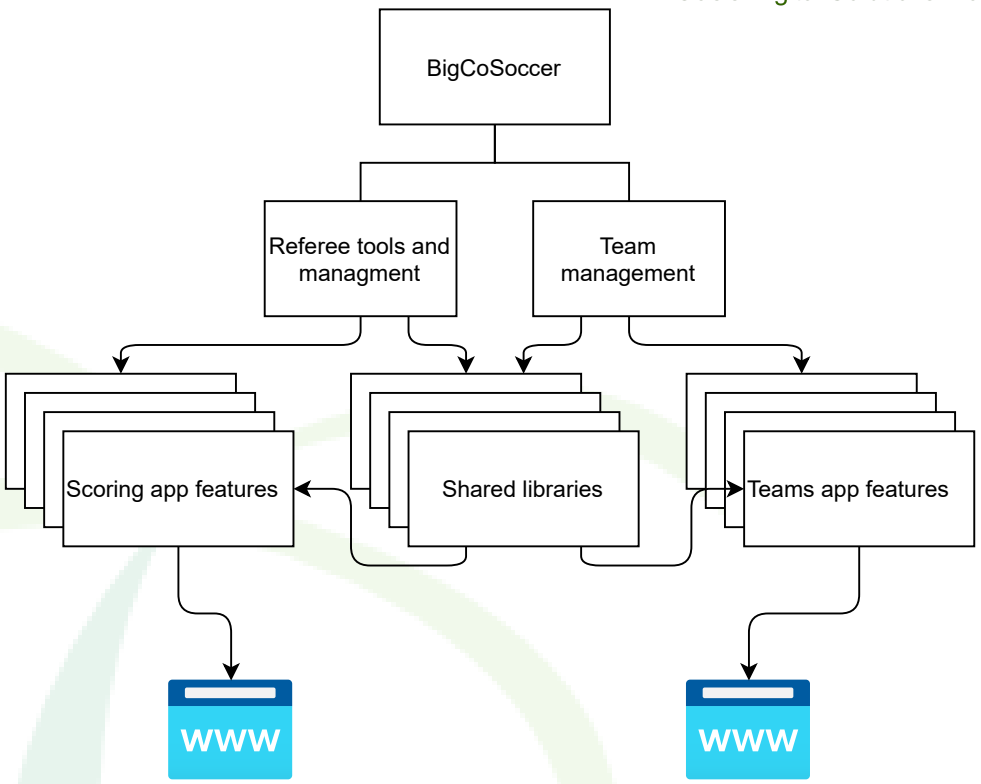
Paul Spears - @TheEvergreenDev - Oasis Digital Solutions Inc

Oasis Digital Solutions Inc.

What makes enterprise software development different is not necessarily that it is "big" or "at scale". Enterprise development can be identified with one characteristic: there are major considerations and/or constraints in architecture, design, or implementation that are external to the immediate application team, but internal to that team's organization. The two most common constraints are the need to share/reuse organizational resources, and the ability to withstand infrastructure, architectural and/or visual design changes. These constraints create unique challenges that are often not found in personal code bases, small team projects, or even software that is being built at scale by a unified team.

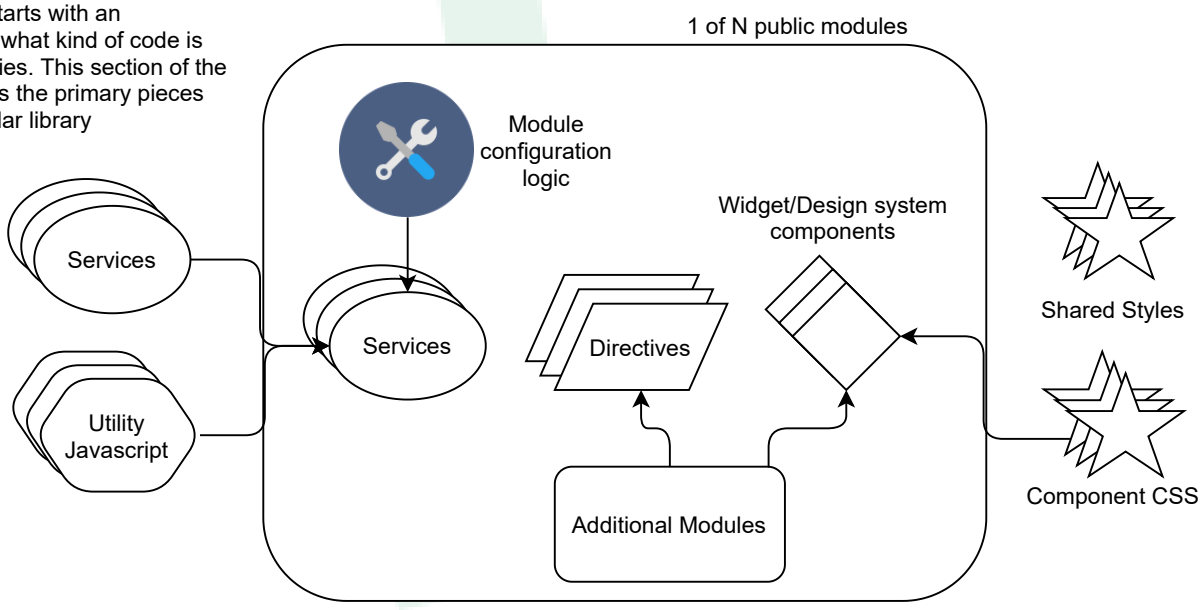
Practical examples of such constraints may include a mix of:

- the need to use an existing company authentication system
- the need to use a company defined coding style or best practices guide
- preparation for a library migration
- restrictions in library choice and access
- coordination with multiple teams
- special considerations for production up-time
- the need to pass review or audit from another part of the organization



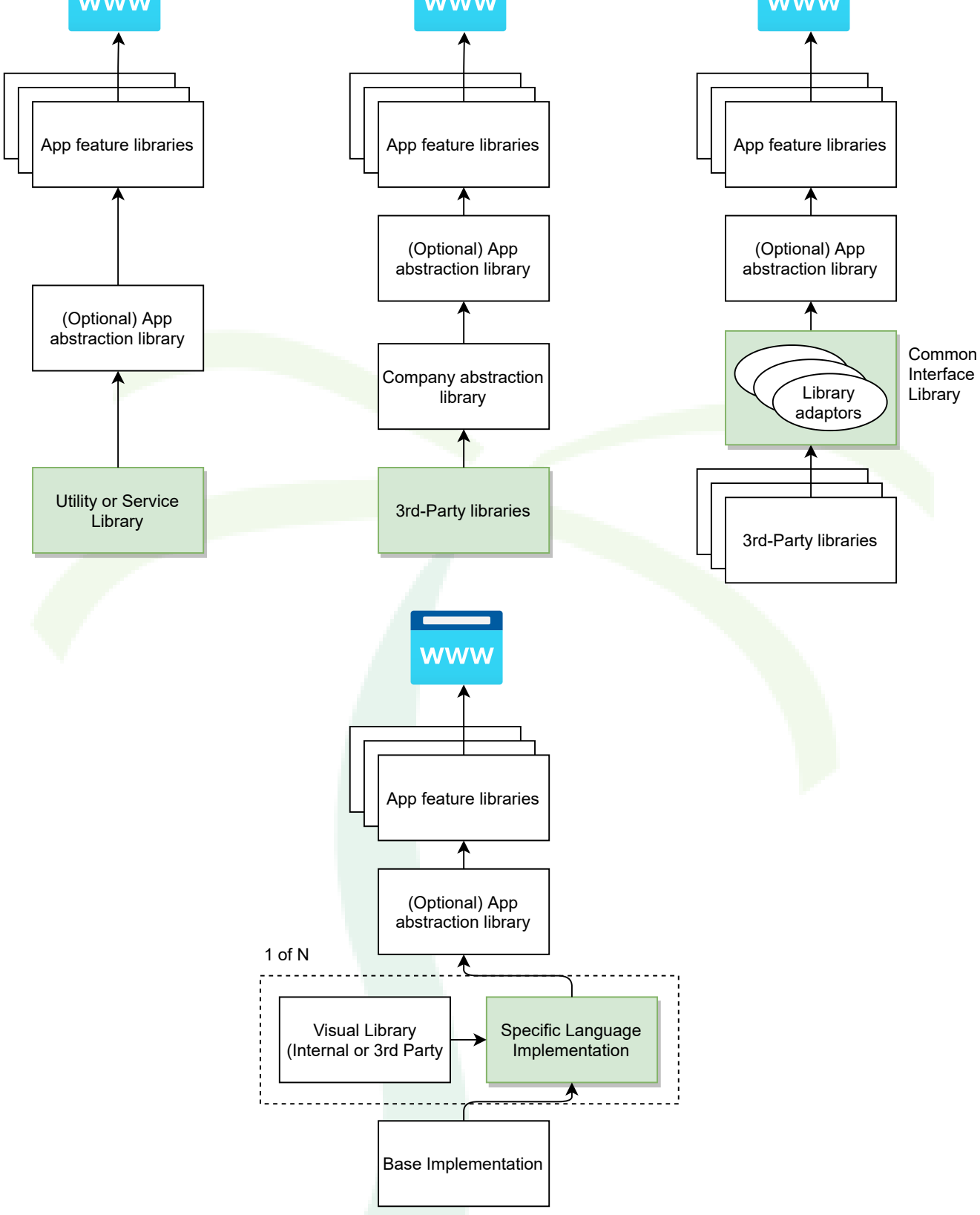
What's in a library?

Enterprise development often manifests as a large collection of libraries. Knowing how to organize and manage these libraries is a key skill. This starts with an understanding of what kind of code is exposed by libraries. This section of the diagram illustrates the primary pieces found in an Angular library



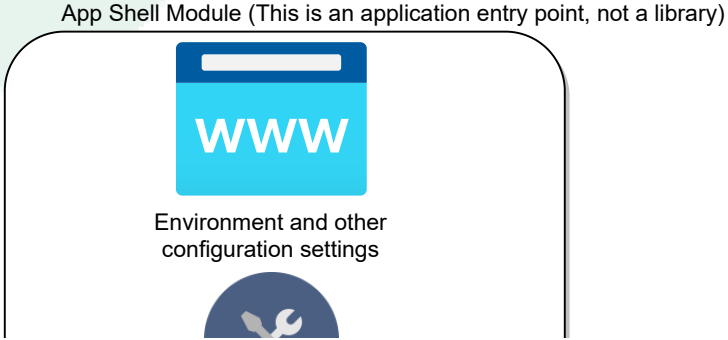
Layering techniques for shared libraries

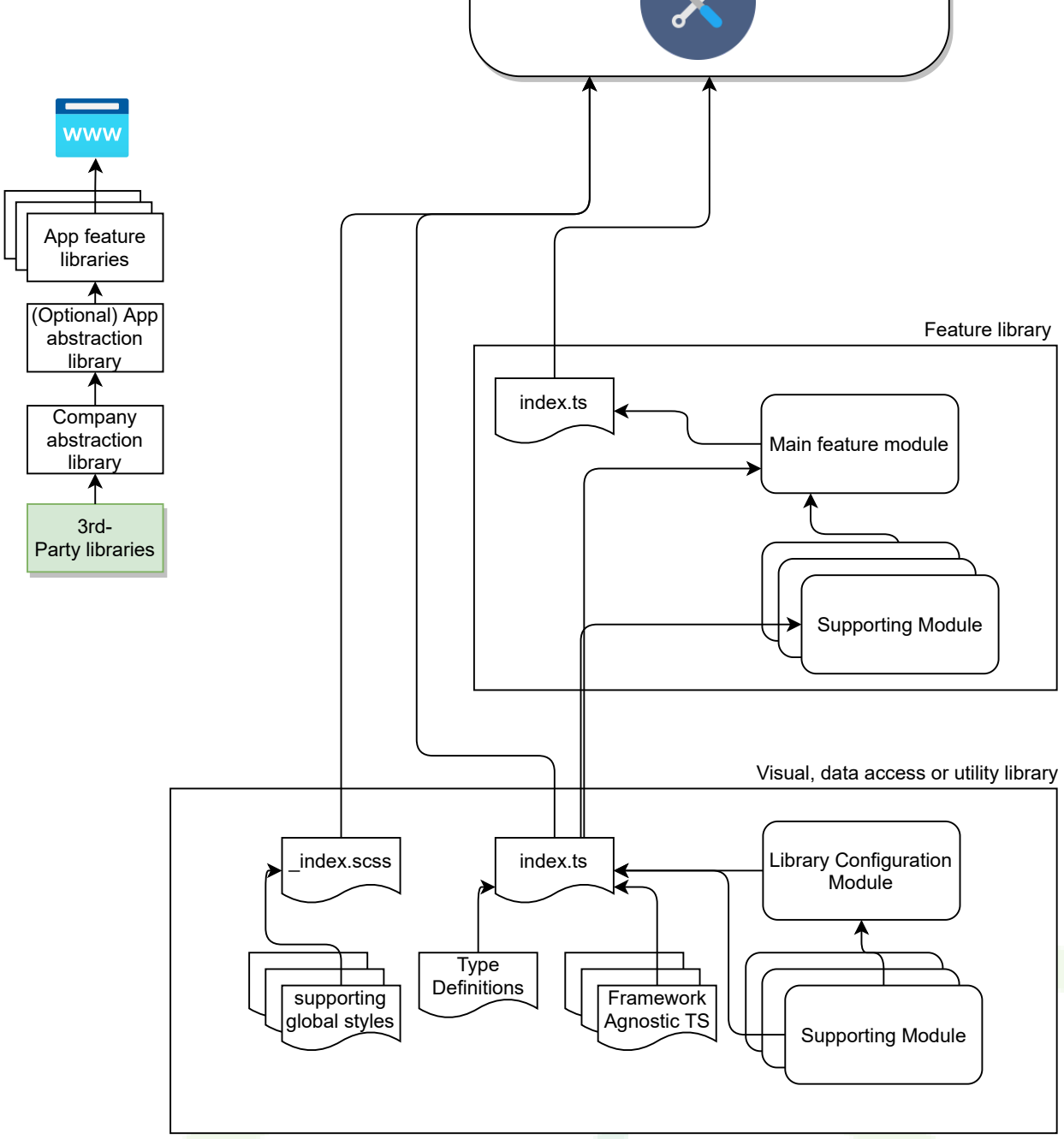
In order to remain flexible to changing demands within an enterprise environment, libraries are usually built in a layered architecture. The following four diagrams illustrate ways in which libraries can be stacked together to minimize complexity for application authors and minimize the impact to library consumers in face of change. In each of the diagrams, the core functionality to be shared is highlighted in green.



Modules and libraries

It is important to understand the relationship between Angular modules and libraries. This diagram illustrates how files and modules are shared across libraries





Modules, Components and Dependency Injection

The diagrams up to this point illustrated how the build time constructs of libraries, modules, and files related to one another. In addition to these build time constructs, Angular development requires the understanding of its run time dependency management system. This diagram outline an application module and component hierarchy as well as a few services. These entities are left unlabeled to allow an instructor to walk through multiple scenarios in class.