Web Application Architectures Seminar

Yasmin Mushahdi
Kurs 1dv612 Web Architecture
2020 feb-march

1. What is a web application?
An application is a program or a software package that allows a user to run a set of functions by interacting with the software.
There are different types of apps distributed on our devices, desktop apps, mobile apps, and web apps.
To use a mobile or desktop app the user must first download the software on to the device and open it like any other program to access the functionality. While downloading the software requires an active internet connection there is no need to be online while using the app, like any other program.
But a web app stands out a bit because you need a web browser or web client (Firefox, Chrome, Safari..) to open a web application. The web app is dependent on an active internet connection to run and it does so by using a protocol, usually HTTP, but there are others like FTP, WebSocket.

2. What does web application architecture mean?

- **What does a web application architecture include?**

Web application architecture is the process of structuring and designing the technical, functional components in a web application, and then lastly the design of the visual aspect of the application before deploying the final product. The architecture includes a variety of components, which are all dependent of the type of an application, and usually an architecture consists of three main layers:

Presentation layer
The presentation layer's main purpose is to display information into readable format(ASCII, GIF, JPEG, MPEG) and collect user input to pass it further to processing in an appropriate format. Code conversion.
It also handles encryption and decryption and exchanges the data in a safe way.

Application/Business layer
Business layer coordinates data between the user interface and the data storage, by having access throughout the system components this layer is where all logic is connected and acted upon.

Data access layer (DAL)
This layer handles all the communication and accessing of data from the data storage.

The traditional N-Tier architecture consists of a total of 7 layers, each layer represents the independent component of an application.

3. How have architectures for web applications developed over time?
Earlier there were only websites and for the most part they consisted of static information. In each HTTP request the user received a full HTML page. Client code and server code was tightly coupled and as mentioned before, all layers were simple and well defined.

But with time, more users, and more functionality the architecture had to change to be used more effectively. Now only the requested information gets loaded and not the entire page. This created new changes in how to arrange components and layers to create an effective and less resource wasting architecture.

Basically there was the static web sites, with the Internet's growth and expansion this changed to create a more efficient web with only loading content which is requested, and this introduced to the dynamic page, where content depended on various factors (who the user is, what content is requested, calculations depending on outside factors).

4. What architectural design- and anti patterns are there for web applications?

There are many different architectural design patterns and some are more alike than others and they all are propositions to solve a range of different problems. According to [Peter Wayner](https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice), there are at least 5 architectural design patterns that all developers should know about.

- Layered architecture -
- Microkernel architecture – has a code base but uses plugins for a various other logic.
- Micro service based architecture -
- Event driven architecture
- Space based architecture

Anti patterns
Architectural anti-patterns are referring to mistakes or problems occurring while trying to achieve a certain structure. Any pattern which is not followed consistently tends to become an anti-pattern or grow into something which was not intended. There are many known anti-patterns within software development and some are more familiar than others. To name a few:

- [**God Object**] – a class that does EvErYtHiNg, this means that a class progressively is holding all the responsibilities, and   this becomes hard to maintain. By having a class with too many responsibilities, it is easy to forget what, where, and how the code works.

- [**Big Ball of Mud**] - is the term for misusing layers architecture core practices – keeping the layers separate. Keeping it is neat and tidy.

- [**Reinventing the Wheel**] - happens when the refactoring of the code has become a distant matter. By skipping to evaluate where code practically solves the same problem and reuse that part, by maybe adding a library, or rewriting code to become more generic eventually causes this pattern.

Assignment

Create an artifact describing the architecture of the examination application. The following requirements must be met.

- The documentation for your web application must be stored on the GitHub wiki associated with your private examination repository.
- The documentation must contain a picture describing the overall system.
- The type of communication between subsystems must be documented.
- The behavior, interaction between subsystems and users must be documented.

Initial planning:

So far this picture describes the initial plan explaining the overall architecture of and between subsystems, while the overall system may be changed.

Inner communication between **JS(Node)**, and the **server** will use HTTP.

I chose to work with JavaScript because it is a complete language for web development, it is suitable both client and server side code. The language is supported by useful frameworks and libraries that make it easy to work with Github API further on.

Nginx will communicate with the Internet by HTTPS.

Now, I chose Nginx because of the useful configuration options, it works asynchronously which means that it serves without blocking the other requests. If I want to learn best practices and use performance promoting features then Nginx is a good tool.

Github communication using HTTPS:

- To request and data, eg. organizations
- To receive Webhook data