System Design

Module 1 Part 1

Tiers of Software Architecture

Tier is a logical and physical separation of components in application or service.

Components

1. DB

2. Backend Application Server

3. UI

4. Messaging

5. Caching

etc..

These components make up a web service.

Tiers are defined at component level and not at code level.

Single Tier Applications

User User interface, backend business logic and database reside on the same machine.

Examples: Desktop applications like PC Games, MS Office, Photoshop etc.

Pros of Single Tier Applications

1. No network latency since every component is located on the same machine.

This adds up to the application performance. However, the actual performance of the application depends on the hardware requirements and how powerful the machine it runs on is.

2. Data privacy and safety, is of the highest order in single tier apps since the user data is in their machine and doesn't need to be transmitted over network for persistence.

Cons of Single Tier Applications

1. Application publisher has no control over the application. Any patches or application

2. Upgrades are handled by the customer.

3. Software testing has to be thorough since there is no room for mistakes.

4. Code is vulnerable to tweaking and reverse engineering.

5. App performance and look and feel can be inconsistent as the app rendering largely depends on configuration of the user's machine.

Two-tier Applications

- Involves Client and Server
- Client contains UI with business logic in one machine.
- Backend server includes the database running on a different machine.
- DB server is hosted by the business and has control over it.

Examples: To-do list apps, mobile games

Why to have business logic on Client side in Two tier apps

- Having business logic on client side cause code vulnerabilities. But for simple applications like to-do list, code vulnerabilities won't harm the business.
- Having code on client side reduces network calls and app latency.
- The only call made to backend server would be for data persistence.
- In mobile games, the source code will be huge. Having it on the client device and calling backend for game state persistence benefits in terms of latencies and also less servers calls means less money spent to keep the servers running.
- Picking Two tier architecture for a service depends on the business requirements and use case.

Three tier Applications

- Three-tier applications are pretty popular and largely used on the web.
- UI, Business logic and database all reside on different machines and thus have different tiers.
- They are physically separated.

N tier Applications

- Has more than 3 components
- Ex:
  - Cache
  - Message Queues for asynchronous behavior
  - Load balancers
  - Search servers for searching through massive amounts of data
  - Components involved in processing massive amounts of data
  - Microservices, web services

Why so many tiers?

1. Single responsibility principle
    a. Giving dedicated responsibility to a component. Ex: DB should only persist data and not hold any business logic like stored procedures. Having business logic in DB will need more effort when we need to [plug to another DB. The business logic also need to move to the new DB or fit into the application code.
    b. Keeps things cleaner. Components will be loosely coupled and multiple teams can contribute to development of different components.
    c. Changes/Outages in one component will not affect the entire application. Only parts of the application may get affected. Ex: DB outage will make the DB server down and only the parts of app depending on DB will be affected.
2. Separation of concerns
    a. Keeping components separate makes them reusable
    b. Enables to scale the service easily when things grow beyond a certain scale in future.

Layers vs Tiers

1. Layers
    a. User interface, business layer, service layer, data access layer.
    b. Layers are at code level.
    c. User => Controller => Service => Data Access => Database
    d. Logical/conceptual organization of code.
2. Tiers
    a. Physical separation of components.

Web Architecture

- Client server architecture is the fundamental building block of the web.
- Works on request response model.

1. Client
    a. Hold the UI. Ex: Mobile, desktop etc.
    b. Types of client: Thin client, Thick client
    c. Thin Client - Holds no business logic. The client in 3 tier applications.
    d. Thick Client - Hold some part of business logic. Like 2 tier applications.
2. Server
    a. Every online service needs a server to run.
    b. Types:
        i. Application servers
        ii. Proxy server
        iii. File server
        iv. Virtual server
        v. Data storage server
        vi. Batch job server and so on
    c. All components of web application need a server to run, be it db, message queue, cache or any other component.

Communication between client and server

- Entire communication happens over HTTP protocol, protocol for data exchange over WWW. It is a request-response protocol.
- HTTP is a stateless protocol and every process over HTTP is executed independently and has no knowledge of the previous process.
- Every client hits a REST endpoint to fetch data from backend.

REST API

- Acts as gateway or entry point to the system
- Encapsulates all business logic and handles all the client requests, irrespective of the type of client(desktop, mobile, etc.)

HTTP Push and Pull

There are two modes of data transfer between the client and the server

1. HTTP PULL
    a. For every response, there has to be a request first. Default mode of http communication.
    b. If a client wants to get updated data from server, it makes a request. What is the data is not updated? It may make repeated requests. These unnecessary requests from the client uses unnecessary bandwidth and increases the server load and may eventually bring down the server.
    c. HTTP GET call or UI via AJAX polls the server repeatedly every X unit of time set by the developer.
2. HTTP PUSH
    a. To tackle above scenario, client sends a certain request to server only once. After the first request, the server keeps pushing the new updates to client whenever they are available.
    b. This is also known as a callback.
    c. Cuts back network bandwidth usage and the load on the server by notches.
    d. Ex: User notifications.

Clients use AJAX to send requests to server in both HTTP PULL and HTTP PUSH

Technologies based on HTTP PUSH mechanism

1. Ajax long polling
2. Web sockets
3. HTML5 Event source
4. Message queues
5. Streaming over HTTP

HTTP Push

- Time to live(TTL): Time after which the browser kills the connection after a HTTP PULL request if the client doesn't receive a response from the server.
- Open connections consume resources and there is a limit on how many open connections a server can handle at one point. The server may run out of memory.

Persistent Connection:

- Network connection between client and server that remains open for future requests and responses, as opposed to being closed after a single communication.
- Persistent connections facilitate HTTP PUSH communication between client and server.
- Brower kills open connections every x units of time. To have persistent connections, we have heartbeat interceptors.
- Heartbeat interceptors are blank responses between client and server to prevent the browser from killing the connection.
- Persistent connections consume a lot of resources compared to HTTP PULL.
- Use cases of persistent connections: Browser based multiplayer game has a large amount of request response activity within a limited time than a regular web application. From user experience standpoint, using persistent connections is apt.
- Long opened connections can be implemented by multiple techniques such as AJAX Long Polling, Web sockets, Server sent Events, etc.

HTTP PUSH Technologies

Web Sockets

a. Preferred when we need bi-directional, low latency data flow from client to the server and back.
b. Typically preferred for chat applications, real time social streams, browser based massive multiplayer games,etc.
c. Web sockets can keep the client-server connection as long as we want.
d. Does not work over HTTP, the mechanism runs over TCP.
e. Both client and server should support web sockets mandatorily.

AJAX - Long Polling

f. Lies between AJAX and web sockets.
g. The connection in long polling stays a bit longer than polling.
h. After receiving the request, the server does not send an empty response. Instead it holds the response until it finds an update to be sent to the client.
i. If the connection breaks, the client has to re-establish the connection to the server.
j. Upside: fewer requests to be sent to the server then regular polling mechanism.
k. Used in simple asynchronous data fetch use cases when we don’t want to poll the server very often.

HTTP PUSH Technologies

HTML5 Event Source API and Server Sent Events

a. Client does not poll for data, server pushes the data to client when something is available.
b. Incoming messages from server are treated as events.
c. Cuts down network bandwidth usage drastically as there are no blank request-response cycles.
d. To implement this, backend should support the technology and the UI should use HTML5 Event Source API to receive incoming data from backend server.
e. Once the client establishes a connection with the server, the data flow is one directional only, from the server to client.
f. SSE is ideal for scenarios like real-time Twitter feed, displaying stock quotes on the UI, real-time notifications, etc.

Streaming over HTTP

g. Ideal for cases where we need to stream extensive data over HTTP by breaking into smaller chunks.
h. Possible with HTML5 and JavaScript Stream API.
i. Both client and server must agree to conform to the streaming settings to stream data.
j. Makes possible to watch partially downloaded video chunks on the client.
k. Used for streaming multimedia content like huge images, videos etc. over HTTP.

Client-Side vs. Server-Side Rendering

- Browser components
  - Browser engine
  - Rendering engine
  - Javascript interpreter
  - Networking and UI backend
  - Data storage etc.
- Using these components,the browser converts the web page response it gets from the server into HTML page.
- Rendering engine constructs the DOM tree.
- All these activities take time.
- To cut down this time, the server can generate HTML and send that directly to the UI. This is called server-side rendering.
- Server side rendering is perfect for delivering static content such as WordPress blogs. Also crawlers can read generated content easily.
- Modern websites are highly dependent on AJAX. Every AJAX request will trigger a server side rendering and causes latency increase, where as with client side rendering, only the a part of the page is updated.
- For modern dynamic AJAX websites, client side rendering works best.
- We can leverage a dynamic approach to get the best of both worlds.