Adv Web Dev Architecture
Lecture 2

Amir Amintabar, PhD

Outline

- 1 Course structure
  - Course outline
  - Announcements on the learning hub front page
  - Assessments, assignments and term project topics
- 2 What is internet software architecture all about?
- 3 The architecture we focus on: API-centric, service based
- 4 JSON as a way of data payload in API communications

1

What is internet software/Application Architecture?

2

- **Architecture** refers to interaction/relation among components.

- In building construction: interaction/relation among walls , floors, stairs
  - Focus is on space usage, flow, beauty, cost, style

- In computer architecture: interaction/relation among CPU, Memory, Storage , I/O …
  - Focus is on speed, power usage, efficiency, computational power

- In web application: interaction/relation among UI, Database, logic, etc
  - Focus is on reliability, scalability, security, ease of implementation, modularity

What are the benefits of studying architectural patterns?

- Architectural patterns help us better evaluate the efficiency , robustness, security, performance, modularity, cost of maintenance, development and extension, cost of deployment our designed application:
- **Efficiency** (e.g. I am looking for a lightweight server, what backend technology works for me),
- Cost of **maintenance** (e.g. my application is written in JavaScript and ASP .net, I need two developers. If I switch to nodejs I only need one developer good at JavaScript)
- **Development and extension** (e.g. my client wants me to add this new module to Django , but my Python developer left my company. Can I ask my nodejs guy to create a service and hook it up to Django via API? The answer is yes only if my Django application is originally developed as an API-centric application)

What are the benefits of studying architectural patterns? …

- **Robustness** (e.g. is there a single point of failure in my design? How the requests' load is distributed among modules. What if the DB fails)
- **Security** (e.g. how easy I can improve the security? Should I look into every single line of my code, or there is a gateway component I originally designed which is the only portal being in touch with the outside world. That gateway is the only place I need to secure)
- **Modularity** (e.g. each module/service must carry one main functionality)
- Cost of deployment (e.g. do we need a dedicated server? If we use Ruby on Rails, perhaps we need a dedicated sever. Do we need docker?)

There are many recognized architectural patterns and styles, among them:

Layered (n-Tier) Architecture

- layers = functional elements.
- Presentation layer (UI layer)
  - interaction with the end-user is handled(e.g. form page)
- Business layer (business logic)
  - controls the application’s logic (e.g. decision making, routing requests)
- Application layer
  - where the core functions of the app is implemented(e.g. libraries)
- Data access layer
  - where access and retrieval of data are handled (e.g. databases)

Service Oriented Architecture (Microservice)

- The components could be each providing a separate. All forming a collection of loosely coupled (almost independent) services. When a service reaches a specific level, it could be created in the form of a separate, stand alone app.
- We could as well split the services into smaller pieces so that each service executes a single functionality called **microservice**.
- Each service can reside in a separate machine. Be written in a different technology stack and can be up and running separately. For example the messenger app is a service to the facebook app. If facebook shuts down, it does not necessarily cause the messenger app to shut down too

SOA: Service-Oriented Architecture
MSA: Microservice Architecture slightly different but refer the same thing in this course

Monolithic Architecture

- Opposite to microservices is monolithic architecture where in this context, means composed all in one piece.
- A monolithic architecture is the traditional unified model for the design of a software program. Often entire software resides in one machine, entire software is written in one backend technology stack.
- For example in a monolithic web application entire application is usually written in one backend technology, say Java and hosted in one machine. If you want to add something new to it you
- 1- shut down entire application
- 2- add the new feature
- 3- (may need to) recompile *everything* before deployment
    - All application logic (UI, business logic, data access) is bundled together in a single deployable unit. Simple to build and deploy but hard to scale and maintain as complexity grows.

Microservice arch. pros and cons

- Pros:
- It enables you to add new components to the system and fix any bugs without shutting everything down.(you can fix messenger app without shutting down facebook)
- One service is meant to be responsible for one function (for example, messaging, uploading files, registering users, and so on). That’s why you can assign different teams to work on each service and thus speed up the development process
- Cons
- Its complicated compared with monolithic. Since each service could be hosted in a different part of the globe, the only way the services can communicate is via API. Thus you have to deal with so many API calls!
- Nonetheless, many large firms choose the Microservice architecture: Uber, Netflix, Amazon, Ebay, Sound Cloud, Groupon, realtor.com (Richmond BC)
- Breaks the system into small, independent services, each responsible for a specific business capability.Improves scalability and flexibility but increases operational complexity.

Summary and more by ChatGPT (we need to verify)

- **1. Monolithic Architecture**
- **Purpose:** Quick to build and deploy for small apps or MVPs.
- Best for: prototyping, or apps that don’t expect frequent independent feature updates(Q: why not suitable for this?)

- **2. Layered (n-Tier) Architecture**
- **Purpose:** Enforces separation of concerns (UI, business logic, data).
- Core banking platforms separate UI, business rules, and data persistence. This layered design ensures strict regulatory compliance, testability, and maintainability across millions of transactions.(Q: What's especial about banking platforms that we need to separate data from business ?

- **3. Microservices Architecture**
- **Purpose:** Split complex systems into independently deployable services.
- Best for: Large-scale systems needing high scalability, agility, and independent team ownership (Netflix, Amazon Q: who else?).

- **4. Event-Driven Architecture**
- Purpose: Trigger actions asynchronously via events (publish/subscribe model).
- Best for?
- Q: Example?

- **5. Serverless / Function-as-a-Service (FaaS)**
- **Purpose:** Run small, stateless functions in the cloud without managing servers.
- **Best for**: Apps with unpredictable traffic (the server is developed by someone else and made available to you via API calles) tasks like image processing throw sending API request to an image server, vending machine payments
- **Example**: apps you developed in term one where you had no server side scripting involved

- **6. Model-View-Controller (MVC)**
- **Purpose:** Organize code into UI (View), logic (Controller), and data (Model).
- **Best for**: Web frameworks (Django, Rails, ASP.NET MVC) — good when app logic is closely tied to views (Q: what does this mean?).
  GitHub was originally built on **Ruby on Rails (MVC)**.(true?)

- **7. Model-View-ViewModel (MVVM)**
- **Purpose:** Improve UI separation, enable two-way data binding.
- Best for: Rich client-side apps with reactive UIs (Angular, Vue, React with state management).
- Real-Life Example?

- **8. Micro-Frontends**
- **Purpose:** Break large front-end apps into smaller, independently deployable parts.
- Best for: Enterprises with multiple teams contributing to a single large front-end (e-commerce sites, dashboards).
- **Real-Life Example: Spotify** uses micro-frontends to scale different parts of its web player.(true?)

- **9. Hexagonal Architecture (Ports & Adapters)**
- **Purpose:** Decouple core business logic from external systems (DBs, UI, frameworks).
- Example: Lufthansa Systems (aviation software)
- They use hexagonal architecture to decouple flight scheduling logic from databases, UI, or external APIs. This makes systems testable, reliable, and adaptable to new data sources.
- **10. CQRS (Command Query Responsibility Segregation)**
- **Purpose:** Optimize performance by separating write (commands) from read (queries).
- LinkedIn ? uses CQRS in its feed and messaging systems. Write-heavy actions (posting updates, sending messages) are handled separately from read-heavy queries (loading millions of feeds), ensuring performance at scale.

Can we utilize multiple architectural pattern into one products ?1/3

- Most big companies **don’t stick to just one architecture** — they evolve over time and mix patterns depending on the system’s needs. (examples from ChatGPT – to verify)
- **Netflix**
- **Started:** Monolithic (early DVD rental days).
- **Now:** Microservices for core streaming (recommendations, billing, video encoding).
- **Also uses:** Event-driven patterns for streaming telemetry, CQRS for separating user actions (e.g., play, pause) from analytics queries.
- Netflix shows how companies **don’t stick to one architecture**. They evolve:
  Monolith → Layered → Microservices → Event-driven + CQRS as scale and complexity increase.

- **Spotify**
- **Front-end:** Micro-frontends (different teams ship UI parts independently).
- **Back-end:** Microservices (recommendations, search, music catalog).
- **Also uses:** Event-driven patterns for real-time activity feeds (“Your friend is listening to…”).

Can we utilize multiple architectural pattern into one products ? 2/3

- **Alibaba**
- **Front-end:** MVVM (Vue.js for rich, reactive UIs).
- **Back-end:** Microservices to support large-scale e-commerce operations.
- **Also uses:** Serverless (Alibaba Cloud's Function Compute) for handling burst traffic on events like Singles' Day sales.
- **LinkedIn**
- **Core:** CQRS for feeds and messaging (separating reads/writes).
- **Also uses:** Event-driven architecture for notifications, microservices for modular scaling, and hexagonal architecture patterns in some domain-driven designs.

Can we utilize multiple architectural pattern into one products ? 3/3

- **Coca-Cola (IoT / vending machines)**
- **Uses:** Serverless (AWS Lambda) for vending transactions.
- **Also uses:** Event-driven workflows for telemetry and inventory reporting, plus microservices for backend processing.

- **Lufthansa Systems**
- **Core:** Hexagonal architecture (domain-driven flight scheduling). Q: Why Hexa?
- **Also uses:** Layered architecture for ERP-like internal tools, event-driven systems for real-time flight updates.
- **Uber**
- **Started:** Monolithic Ruby on Rails app.
- **Now:** Migrated to Microservices to scale different domains (maps, payments, ride matching).
- **Also uses:** Event-driven architecture for real-time ride dispatch and pricing updates.

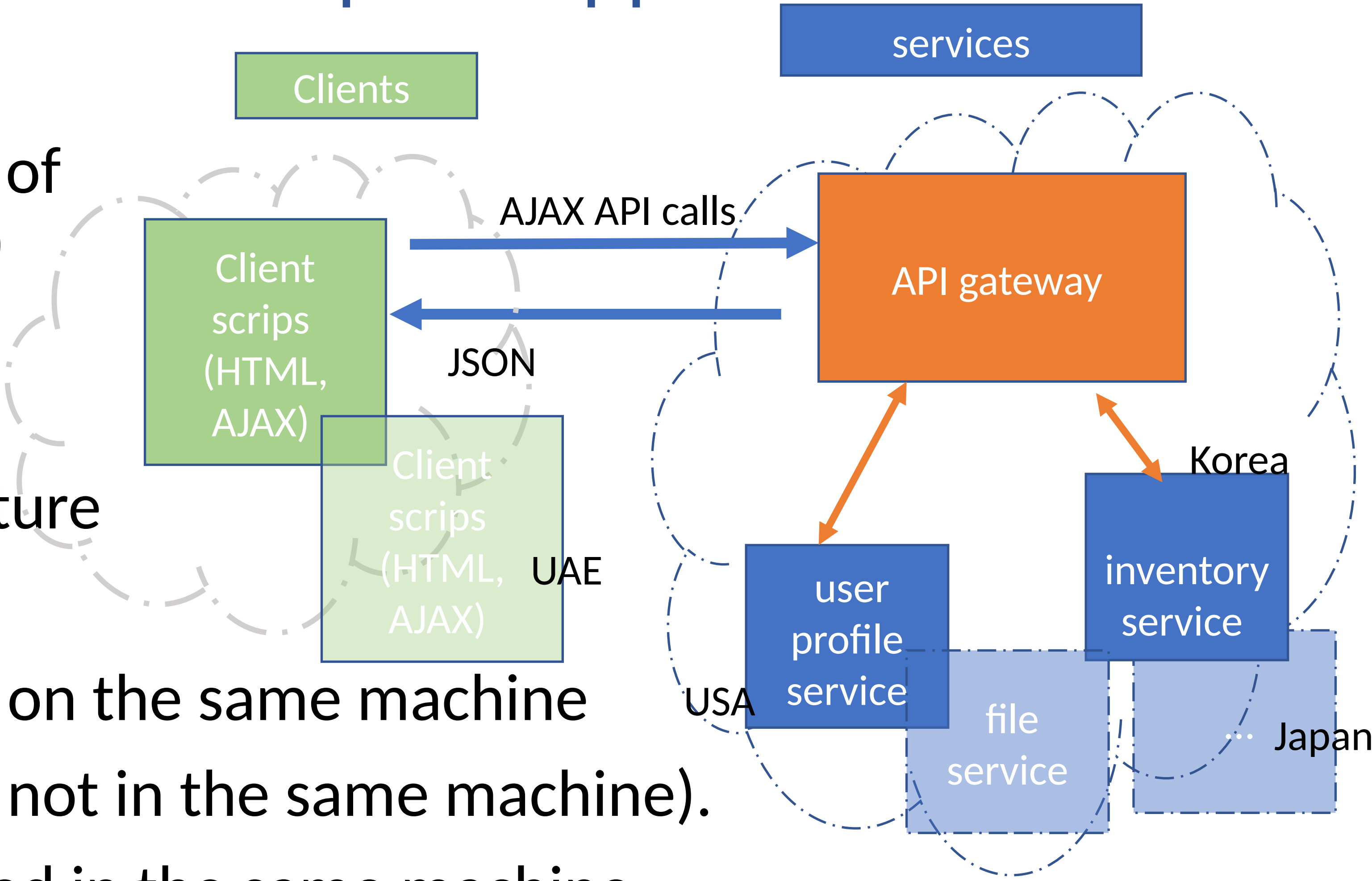
3

What we study in this course

An API-Server with service based architecture

which is also RESTful
Q: it fully rests ?

Web architecture we use to develop our app in this course



We practice a simplified version of Microservice architecture (MSA)

We design an API centric MSA

API:
API calls follow **RESTful** architecture

Different origin:

Clint and servers are not hosted on the same machine

(HTML files and nodejs files are not in the same machine).

Even services might not be hosted in the same machine.

Response:

Responses coming back from services are often in **JSON** format

Learning outcome we seek from developing API centric architecture

- Design and develop programs using REST, JSON and HTTP.
  - Learning the RESTful API architecture
  - Working with **HTTP methods** (GET, POST, PUT etc)
  - Working with **JSON** as a format to pass data back and forth
- Design a simple web service.
  - A simple web service that returns result to AJAX requests like mydomain.com/API/Patients/1
- Design and develop a RESTful API by applying the best practices and REST constraints.
  - You practice this in one of your labs and in your term project
- Document and write specifications.
  - using swagger

- Practice implementation of CRUD operations.
  - Review of RDBM
  - Doing **C**reate, **R**ead, **U**pdate and **D**elete operations on a database via API requests
- Implement error handling, HTTP status codes, secure API.
  - 404, 303, 200, 505, …with messages such as
  - Bad request
  - Unauthorized
  - Not found
- Apply change management and versioning of designed APIs.
  - mydomain.com/**V1**/API/Patients/1
  - mydomain.com/**V1.1**/API/Patients/1
- Secure a backend API server.
  - simple token based approach
  - Hashing using promises

- Also
  - Working with web storage
  - Private key/ public key
  - Asynchronous programming
  - Practicing denial of service attack
  - How to design scalable applications
  - How oAuth works
  - You may need to install SSL (depends on your host provider how to do it)

Lets get started ….

4

JSON

and

HTML Web Storage

Web Storage (inside your browser, not server)

- We want to store data locally within the user's browser.
- Before HTML5, application data had to be stored in cookies.
- Web storage is more secure, and large amounts of data can be stored locally (generally a couple of mega bytes, 5MB or up) .
- Note that web storage is per origin! That means per domain and protocol.
- All pages, from same origin, can store and access the same data **within same browser** on your computer of course.
- It is a client thing (in your browser's local data)
- Web storage is client-side storage, which means it resides in the local data storage of your browser

localStorage vs sessionStorage

- There are two HTML web storage objects for storing data on the client:

- window.localStorage - stores data with **no expiration** date
- window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)

- Since it is a new thing, check browser support for localStorage and sessionStorage before using web storage, :

-
  ```
  if (typeof(Storage) !== "undefined") {
    // Code for sessionStorage/ localStorage.
  } else {
    // Web Storage is not supported in this browser..
  }
  ```

The localStorage Object

- no expiration date.
- The data will not be deleted when the browser is closed
- will be available the next day, week, or year.
- Example

-
  ```
  // Store
  localStorage.setItem("myKey", "Monday and Friday");
  ```

-
  ```
  // Retrieve
  localStorage.getItem("myKey");
  ```
- **Note:** Name/value pairs are always stored as strings. Remember to convert them to another format when needed!

Example

- You write stuff in a browser tab (open writer.html)
- reader.html reads the same stuff written(stored) by writer.html next day or same time in different tab

The localStorage Object Example/ writer.html

This code snippet writes something onto local storage

```
<html lang="en">
<body>
    <script>
        const msg_notSupported = "Sorry web Storage is not supported!";
        const msg_key = "hidden secret";
        const msg_written="A piece of data was written in local storage for the key:"
;
        if (typeof (Storage) == "undefined") {

            document.write(msg_notSupported);
            window.stop();
        }
        localStorage.setItem(msg_key, "2021");
        document.write(msg_written+msg_key);
    </script>
</body>
```

The localStorage Object Example/ reader.html

This code snippet tries to read from local storage (remember same domain, same browser, same machine)

```
<html lang="en">
<body>
    <script>
        const msg_notSupported = "Sorry web Storage is not supported!";
        const msg_key = "hidden secret";
        const msg_read="stored data for the key ";
        if (typeof (Storage) == "undefined") {

            document.write(msg_notSupported);
            window.stop();
        }
        document.write(msg_read+msg_key+":"+localStorage.getItem(msg_key));
    </script>
</body>
</html>
```

Question!

How can I store the entire object into the local storage ?

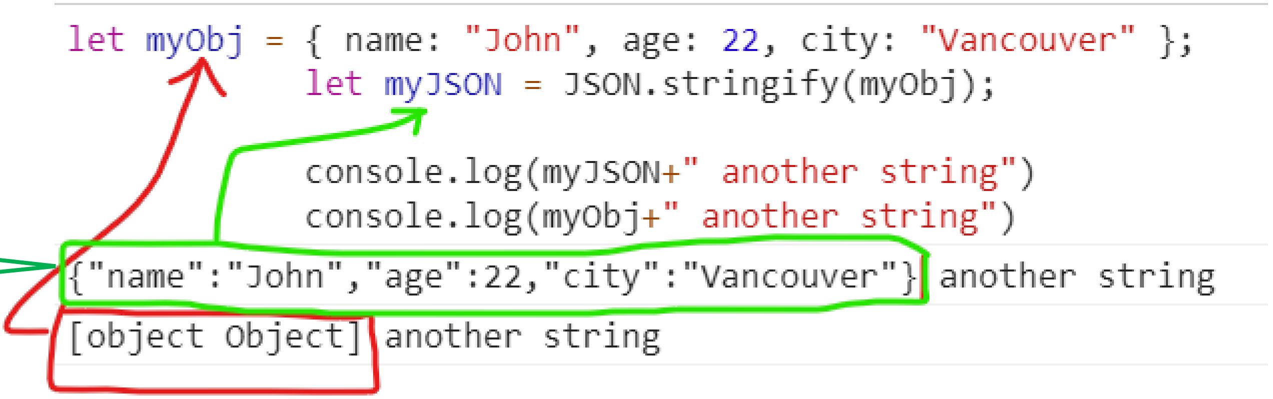
JSON

JavaScript Object Notation

JSON or JavaScript Object Notation

1. You can flatten a JavaScript object into a JSON string and send it to another computer over the net.

```
let myObj = { name: "John", age: 22, city: "Vancouver" };
let myJSON = JSON.stringify(myObj);
console.log(myJSON + " another string");
console.log(myObj + " another string");
```



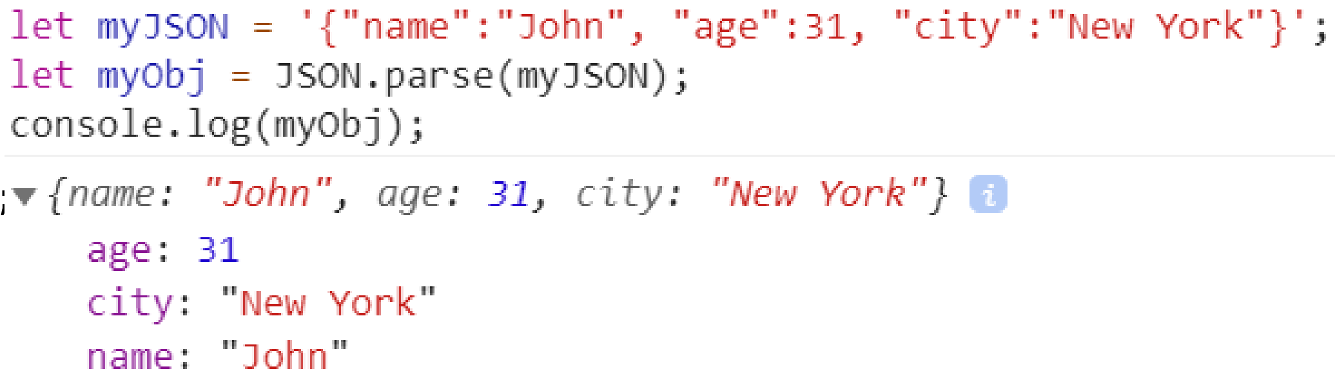
```
let myObj = { name: "John", age: 22, city: "Vancouver" };
    let myJSON = JSON.stringify(myObj);

    console.log(myJSON+" another string")
    console.log(myObj+" another string")
{"name":"John","age":22,"city":"Vancouver"} another string
[object Object] another string
```

As you have noticed, json behaves like a flat string See the result of concatenating it with another string .

- **Q:** What is the type of myJSON here?

2. At the other end the json string can be parsed back to an object:

- `let myJSON = '{"name":"John", "age":31, "city":"New York"}';`
- `let myObj = JSON.parse(myJSON);`
- `console.log(myObj);`



```
let myJSON = '{"name":"John", "age":31, "city":"New York"}';
let myObj = JSON.parse(myJSON);
console.log(myObj);
▼{name: "John", age: 31, city: "New York"}
    age: 31
    city: "New York"
    name: "John"
```

5

More programming Tips

Using same event handler for multiple buttons! (based on JavaScript closure)

Passing parameters to Event handlers

- Remember we said if function name is followed with brackets, it gets executed immediately e.g. myFunc().
- A function can return a number, a string , a Boolean **or even a function**!!!
- Yes a function in JS can return a function!!!!
- Based in these two here is an example where parameters were passed to same function handling events for two buttons:

```
<input type="button" value="Hi" id="button1">
<input type="button" value="Hi" id="button2">
<script>
    function handler(a) {
        return function () {
            console.log('Hi ' + a);
        }
    }
    document.getElementById("button1").onclick = handler(1);
    document.getElementById("button2").onclick = handler(2);
</script>
```

This is based on a feature of JavaScript called ***closure***.

Variable a is assessable in the function returned by the enclosing function

Object constructor with methods (another example was given in lecture 1)

Example: Object constructor with methods (this is old way, you can now use classes in JS)

Q: What does this code snippet do?

```
let arrayButtons = [];
function Button(color, width, height, top, left, order) {
    this.order = order;
    this.btn = document.createElement("button");
    this.btn.style.backgroundColor = color;
    this.btn.style.width = width;
    this.btn.style.height = height;
    this.btn.style.position = "absolute";
    document.body.appendChild(this.btn);
    // A method to set location
    this.setLocation = function (top, left) {
        this.btn.style.top = top;
        this.btn.style.left = left;
    };
    /* we call this method to set original
    top,left during creation of the object*/
    this.setLocation(top, left);

}
arrayButtons.push(new Button("Red", "100px", "100px", "0px", "0px", 0));
arrayButtons.push(new Button("Blue", "200px", "100px", "200px", "200px", 1));
// this has to be in a separate function, e.g. moveRandom ... or something
setInterval(function () {
    arrayButtons[0].setLocation(
        Math.floor(Math.random() * 100) + "px",
        Math.floor(Math.random() * 100) + "px");
}, 500);
```

References

- https://gearheart.io/articles/how-build-scalable-web-applications/
- https://www.cleveroad.com/blog/web-application-architecture
- https://stackify.com/web-application-architecture/
- https://en.wikipedia.org/wiki/Software_architecture
- https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/
- https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html
- https://www.sciencedirect.com/topics/computer-science/presentation-logic
- https://lanars.com/blog/web-application-architecture-101
- ChatGPT was used for proofreading and content generation too