Simple to Complex Frontend Applications (pdf)

Last Updated: 01.04.2023

Introduction

This article will analyze react projects from simple to complex according to **data access**.

The need for data, the way the data is retrieved, the way it is kept, its properties, and many similar issues cause the concept and library in your project to change.

Therefore, if we can estimate the difficulty level of our project before starting the project, this will allow us to make better decisions about the following concepts;

- Rendering
- Networking
- State Management
- Routing
- Bundling

- Product Management, UIX Design → Architecture Interaction
- Git Repo Structure, Branching Types
- DevOps
- Security
- Internalization, Localization
- Accessibility

Let's examine the structures I call simple to complex. We can make this grouping as follows;

- **Simple** - No API Usage (Static Rendering)
- **Simple** - No API Usage (Client Side Rendering)
- **A Little Harder Than Simple** - Public API Usage But Has Not Authentication
- **Intermediate Level** - Private API Usage Authentication
- **Hard** - Private API Usage Authentication and Authorization (Resource Access according to Organization, Roles)
- **Complex** - Private API Usage Authentication, Authorization, and Service or Product Sell Flow (Offer, Order, Customer Success, Payment, Invoice)

Note: If the simplicity I am talking about here is not simplicity due to access and interaction with Data, we do not include complexities such as Animation, Visual Effects, etc., in your project.

A. Simple Level

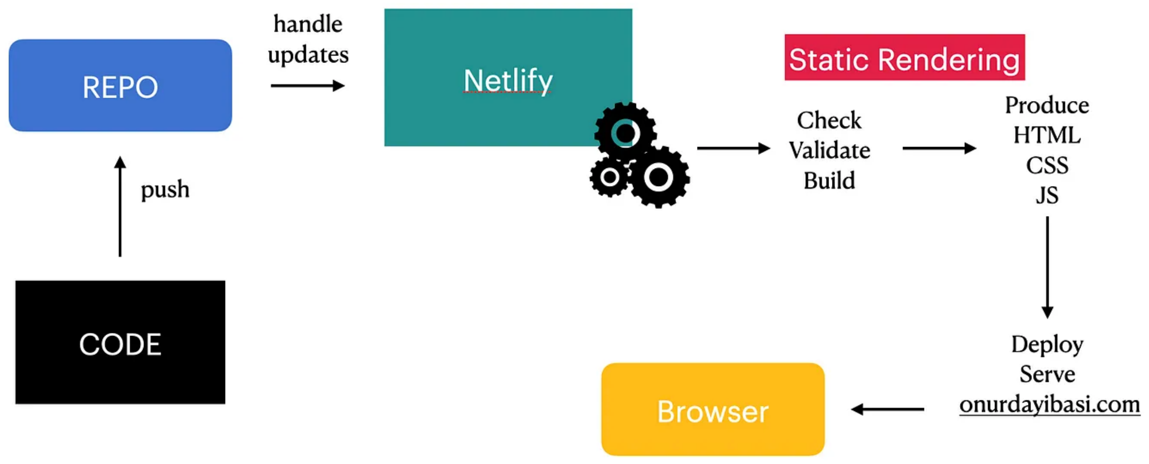
(Level - 1) Static Rendering

Such projects are usually blogs, news, and prose projects that do not have much user interaction on the page. An important issue is that their content is not kept in the database but directly in Markdown files or similar XML derivative files in GitHub and converted to HTML files during the build.

Such examples do **not interact with the server***; ***the data is already defined*** as Frontend projects.

To give an example from my project, OnurDayibasi.com is a site like this. This site is built on **Gatsby,** and all the data is in the project. This data is served via **Netlify** with a **static rendering** method. So it uses data from any external source or database.

When we update the code in the operation part of the application and push it to the Repository (GitHub), the plugins connected to the project on the Netlify platform capture these changes via **hooks**, pull the relevant version of your project from the Repository on GitHub, convert it to HTML, CSS, and JS with the **static rendering** method, and deploy it to the relevant place to be served via DNS.



onurdayibasi.com DevOps Cycle

The result is a static HTML, CSS-intensive website.

Most blogs, personal websites, and documentation help sites are built this way.

Previously, such sites were developed with WordPress, Joomla, Drupal, and PHPForum tools and deployed to BlueHost-style sites. This structure, which we call **LAMP** Stack, was developed as Linux, Apache (WebServer), PHP (Programming Language), and MySQL (DB).

Of course, **MEAN** Stack also came out (Mongo, Express, Angular, and Node).

A simpler and cheaper stack emerged for simple blogs, personal websites, and documentation (**JAM** Stack). With Javascript, API, and Markdown, you can view your website on cloud service providers such as **Vercel and Netlify** through some Frameworks and Scripts, and you can see your work without paying money.

These sites are Content-heavy and are structures where ClientSide re-rendering and interaction with JS inside the site is not too much, and the Content is kept in the

GitHub project. Developers can write content in HTML or in structures such as MD (Markdown) →, MDX (MarkDown for Component), → JSX instead of HTML for faster creation.

Note: Mostly, Next.js, Gatsby, and other JamStack frameworks are used in the infrastructure of the projects used in this section.

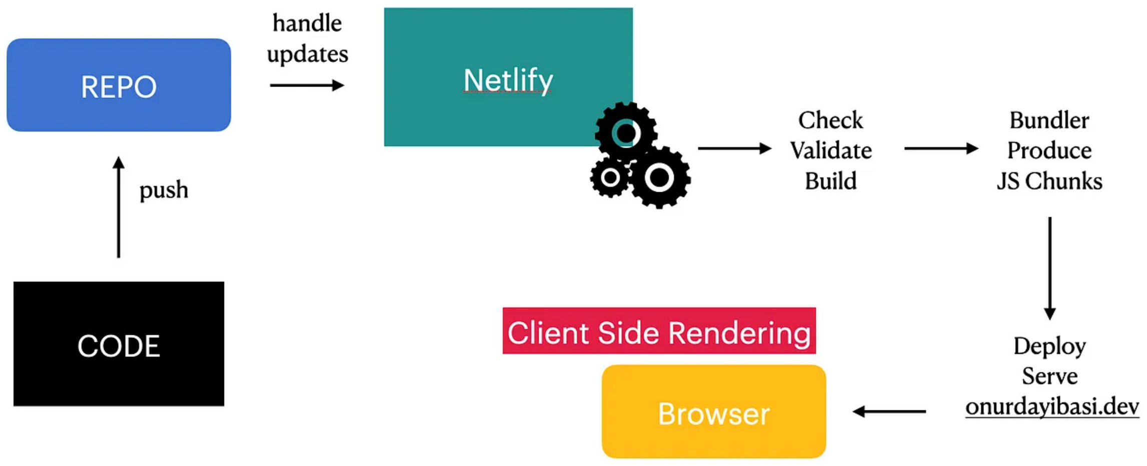
(Level - 2) Client Side Rendering

Examples of such projects are structures that do not interact with the server and keep the data in itself, but JS-intensive pages render on the browser and product pages.

Again, to give an example from my projects, the digital garden I created on onurdayibasi.dev is a site like this. This site, built on CRA, has all the data in the project.

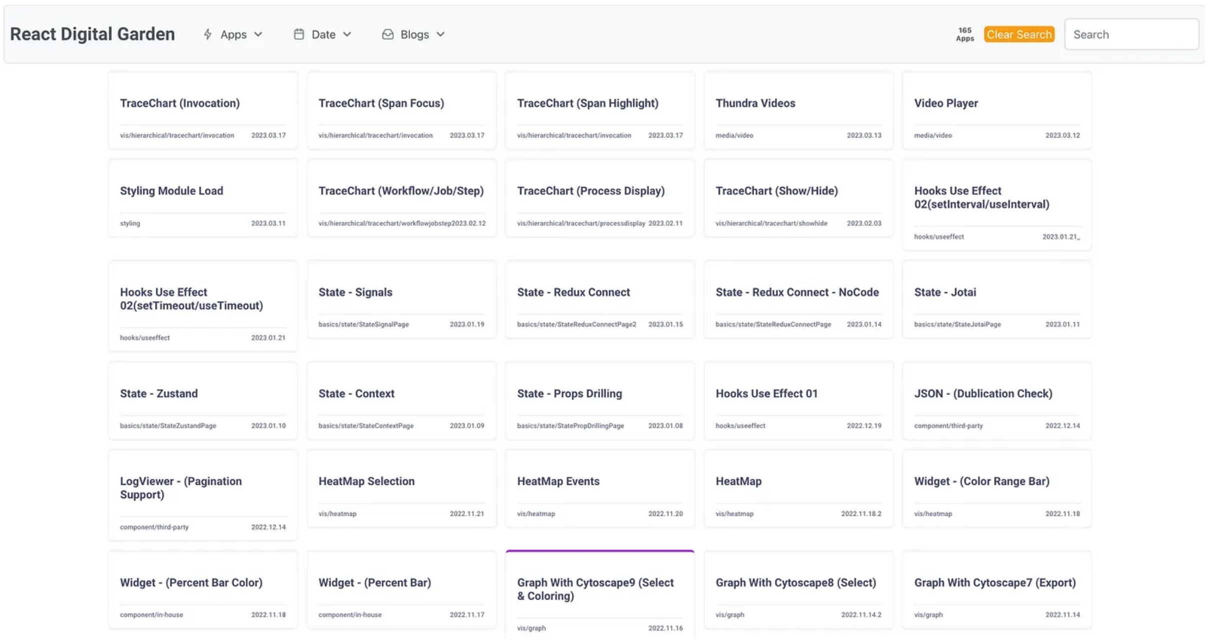
This data is created by running the JS files served via the browser's **client-side rendering** method via **Netlify**. In other words, it does not use data from any external source or database.

There is not much change in the DevOps part of the structure you see below compared to the example we gave in the 1st example. The Rendering concept is now executed on the Browser (Client Side Rendering), and pages are created with the Client Side Rendering method. Bundlers (Webpack, Vite, ESM, etc...) divide our project into parts according to Routing (Suspense, import) and allow libraries to be downloaded individually.



onurdayibasi.com DevOps Cycle

This results in an application with a lot of UI interaction but is quite simple regarding data usage.



TraceChart (Invocation)	vis/hierarchical/tracechart/invocation	2023.03.17
TraceChart (Span Focus)	vis/hierarchical/tracechart/invocation	2023.03.17
TraceChart (Span Highlight)	vis/hierarchical/tracechart/invocation	2023.03.17
Thundra Videos	media/video	2023.03.13
Video Player	media/video	2023.03.12
Styling Module Load	styling	2023.03.11
TraceChart (Workflow/Job/Step)	vis/hierarchical/tracechart/workflowjobstep	2023.02.12
TraceChart (Process Display)	vis/hierarchical/tracechart/processdisplay	2023.02.11
TraceChart (Show/Hide)	vis/hierarchical/tracechart/showhide	2023.02.03
Hooks Use Effect 02(setInterval/useInterval)	hooks/useeffect	2023.01.21_
Hooks Use Effect 02(setTimeout/useTimeout)	hooks/useeffect	2023.01.21
State - Signals	basics/state/StateSignalPage	2023.01.19
State - Redux Connect	basics/state/StateReduxConnectPage2	2023.01.15
State - Redux Connect - NoCode	basics/state/StateReduxConnectPage	2023.01.14
State - Jotai	basics/state/StateJotaiPage	2023.01.11
State - Zustand	basics/state/StateZustandPage	2023.01.10
State - Context	basics/state/StateContextPage	2023.01.09
State - Props Drilling	basics/state/StatePropDrillingPage	2023.01.08
Hooks Use Effect 01	hooks/useeffect	2022.12.19
JSON - (Dublication Check)	component/third-party	2022.12.14
LogViewer - (Pagination Support)	component/third-party	2022.12.14
HeatMap Selection	vis/heatmap	2022.11.21
HeatMap Events	vis/heatmap	2022.11.20
HeatMap	vis/heatmap	2022.11.18.2
Widget - (Color Range Bar)	vis/heatmap	2022.11.18
Widget - (Percent Bar Color)	component/in-house	2022.11.18
Widget - (Percent Bar)	component/in-house	2022.11.17
Graph With Cytoscape9 (Select & Coloring)	vis/graph	2022.11.16
Graph With Cytoscape8 (Select)	vis/graph	2022.11.14.2
Graph With Cytoscape7 (Export)	vis/graph	2022.11.14

onurdayibasi.com

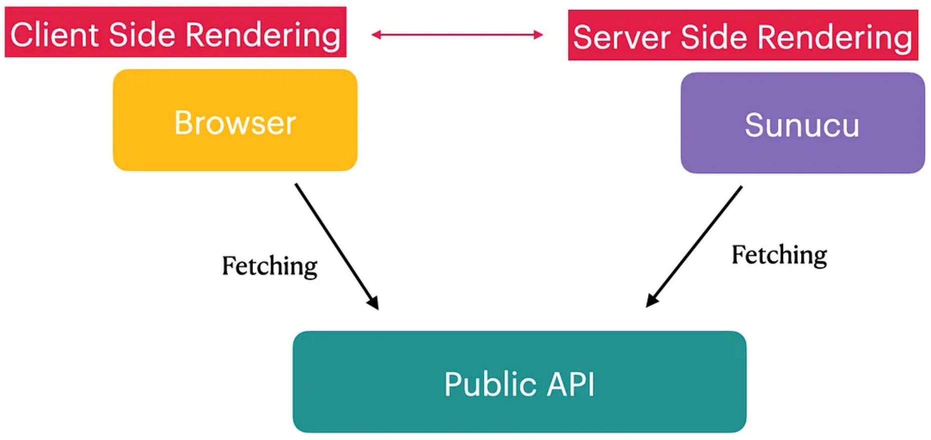
When you look at it, there are about 165 small applications inside, many UI libraries, and many examples. So even though our application is very complex in terms of UI (Chart, Table, Dashboard, Layout, Styling, Events, Drag-Drop, and Code Editor), it is a very weak project regarding data interaction.

(Level - 3) API Usage with CSR or SSR

Such projects are publicly accessible projects that are developed using public APIs that we see in homework assignments or clone project examples that we do to improve ourselves.

There is access to data here, but usually, only viewing is performed on this data without updating because data entry is not requested on sites that are not authenticated.

In some cases, changes to the data may be allowed, such as a captcha or in developing some UI tools with limits. But these are very limited because the second cost is a security problem.



Public Web Site

In these structures, you can access the data and make requests to the extent the Public API allows. However, it is a more difficult project approach than the two examples above. This is because your project comes with the following extra work.

- Server State Sync operations (React Data)
- State Management Concepts (React Data)
- Fetching (React Networking)

Let's list the most frequently mentioned sample projects and APIs below;

- How to Build a Hacker News Clone Using React (API)
- Netflix Clone
- Medium Clone
- Wheater App (API)
- Coin Exchange (API)

As I said, clone and sample projects of this type can be used over open APIs that are frequently available in the market, generally used for display, i.e., listing and visualization.

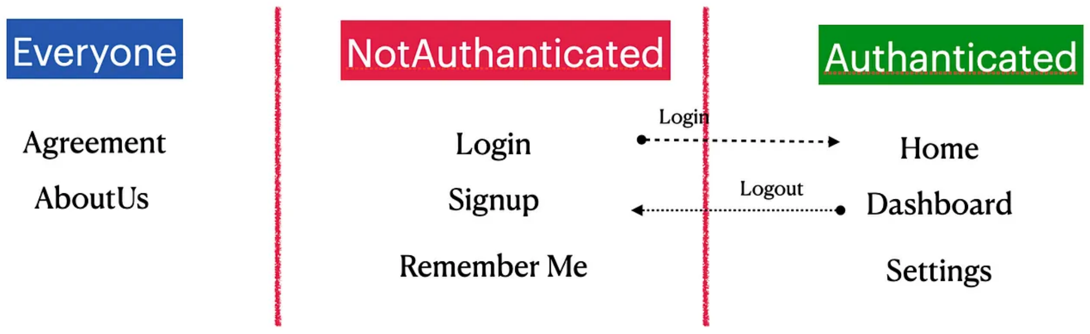
B. Intermediate Level

(Level - 4) Private API Usage - Authentication

Things start to get complicated when it comes to identification. When identity comes into play, the data type is divided into three parts;

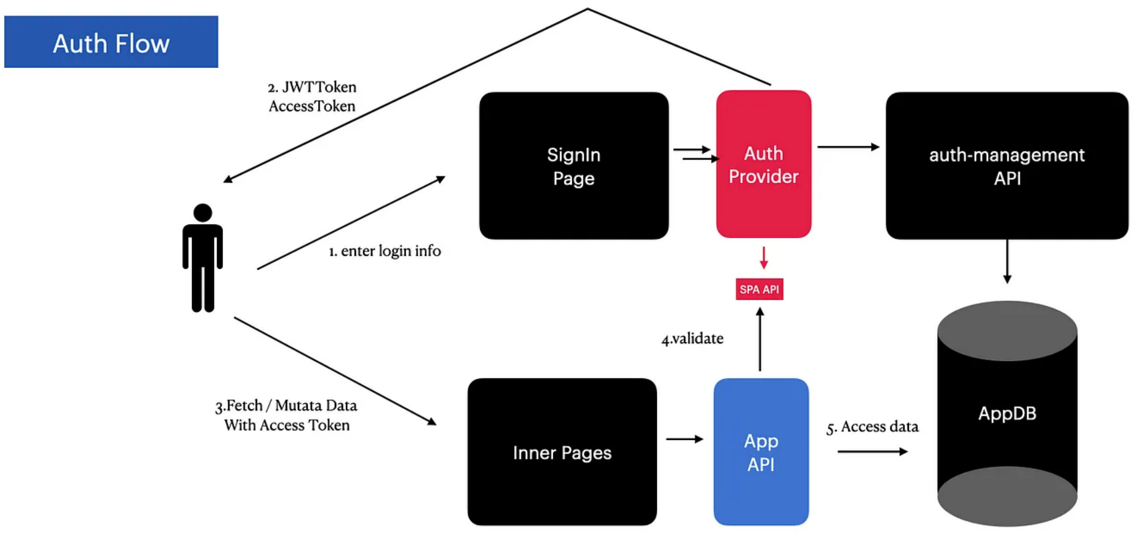
- Only the data that can only be seen by people who are logged into the system and whose identity has been determined
- Only unidentified persons will be able to see
- Data that is not important to log in to the system, i.e., data that can be viewed in both cases

In this case, we need three **types of pages**.



- First, these pages are publicly visible and are not related to authentication. You can access these pages whether you are authenticated or not. **(Agreement, AboutUs, TermsOfService)**
- Secondly, pages that you can only see when you log in. Pages such as **Home, Dashboard, or Settings** that are filled with your data only for you, screens where only your information can be changed.
- Thirdly, pages like **Login, Signup, and Remember Me** that you can see when you Logout.

The situation I mentioned above actually involves **Routing and Redirection**. You also need to set up the Authentication Mechanism in the background. This Authentication flow has different types. Let's talk about the most commonly used JWTToken method today;



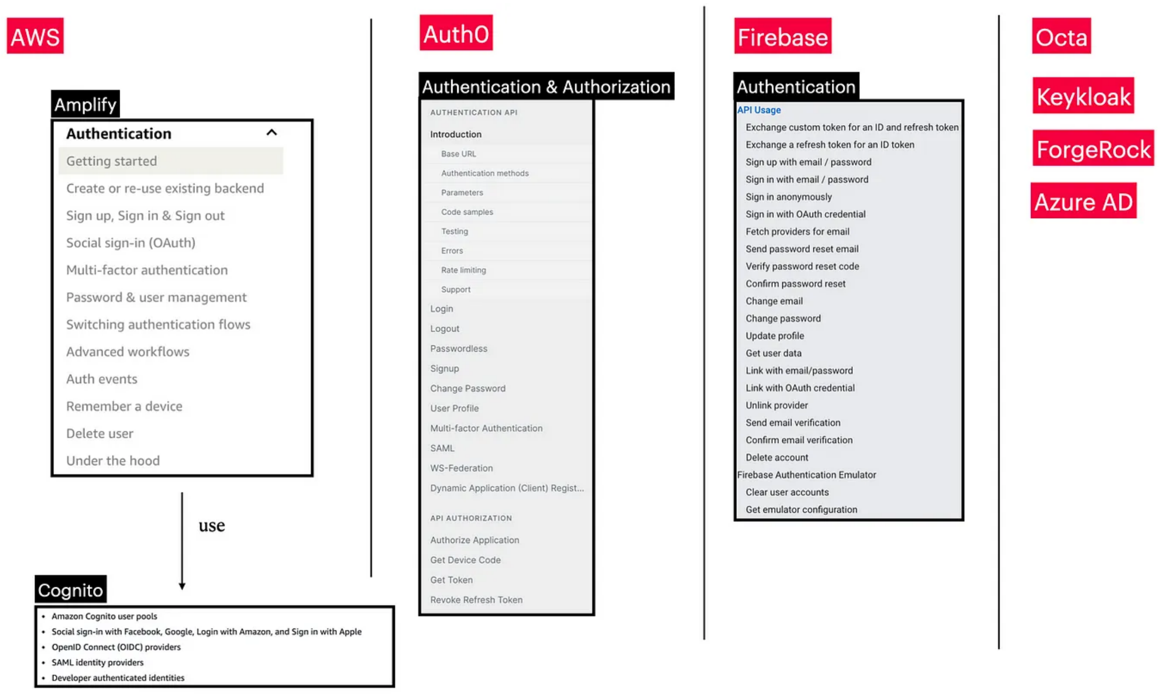
Auth Flow

In the picture above;

- In the **1st step,** the user enters username and password-like information from the SignIn page.
- In step 2** If AuthProvider confirms this, it sends the user an AccessToken to their browser and redirects the user to the internal pages of the app.
- In step 3, the user accesses the App API to retrieve data with this access token.
- The App API validates the Access Token AuthProvider in the incoming request. It retrieves the user information and, with this, pulls the information from the DB and returns it, or the system renders the page on the server side.

Developing the parts in the Auth Provider section **in-house** can be difficult in this section. You need to master many security issues.

There are many ready-made services and libraries in this section. If we list the services and their capabilities below;


AWS
Amplify
Authentication
Getting started
Create or re-use existing backend
Sign up, Sign in & Sign out
Social sign-in (OAuth)
Multi-factor authentication
Password & user management
Switching authentication flows
Advanced workflows
Auth events
Remember a device
Delete user
Under the hood
use
Cognito
Amazon Cognito user pools
Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple
OpenID Connect (OIDC) providers
SAML identity providers
Developer authenticated identities
Auth0
Authentication & Authorization
AUTHENTICATION API
Introduction
Base URL
Authentication methods
Parameters
Code samples
Testing
Errors
Rate limiting
Support
Login
Logout
Passwordless
Signup
Change Password
User Profile
Multi-factor Authentication
SAML
WS-Federation
Dynamic Application (Client) Regist...
API AUTHORIZATION
Authorize Application
Get Device Code
Get Token
Revoke Refresh Token
Firebase
Authentication
API Usage
Exchange custom token for an ID and refresh token
Exchange a refresh token for an ID token
Sign up with email / password
Sign in with email / password
Sign in anonymously
Sign in with OAuth credential
Fetch providers for email
Send password reset email
Verify password reset code
Confirm password reset
Change email
Change password
Update profile
Get user data
Link with email/password
Link with OAuth credential
Unlink provider
Send email verification
Confirm email verification
Delete account
Firebase Authentication Emulator
Clear user accounts
Get emulator configuration
Octa
Keykloak
ForgeRock
Azure AD

Authentication Services

AWS

- Cognito API Reference
- Amplify API Reference

Auth

- https://auth0.com/

Firebase

- https://firebase.google.com/docs/auth

Other

- Octa
- Keykloak
- ForgeRock
- AzureAD

❓ Ok, with this authorization, we have defined the user ID and then performed the data access works. Can we get additional jobs besides this?

✅ Yes. Additional requests may come according to the needs of the project.

- Analytics Needs
- Monitoring Needs
- Communication with the User (Chat, Message)

Analytical Needs

You may need to include some of the following tools in your product to analyze user audiences and behaviors. For user-based tracking needs on these tools, you may need to make their breakdowns user-based.

- Google Analytics Integration
- Clevertap
- FullStory
- Heap
- Amplitude
- MixPanel
- Countly

Monitoring Needs

Like in Analytics, you may want to measure the system's performance when a user has a problem; even if the user does not communicate this problem, you may want to capture error information, repeat its use, and detect the error.

Passing credentials to such monitoring tools ensure that connections are established to communicate with the customer when necessary.

- Sentry
- DataDog
- Replay io
- NewRelic RUM
- SemaText
- Atatus

- Amazon RUM

Communication and Interaction with the User

Since the user wants to enter the system and communicate with you, you need to integrate with some extra tools, create live chat and message forwarding structures and enable them to communicate with you. There are some ready-made services in this regard;

- Intercom
- Hubspot

User Settings

When the user is defined, it is necessary to make some personal adjustments and make extra screens for these to be reflected on the screens. For example

- Upload avatar
- User information and password update
- Screen Theme, Colors, Font Size, etc...

Difficult Level

(Level - 5) Private API Usage, Authentication, Authorization

In such projects, **Authentication** is a side feature, Organization, and Roles, and according to these Roles, it is necessary to develop the access authorizations of the Resources, i.e., screen codes. Why do the **Organization and Role concepts** I mentioned above make things difficult?

First, examine what **Organization** means and how it changes the application structure. Let's remember Conway's law.

Is Organization Structure Important for Application Architecture?

Melvin E. Conway observed in 1967 that organizations and businesses are developing software that reflects their communication systems.

This led to the following **Conway's Law.**

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.[3]- Melvin E. Conway

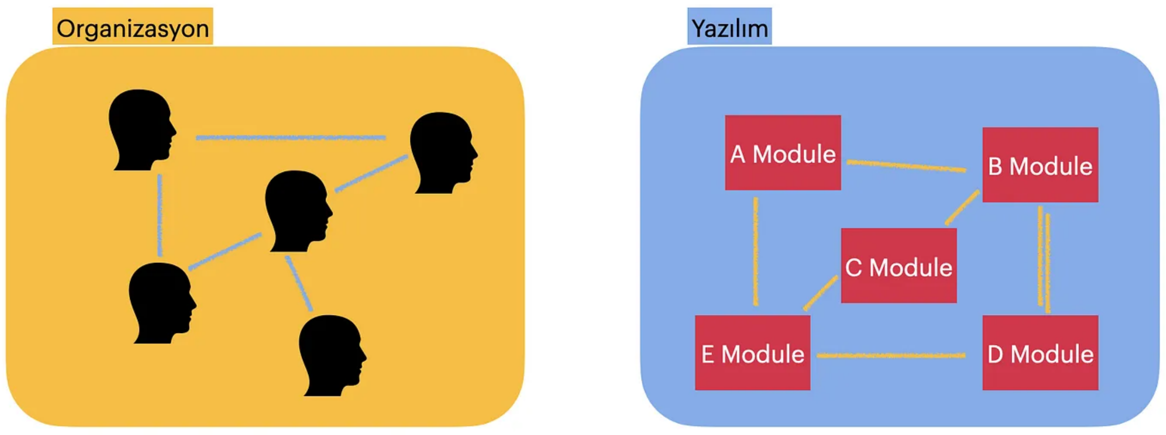
This law is **bas**ed on the idea that a structure similar to the relationship between software modules exists and resembles the structure of an organization. Software interfaces in the system structure work and communicate similarly to organizations' social boundaries.
Conway put this law forward through sociological observation. 1968 National Symposium
this came to be known as the Conway Act.

Why is this topic so important?

For a long time, I have worked in software development teams for the Military, Telecom, Government Institutions, Bank/Finance Companies, etc. I have seen that more successful software is produced when team structures are designed according to this organizational structure or when intertwined with these units.

There are two reasons underlying this success.

- First, the development teams are naturally distributed within the organizational structure, so modules and module interfaces are created with the naturalness of Conway's law.
- Secondly, when the development team works with the organization's domain experts, they can analyze the requirements **better** and identify the needs on the spot.



Conway Yasası (Organizasyon ve Yazılım arasındaki parallellik)

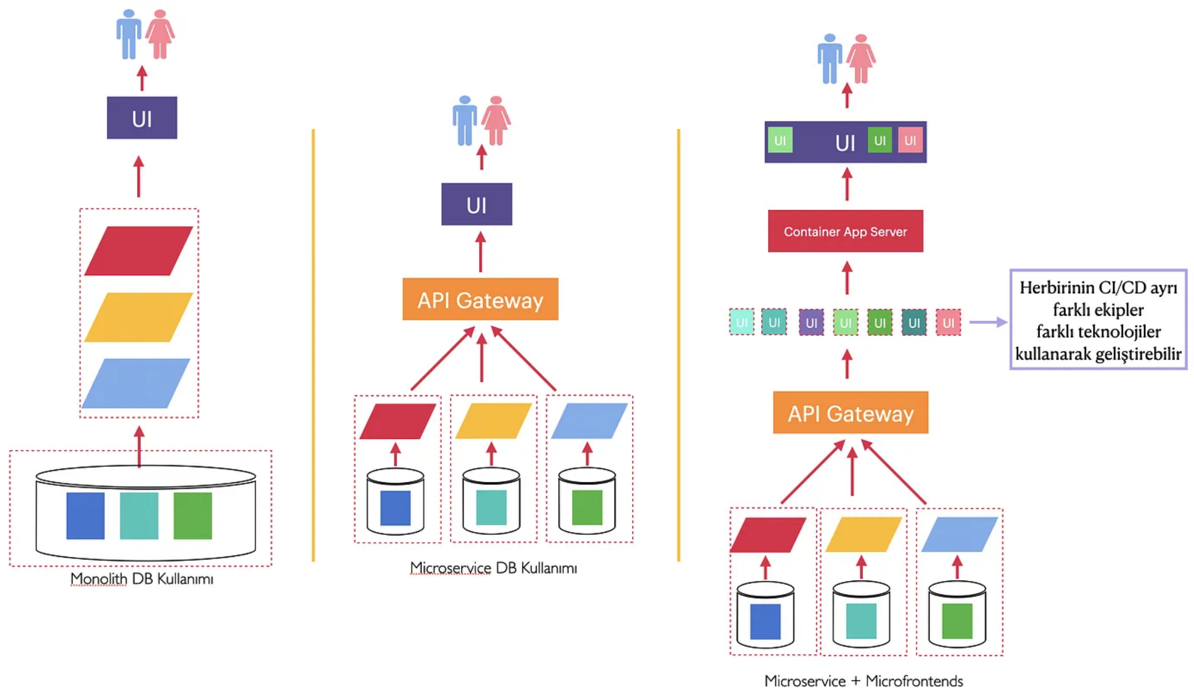
Note
What I mean by successful here is compatibility with the customer's requests. Because the dependency on the organization you are doing business with will gradually increase, a product will emerge with the features the tailor makes for one person. In contrast, this product will be very suitable for that person/organization; it will not be easily adaptable to other customers or organizations.

Are the New Architectural Approaches Related to the Organizational Structure (Micro-service and Micro-Frontends)?

Yes, there is. In the transition from a Monolith structure to a Microservice structure or even Microfrontend structure, the modularity in the software directly affects the teams and the communication between these teams.

Since the development teams can be matched with micro-level software, more compatible and successful software emerges. For example, in this way, the following partitioning can be done much more simpler.

- Invoice Services,
- Advertising and Campaign Services
- Product Service
- Sales Service
- Customer Service
- CRM etc...


UI
Monolith DB Kullanımı
UI
API Gateway
Microservice DB Kullanımı
UI
UI
UI
UI
Container App Server
UI
UI
UI
UI
UI
UI
UI
Herbirinin CI/CD ayrı farklı ekipler farklı teknolojiler kullanarak geliştirebilir
API Gateway
Microservice + Microfrontends

Note:
In this regard, you can see from the picture above that software development methods and tools are increasingly turning into micro-structures following Conway's law.

Let's now examine the impact of **Access via Role and Resources** authorization on applications.

Authorization Models

In an application, there are different models to manage and operate the authorizations of the operations that can be done and seen after authentication during access through the user interface or API, for example, the **Attribute-Based Access Model** or **Role-Based Access Models.**

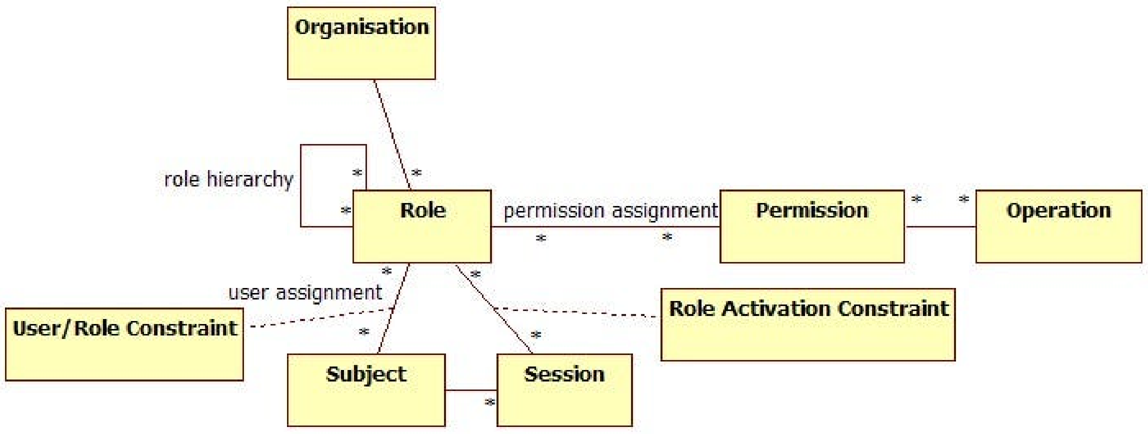
After identifying users or Agent Systems, you need to authorize the operations they can perform on certain resources.

- *For example, the operations that a User (User) will do and the operations that an Administrator (Admin) will do will differ.

At this stage, there are two types of authorization models. The first one is

Role-Based Access Models

Role-based access model is an authorization model where authorizations are kept on roles.



https://en.wikipedia.org/wiki/Role-based_access_control

In this model structure

- **Organization:** Organization, that is, the structure with more than one Role, Role hierarchy.
- Role:** At the organization level, administrator, user, customer, etc... are the concepts we create with jobs and define authorizations.
- **Subject:** The person using the system or another system.
- **Resources:** Resources.
- **Operations:** These are the operations that can be done on resources. Read, Write, etc..
- **Permission:** These are our definitions of operation authorizations on the resource(s).
- **Session:** The subject is the Role and Permission connections on Resources.

Attribute-Based Access Models (Policy-Based)

In the example above, authorization is set up through Roles. At the same time, Attribute-Based recommends creating and using Attribute (Policy-based through Properties) authorizations for more detailed and flexible use.

Qualities

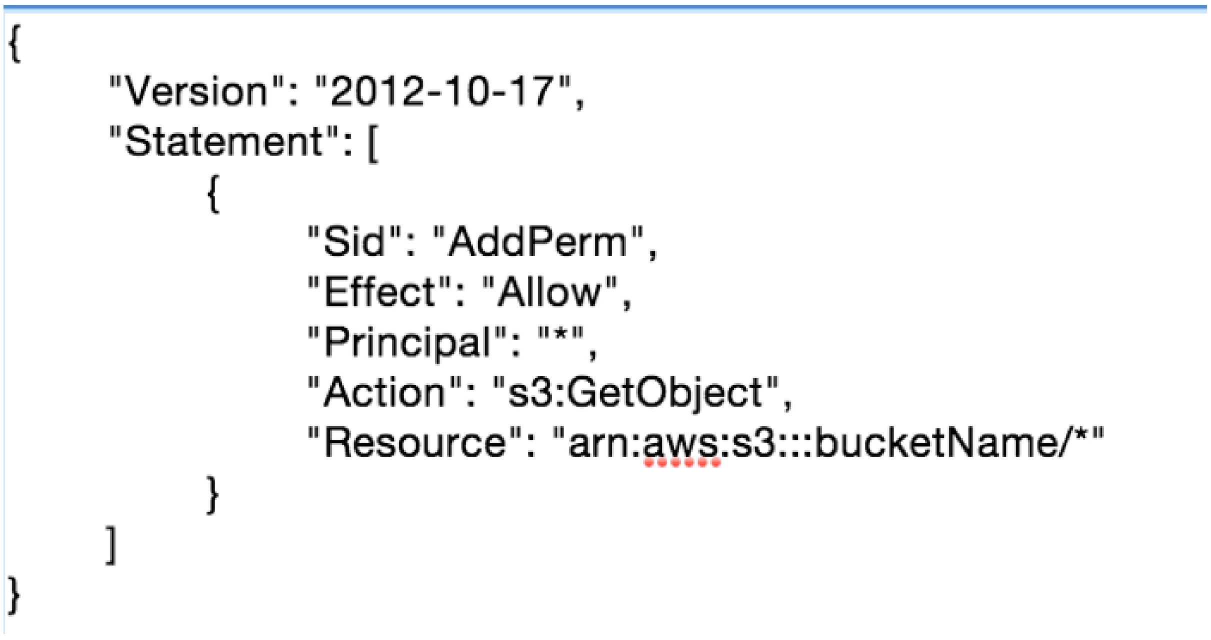
While attributes can belong to anything or anyone, they can be examined in 4 categories;

1. **Subject Attributes:** The attributes of the Person Using or other system using, role, job, place of residence, age, system speed, etc...
2. **Action attributes:** Attributes that describe the action being attempted read, write, see, change, bind
3. **Resource(Object) attributes:** Attributes of the accessed resource or Object (Bank Card Number, CCV number, Code, Date...)
4. **Contextual (environment) attributes**: Attributes that deal with time, location or dynamic aspects of access.

As you can see, Policy Based structures (policies) built on attributes allow us to authorize in a very flexible structure.

It allows us to define policies in great detail.

- Do we prohibit it, or do we allow it?
- Which Resource are we going to operate on?
- What is the action on it...



```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Sid": "AddPerm",
                        "Effect": "Allow",
                        "Principal": "*",
                        "Action": "s3:GetObject",
                        "Resource": "arn:aws:s3:::bucketName/*"
                }
        ]
}
```

Permission.JSON

As above, when we define many policies over attributes and assign them to roles, we can now change roles according to attributes dynamically.

As a Result

In this structure, according to the organizational chart that will use SaaS or Web application;

1st Need Identification Screens

- defining roles
- Define the application resources (pages, buttons, lists, etc...)
- define which resources these roles will have an impact on

2nd Need Invitation Mechanisms

- So how do we invite people to the same organization or workspace? inviting by email and creating users for people who accept this invitation
- development of invitation and invitation acceptance systems via email

3rd Need resources (Summary of implementing Read, Write authorizations on.)

- When invited users enter the system, running mechanisms to view or modify these resources as **Read, Write** according to their authorization.
- Can more than one work on these resources at the same time? How will collaboration work that is? Can users work together on the same resource in real time and make changes?

In summary, the 4th need is to keep Logs and create Reports, Dashboards, or Alarms with several queries through them and mechanisms that enable the relevant people to be informed.

Well, some people made changes to these resources. Keeping logs of these changes and creating reports or alarms if necessary. In other words, if we consider every action taken by users as follows

- Action
- Time

- Resource
- Etc.

System Administrators can create reports, dashboards, or alarms with certain queries through them and request them to be sent to email or any notification channel.

The four features I described above will cause your product to be much more difficult than you think.

Note: There is one more part I did not write here. Apart from the application, the existing system's organizational structures, and the application's integration, you will develop with other resources here.

For example, if an **Active Directory** structure is used, it may be desirable to operate the same organizational structure in your product similarly and to log in similarly.

D. Complex Level

(Seviye — 6) Private API Usage, Authentication, Authorization, Offer, Order, Invoice

In the previous sections, **money** was not involved in **data interaction**. When money is involved, the subject starts to become the most complex. Why?

First, when money is involved, you become responsible for the State, Bank, and Legal aspects.

- Creating Invoice or Invoice after Payment
- Creation of Contract Texts
- The 3 structures I mentioned above are very critical. Ensuring that these steps flow (Transaction) take place correctly, or if there is any problem, that the related transactions can be undone or canceled

Secondly, what kind of products will we sell or rent, physical or as a Service?

- If physical, does this product exist in the system/storage or not?
- If service, authorization arrangements, and preparation of infrastructures for this product service

Thirdly, we must sell or market these products through some pages.

- Creating a market area or marketplace
- Identification of products or product parts
- Creation of campaigns
- Establishing Fee Structures
- Establishing a connection with the warehouse structure

4thly, Delivery of these products or services to customers;

- If it is a software service sold, preparing the infrastructure of the relevant customer organization structure to use this service
- If the product is sold, cargo and other processes need to be initiated to deliver it to the customer
- And this whole process needs to be monitored and followed

5th To remedy an error, malfunction, or problem that occurs during the use of these products or services

- Establishment of return systems
- Creation of Customer Support, Customer Sales and Communication, etc. structures
- Creation of fault recording structures if service
- And keeping records of all this work
- *6thly, Reporting, Dashboard, Warning Mechanisms, Security, Communication, Contract, etc... many infrastructures need to be established, and screens and data need to be kept.

These kinds of applications I mentioned;

- SaaS
- E-Commerce
- E-Learning
- Telco

Frontend applications that provide Product/Service services receive money from their customers in return and issue invoices.

As can be understood from the six items above, things become very complicated and complex if money is involved.