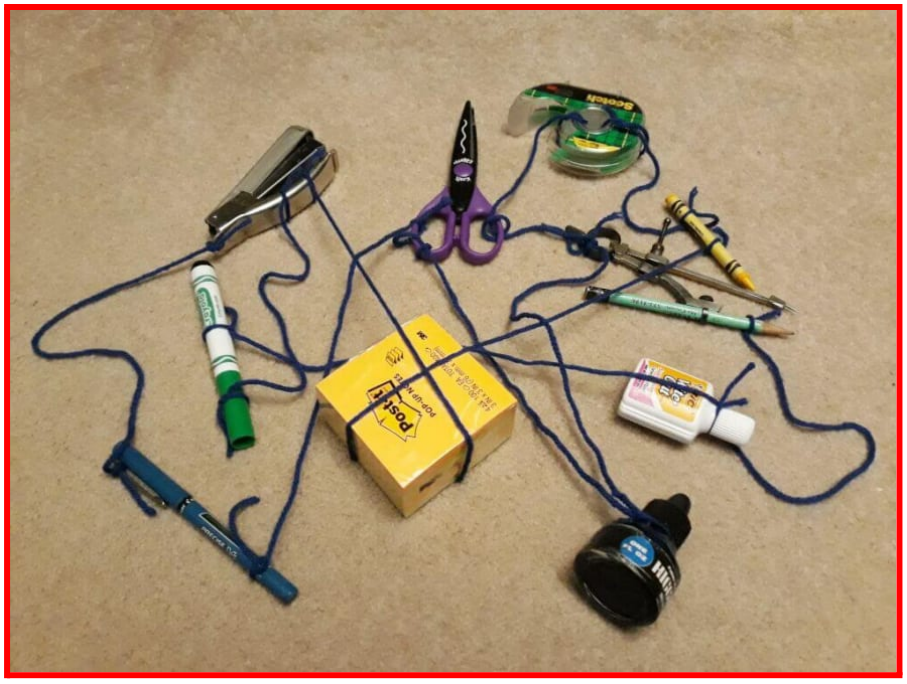
Een andere benadering van de frontend-architectuur

De voordelen van een goede architectuur

Laten we, voordat we in technische zaken duiken, eerst een klein probleem oplossen:



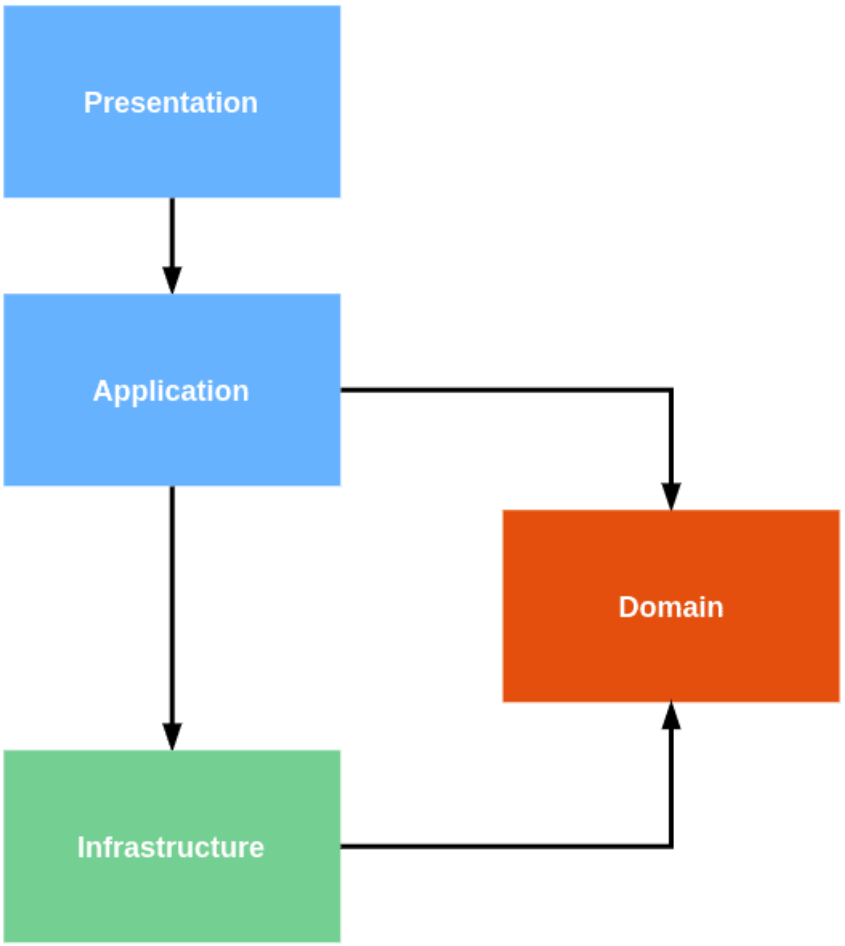
Kan je mij in de bovenstaande afbeelding in één oogopslag vertellen hoe je de nietmachine door een tape kunt vervangen? Sommigen van jullie kunnen een interessante manier bedenken om dit te doen, maar voor de meesten van ons kunnen ziet dit er uit als een puinhoop..het verwart onze hersenen. Kijk nu eens naar dit:

Kan je me nu meteen vertellen hoe ik de nietjesmachine moet vervangen? We hoeven alleen het touwtje dat eraan vast zit los te maken en de tape op zijn plaats te leggen. Je hebt een bijna geen mentale inspanning nodig…

Stel je voor dat alle items in de bovenstaande afbeeldingen modules of onderdelen in je software zijn. Een goede architectuur zou meer op de 2de afbeelding moeten lijken. De voordelen:

- Vermindering van de cognitieve & mentale inspanning bij het werken aan het project.
- Modulaire code, en dus beter testbaar en onderhoudbaar.
- Het proces van het vervangen van een bepaald onderdeel in de architectuur vereenvoudigen.

De andere aanpak introduceren



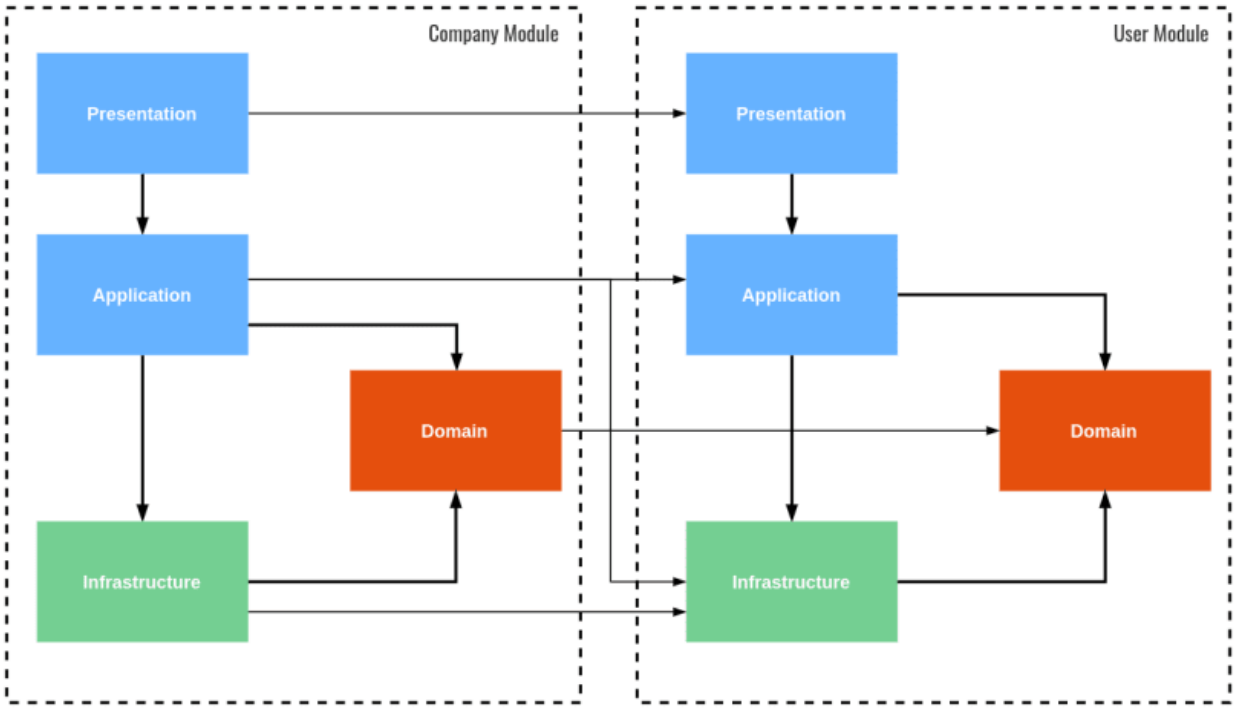
Presentatie: Deze laag is in principe gemaakt van UI-componenten (layout). De presentatielaag is direct gekoppeld aan de applicatielaag.

Applicatie: Deze laag bevat applicatie logica. Het praat met de domein laag en de infrastructuurlaag.

Domein: deze laag is puur voor domein-/bedrijfslogica. Alleen bedrijfslogica leeft in de domein laag, dus er is hier alleen pure JavaScript/TypeScript-code zonder frameworks libs of toeters en bellen

Infrastructuur: Deze laag is verantwoordelijk voor de communicatie met de buitenwereld (verzoeken versturen/antwoorden ontvangen) en het opslaan van lokale data. (Fetch API)

HTTP-requests: Axios, Fetch API, Apollo Client, enz.
(State Management): Vuex, Redux, MobX, Valtio, etc.

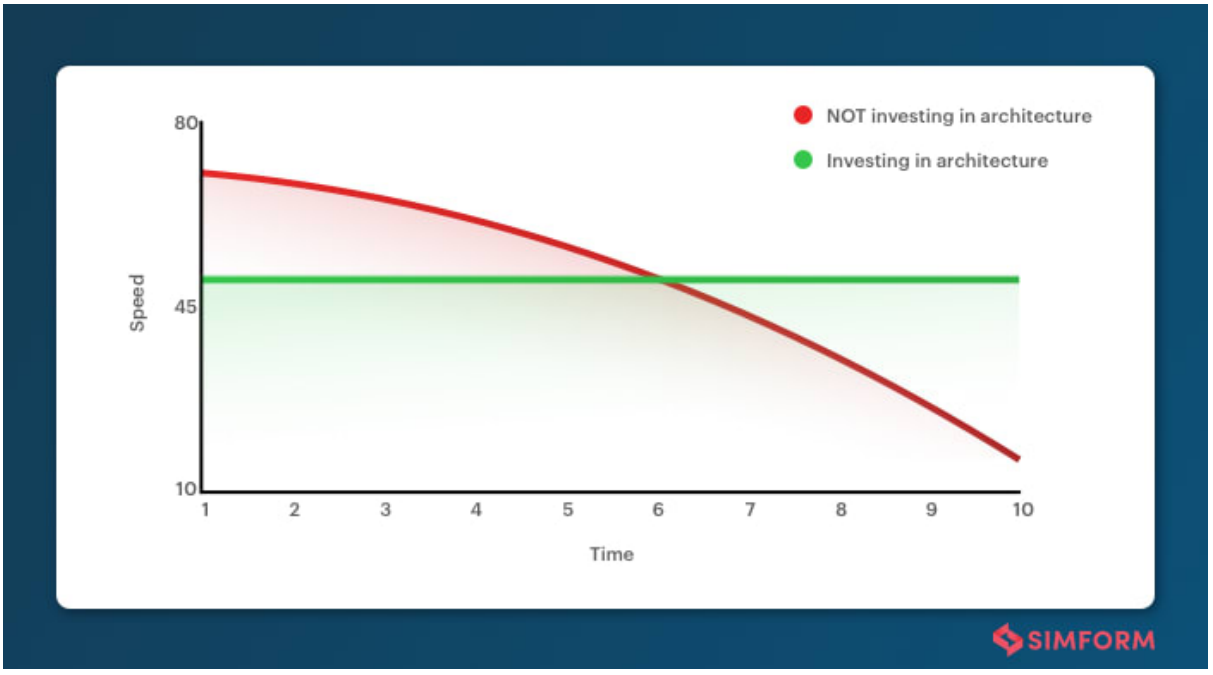


Note: The company module can depend on the user module because in reality, a company is always associated with people (users).

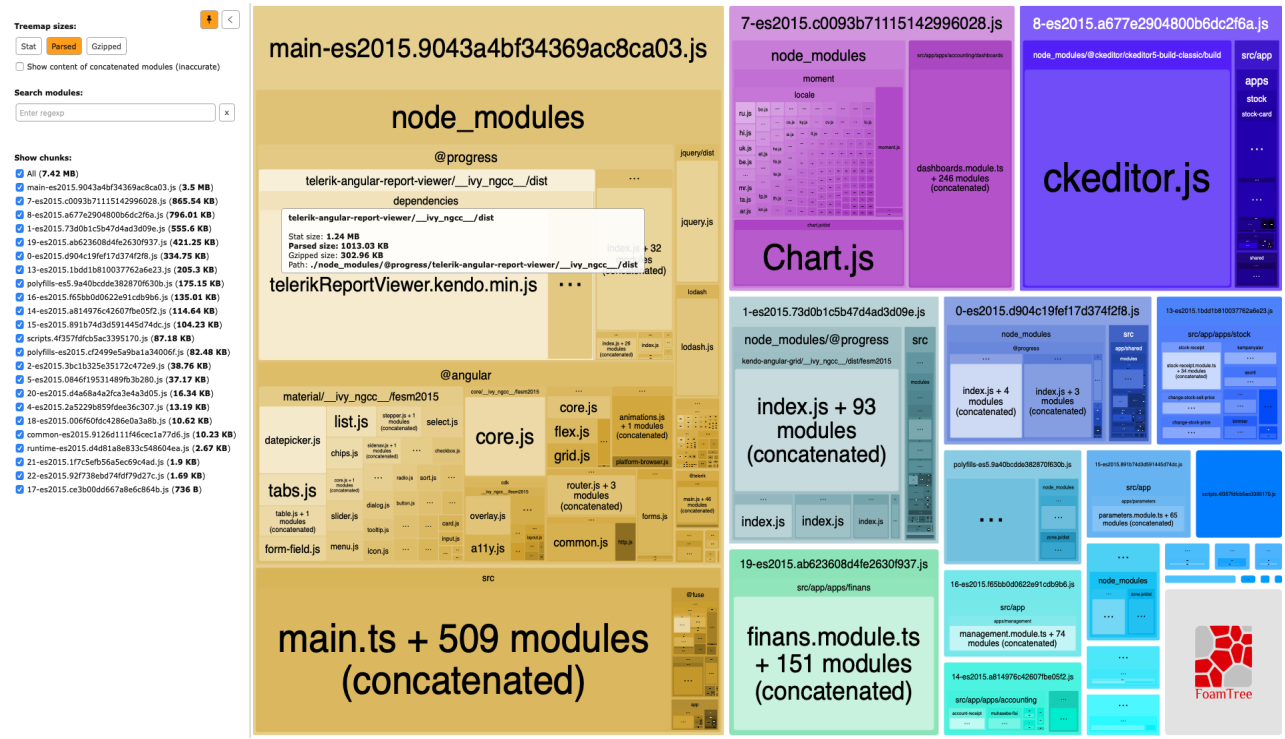
Opgelet:

Hoewel de architectuur alle onderdelen van de applicatie van elkaar kan loskoppelen, is er een nefaste caveat: verhoogde complexiteit. Voor een kleine applicatie is dit gewoon niet werkbaar. Gebruik geen bulldozer om de badkamer te kuisen!

WEL OF NIET??? HMMM...



Frontend architecture is never a “set it and forget it” proposition — no design or plan is ever perfect or complete



De denkoefening

1. Hoeveel stukjes code & snippets zijn direct herbruikbaar?
2. In hoeverre zijn modules afhankelijk van andere modules in het systeem?
3. Als bepaalde onderdelen van de applicatie falen, kan deze dan nog functioneren?
4. Hoe gemakkelijk kun je de afzonderlijke modules testen?

Denk op lange termijn

De 2 software dev constanten:

1. De applicatie is nooit af
2. There is no such thing as bugfree (en no... bugs are not a feature)

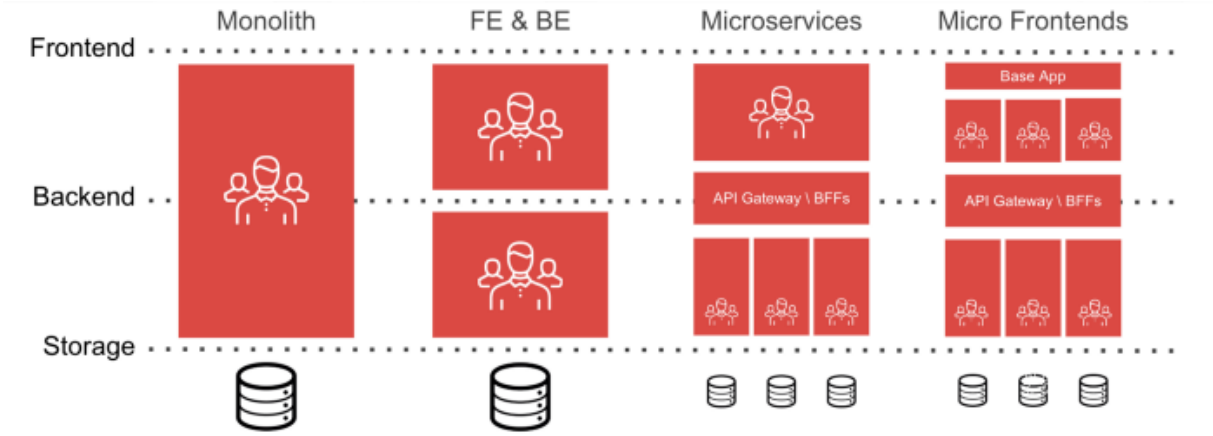
Bij het conceptualiseren van de architectuur is het belangrijk om vooruit te denken. Niet alleen over een maand of jaar, maar ook daarna... Wat kan er veranderen? Wat zijn de mogelijke technische evoluties? Welke feature sets stonden nog op mijn MOSCOW lijst. Het is natuurlijk onmogelijk om dit 100% in te schatten, maar toch het overwegen meer dan waard.

"Hoe meer componenten met elkaar verbonden zijn, hoe minder herbruikbaar ze zullen zijn, en hoe moeilijker het wordt om wijzigingen aan te brengen in de ene zonder per ongeluk de andere te beïnvloeden of botweg kapot te maken"

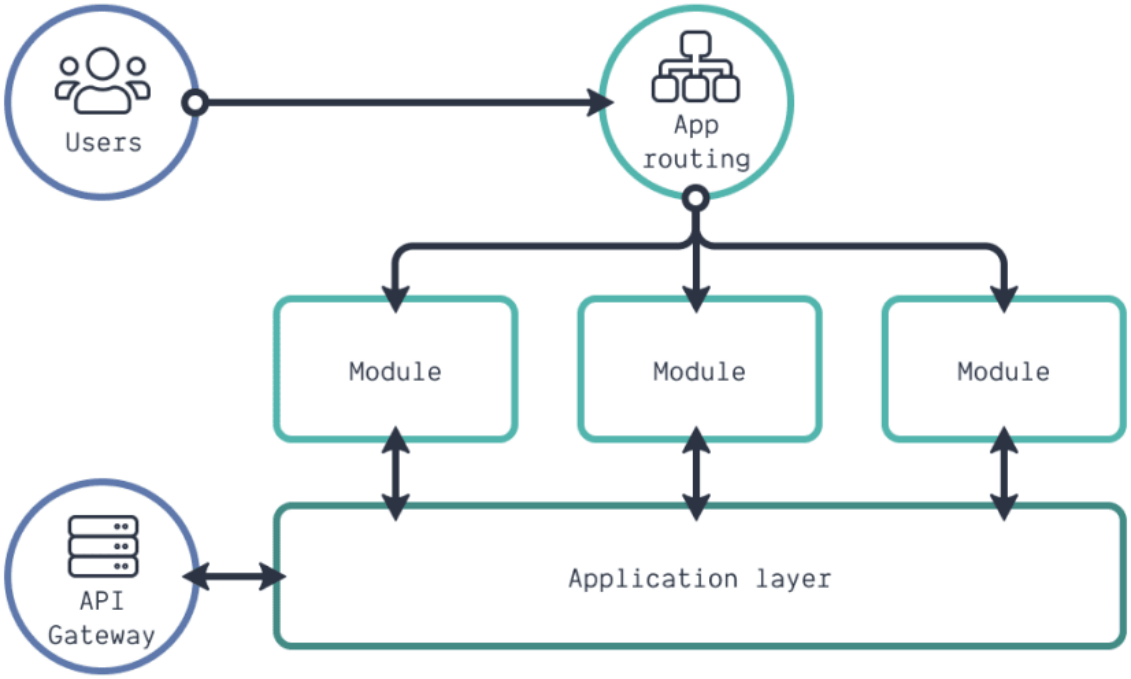
Deze principes zijn essentieel voor het bouwen van een architectuur die de tand des tijds kan doorstaan en moet altijd in gedachten worden gehouden.

"Het geheim van grote apps is nooit grote apps te bouwen. Breek je applicaties in kleine stukjes. Voeg die testbare, hapklare stukjes vervolgens samen tot je grote applicatie"

Evolutie architecture (80-Present)



https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

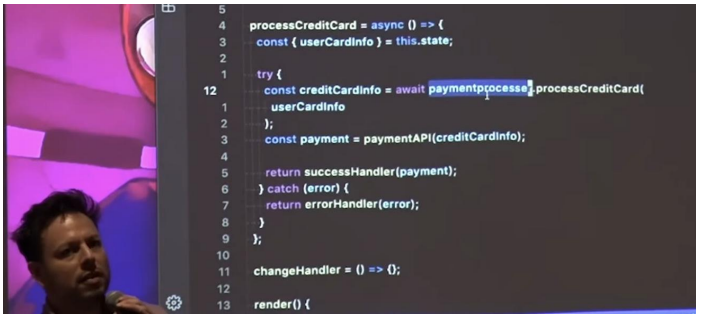


Design Patterns

- Singleton
- Strategy
- Observer
- Decorator

"Ontwerp Patronen zijn typische oplossingen voor veelvoorkomende problemen in software ontwerp. Elk patroon is als een blauwdruk die je kan aanpassen om een bepaald probleem in de code op te lossen."

https://anuraghazra.dev/blog/design-patterns-everyday#Day-4
https://anuraghazra.dev/blog/design-patterns-everyday#Day-17
https://anuraghazra.dev/blog/design-patterns-everyday#Day-15
https://anuraghazra.dev/blog/design-patterns-everyday#Day-8



https://www.youtube.com/watch?v=jmcx3b78V8s&t=1443s