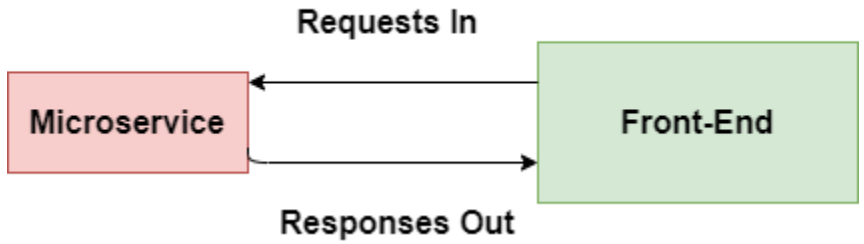
COSC 331 – Microservices and Software

Fall 2020

Recap: Microservice Frontends

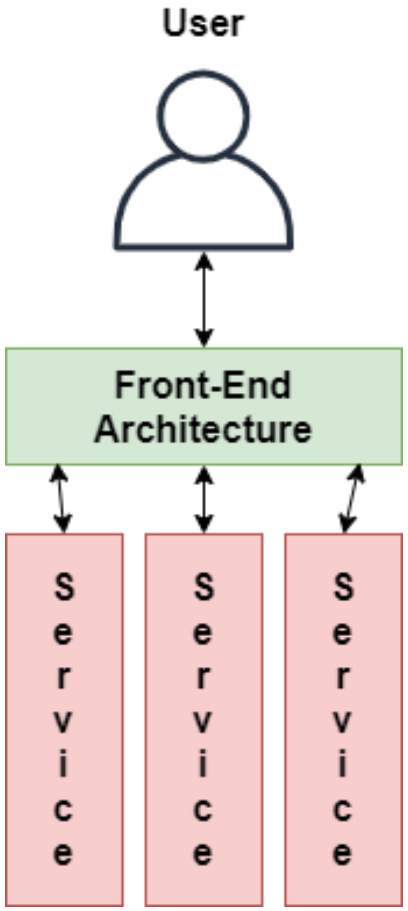
- Last lecture, we looked at the design of microservice front-ends
- This front-end could be client or server sided, depending on the application, although each has it's own benefits and drawbacks
- Balancing the load between the front-end architecture and the microservices is tricky; for scalability we want the front-end to be lightweight with minimal processing required



An Alternate Way of Looking at Front-ends

- Rather than look at front-end design as being simply split into client-side and server-side approaches, we can also talk about ***monolithic and non-monolithic design***
- Recall that monolithic application design is essentially the opposite of a microservices architecture – everything is built into a singular application
- So what then does it mean to have a microservices front-end that is monolithic?

The Monolithic Front-end



- Both the server-side and client-side front-end designs we've looked at so far could be considered ***monolithic***
- This is because the front-end architecture is designed to provide a single, unified output
  - With these designs, all data is forced to go through the front-end, which is responsible for combining and rendering all output into a cohesive UI

Benefits of Using a Monolithic UI

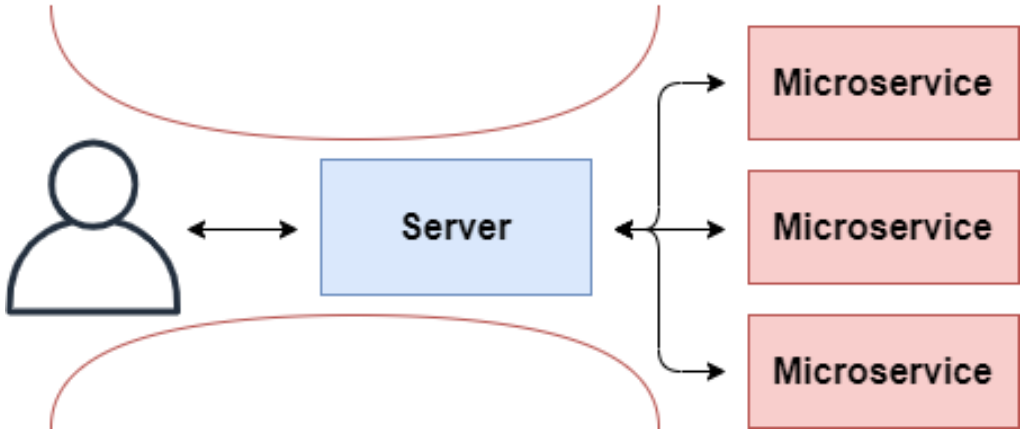
- A monolithic front-end has some advantages:
- All UI-related code and components can be kept to the front-end, preventing our microservices from becoming cluttered with UI-related code
  - This also eases maintenance and debugging, as we can trace UI issues back to a singular point, rather than trying to figure out which microservice is failing
- Provides a ‘buffer’ between clients and the microservices that they consume – good in terms of security and fault-tolerance

Drawbacks to a Monolithic UI

- A monolithic UI can become unwieldy as additional services or components are added
- Requires bespoke design and implementation – the monolithic UI cannot be easily repurposed or re-used; it is designed specific to the application and services that it is used with
  - Remember: one of the major benefits of a microservice architecture is the ability to re-use and recycle

The Biggest Drawback

- We've already touched on the biggest drawback of a monolithic design in the last lecture: the creation of a singular point of failure
- If a monolithic front-end fails, the entire application fails, even if the microservices behind the application are fine

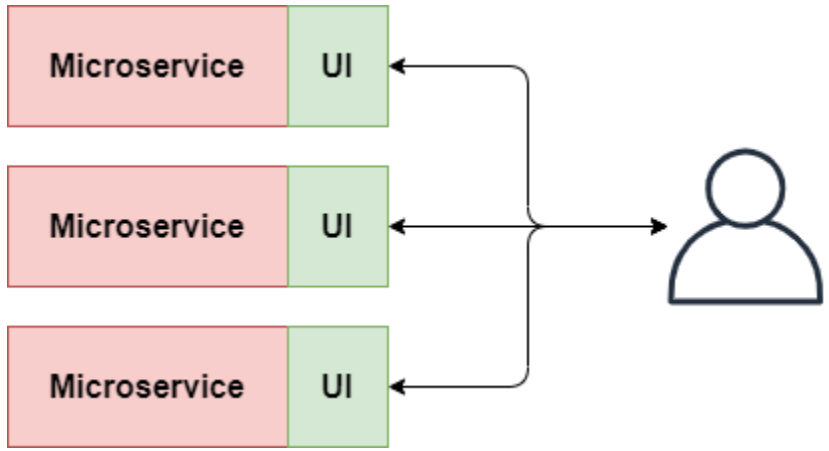


But What About Client-Side?

- At this point you might be thinking: But surely a client-side front-end isn't monolithic – after all, it's directly communicating with the microservices with no intermediary
- But if our client-side front-end is still responsible for processing the data from the microservices to produce a usable, unified UI, then it's still monolithic
  - All we've done is move the monolithic UI from an intermediary server to the client-side, in that case

Micro Frontend – the Alternative

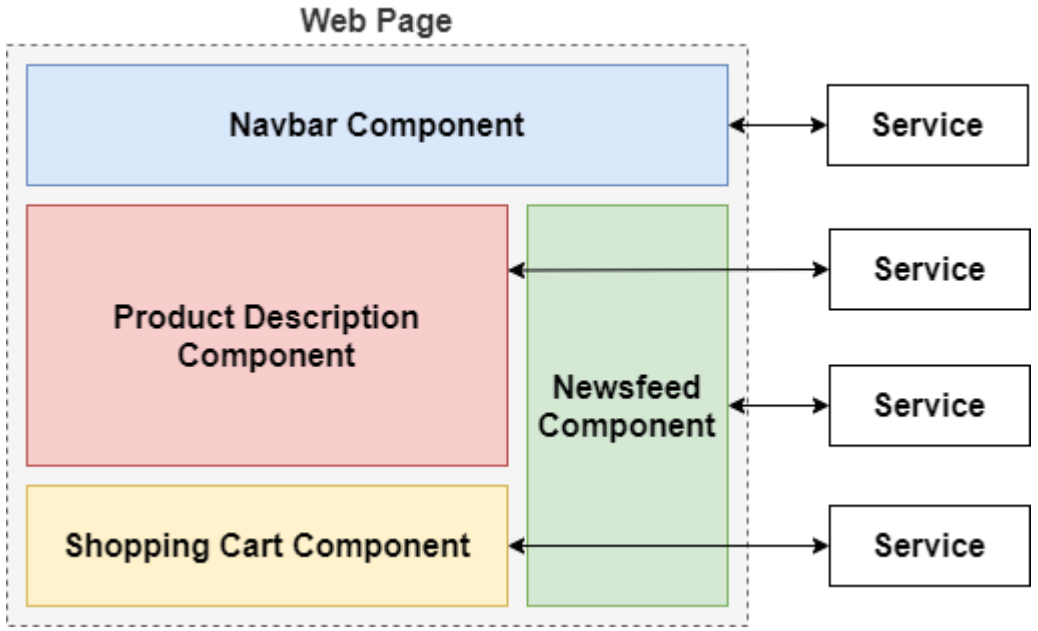
- So if we don’t want to move the UI handling from a dedicated middleman to the client, what if we try moving the UI in the opposite direction – to the microservices?
- This is known as the ***micro frontend*** approach, where each microservice serves as it’s own frontend as well



Micro Frontend Design

- The core concept behind micro frontend design is that a microservice isn’t responsible for the entirety of the UI, but only those UI components related to it’s functionality
- Rather than returning data in JSON or XML format that needs to be parsed and interpreted, the microservice instead returns complete HTML for it’s component – no additional client-side processing necessary, besides simply putting the HTML onto the page

Applications as a Collection of Components



- The micro-frontend breaks up application pages into a collection of individual modules, each maintained by a separate service
- Each service provides all the logic and data functionality required, in addition to producing HTML which is then passed to the client for rendering

Working Asynchronously

- In a traditional monolithic web application, things are usually done ***synchronously*** – after a client request comes in, the client waits while the server prepares the response, and then the server waits for further client input
- By breaking up the application into distinct sub-modules with their own UI rendering, we don't need to sit around and wait – we can render things as they are ready
  - We can render faster components (i.e. navbar) rapidly, while we wait for larger components to return their responses

The Curious Case of JavaScript

- Having a front-end architecture that can serve pages asynchronously is actually quite an advantage, due in part to the drawbacks of JavaScript
- JavaScript is single threaded, and waiting on a request synchronously will block until the request is completed
- However, JavaScript can make asynchronous calls, continuing with what it was doing while it waits for a response
  - And since we aren't doing any UI processing on the client-side, we can load components basically as soon as their asynchronous requests complete and we get a response

Micro Frontends in the Wild

- If you've ever noticed that a web application, particularly single-page applications, tend to load in "chunks" of modules, you were probably seeing a micro frontend
- Compare this to a traditional static web page, where the HTML is served all at once (although other resources may take longer to load in, like videos or images)
- Many major enterprises use micro frontends, including IKEA, Spotify, and Microsoft

Benefits of a Micro Frontend

- Micro frontends allow us to build applications in a piecemeal way – we can fully develop and test an individual component, including both the frontend and backend functionality, separately from the rest of the application
- When working in teams, micro frontends split the UI workload, allowing each team to maintain the frontend for their specific service
- Avoids some of the pitfalls like bottlenecking that come from having a monolithic frontend, while still being capable of providing security like a server-sided application

Drawbacks of a Micro Frontend

- Micro frontends are controversial – are they truly a “micro” service? Does integrating the UI component increase the size and complexity of a microservice to the point that we should just consider them small web applications?
- Micro frontends fragment the UI code of the application across multiple services – this can make maintenance and UI overhauls more difficult
- They introduce a level of complexity that is often not necessary for smaller projects – easier to design one UI and three microservices, than to design three microservices and three UIs

In the Lab

- In the lab this week, we'll be designing a simple, Python-based microservice that implements it's own UI – our very own micro frontend
- We'll then be using JavaScript to asynchronously fetch our UI component, as well as to communicate with our styling service that we designed in Lab 1
- A kind of hybrid architecture of client-side frontend and micro frontend – but this is the kind of thing possible when working with microservices

Reminder - Quiz

- Just a reminder that there will be a Moodle quiz available in this course today (later this evening)
- Quiz will be open for a period of one week, ending on Wednesday at 11:59PM
- Should be fairly short (20-30 minutes)
- Only one attempt will be allowed, so make sure that you're confident in your answers before you submit

Any Questions?