Frontend - React JS Vs Svelte JS

1) React JS uses Virtual DOM whereas Svelte JS directly updates the DOM. This could really impact performance. DOM Updates are really costly in terms of time. So in most of the cases, including our case React JS is faster compared to Svelte JS
2) React JS is mainly a "V" framework, whereas Svelte JS is closer to "MVC" framework. So with Svelte JS it gives a clear way for separating state, view and controllers.
3) React JS uses the Runtime approach to load applications whereas Svelte JS uses the Compile time approach. So React JS applications tend to b larger in size and initial load time will be more compared to Svelte JS applications
4) React JS has a strong community and it has 1635 contributors on Github whereas Svelte JS has only 646 contributors on Github. Having great community will help a developer to fix his bugs quickly

Backend - Spring Boot Vs Go

1) Go based applications have very less start up time and memory consumption compared to Spring Boot based applications. Therefore Go applications are highly performant
2) Though Go has a Garbage collection it is not as good as Spring Boot Garbage Collection.
3) Go compiles to a single binary which makes it easier to deploy with lesser dependencies compared to spring boot.
4) Spring Boot is an integrated MVC framework. Whereas Go is a programming language, so it uses its libraries to build things. So in Go it is more on developer how he wants to structure the project and what libraries he is using
5) Go is known for its concurrency support and it is much better than spring boot
6) Spring Boot has 1035 contributors and Go has 2031. Both are dynamic and have good support. Though Go is gaining greater popularity these days

Architecture aspects - same for both the cases

1) Considering it is a chat application, I need real time updates so it is costly for me to keep a view on the server as it requires reloading the page every time. So we have decided to keep "V-view" on client side
2) As many users will be connected to the chat, keeping controllers and models doesn't work well because a single message has to be broadcasted to everyone connected to the chat. Therefore we felt it is wise to keep "M-model" and "C-controller" on the server side as it gives less latency to broadcast the message to everyone connected to the chat

Key takeaways

1) We felt firstly while building any application it is important to understand our domain and its requirements. In our case a chat application where real time messages are critical.
2) As developers, we should write modular code that remains clear and navigable even after a year. We will keep this in mind when building our project