The Need For Micro Frontends

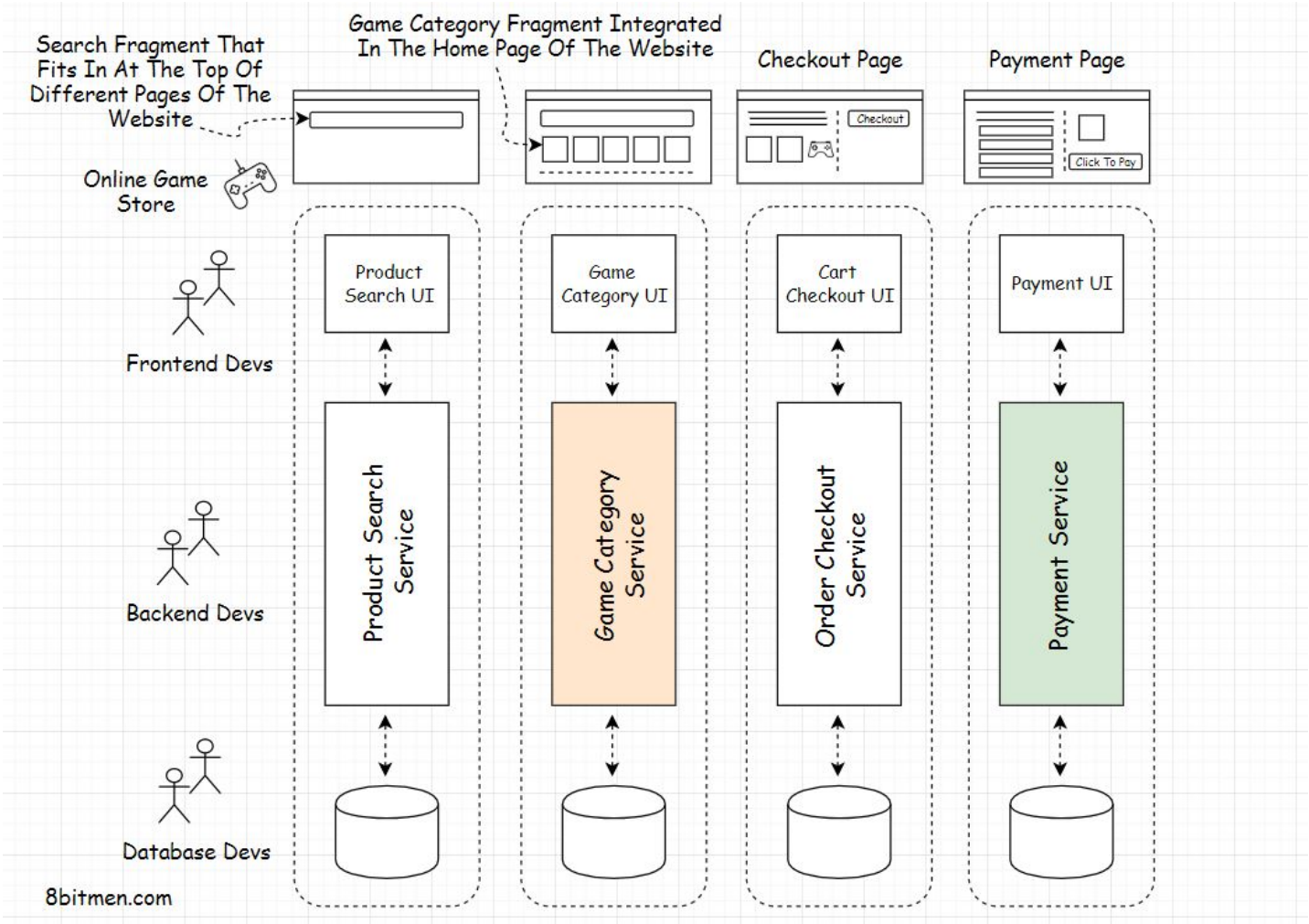
This lesson continues the discussion on micro frontends from the previous lesson

We'll cover the following

- Easier Co-ordination With The Front End Devs
- Leveraging The Right Technology

Individual teams may own a dedicated page in the application like the *checkout page*, they may own a component that fits in a certain page like the *game category* component on the *home page* or they may own both - a dedicated page and a smaller UI component that fits into another page of the website.

These smaller components that integrate into pages are known as *fragments*.



Now you might be wondering splitting up the monolith UI is fine but why do it? What's the point?

Easier Co-ordination With The Front End Devs

When we have full-stack teams owning an entire service end to end, this averts the need for a dedicated front-end team. Since the front end devs now work along with the backend devs in the same team this saves a lot of time that was initially spent in the cross-team coordination that was between the backend microservices and the dedicated front-end teams.

With this new approach, communication is quick and not so formal. This improves the team's productivity, also enables them to deliver better user experience by having more effective coordination between the backend and the frontend devs.

Leveraging The Right Technology

Since, the micro frontends are loosely coupled, just like microservices, we can develop them leveraging different technologies. This lets us off the hook of sticking to just one UI technology to build the entire front end of the website.

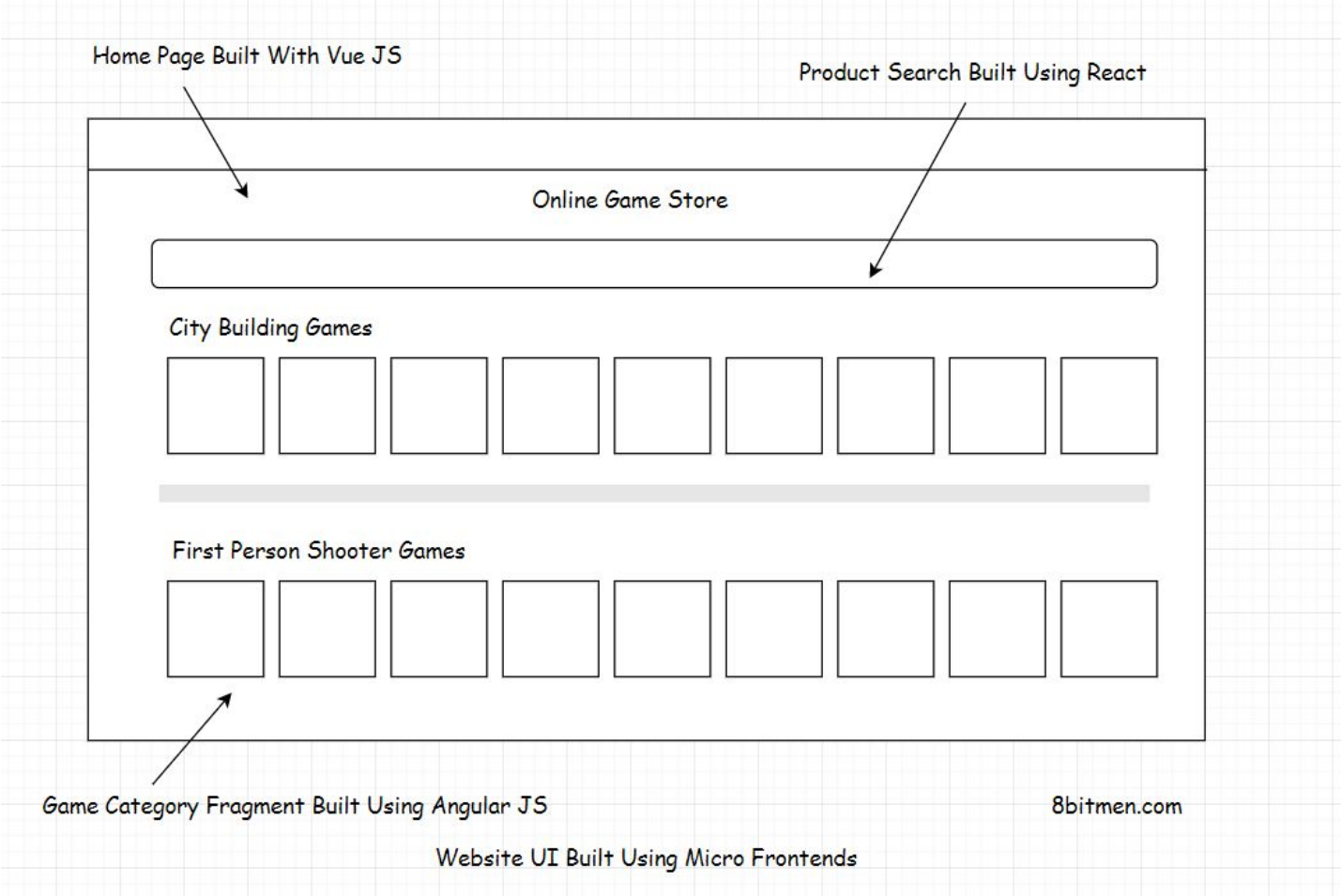
There are a plethora of existing front-end technologies in the industry catering to different use cases, besides the new wave of *JavaScript* frameworks that hits us every year.

With the micro frontends approach, we can pick the right technology to build our front-end component. We often have use cases where just plain *JavaScript, Html & CSS* suffices to build a feature and then there are other cases where we need advanced frameworks like *React, Angular & Vue* to build our feature.

With micro frontends, we don't necessarily have to use *React* to build our component if we don't need to, even if the other components of the website are built using it.

On the contrary, if our website is built on plain *JavaScript* and we want to use *React*, we don't have to re-write the entire website to use *React*. We can just write our component using *React* and then can integrate it into the website.

Even if multiple teams use the same technology to build their *UI* components they can work on different versions of the technology. They can easily upgrade their libraries without impacting the other *UI* components of the website.



Going forward with the micro frontends approach may sound delightful but it's only fit for medium to large websites. This approach won't be that advantageous for simple use cases. Rather it will make things more complex.

Using multiple technologies in a project brings along a lot of architectural & maintenance complexities with it.

With micro frontends, we also need to write additional logic to club all the components together.

We also cannot ignore the compatibility & performance issues when using multiple technologies together. So, there are always trade-offs involved. There is no silver bullet.

Here Is An Interesting Watch – Engineering Culture At Spotify

In the next lesson, let's have a quick insight into how these micro frontends are integrated with each other.