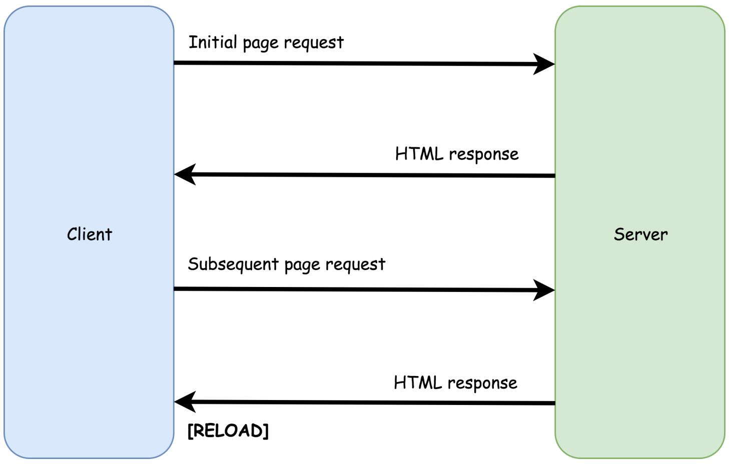
Single Page Application

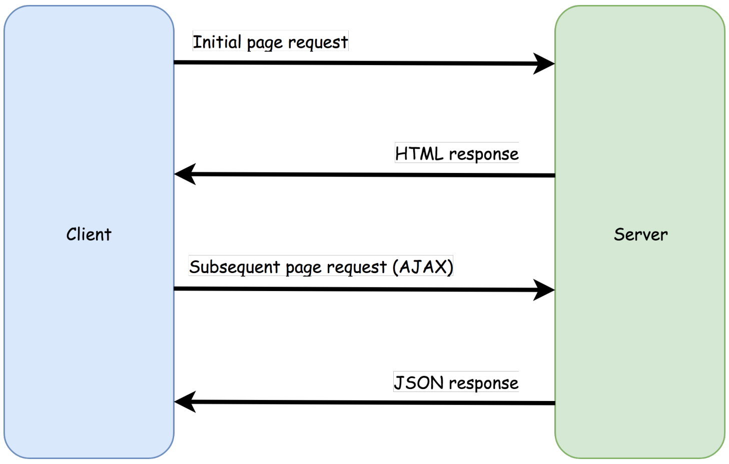
Most websites are broken up into pages in order to make the information they contain easier to consume. The traditional architecture is to give each page a unique URL. To navigate to a page, the browser sends a GET request to the page's URL. The server will send back the page and the browser will unload the existing page and load the new one.

For the average internet connection, the navigation process will likely take a few seconds, during which the user must wait for the new page to load.



With JavaScript and web APIs like XMLHttpRequest, a different model is possible: **the browser can load an initial page, but navigating to new pages will not require the browser to unload the page and load a new one.** Instead, the page content can be loaded from an API asynchronously with AJAX and then written into the current page with JavaScript.

From a user's perspective, such a website would appear to have pages just like any other, but from a technical perspective, this site really only has one page. Hence the name, single-page application.



Routers
A router library is the engine of the SPA architecture. It will mimic browser navigation through JavaScript and various web APIs so that the user gets an experience similar to that of a traditional multi-page app.

Routers will typically include functionality to:

- Handle navigation actions from within the page
- Match parts of the application to URLs
- Manage the address bar
- Manage the browser history
- Manage scrollbar behaviour
- Improving UX

Improving UX
The intention of the single-page application architecture is to improve UX, and it does so in the following ways:

1. SPAs can provide a **more continuous experience for the user**, as navigation no longer requires a page refresh. Data for new pages must still be retrieved, and will, therefore, create some small disruption to the user's flow, but this disruption is minimized since the data retrieval can be done asynchronously and JavaScript can continue to run.

2. Once the SPA has loaded, **navigation between pages is quicker** because SPAs will reuse page elements and therefore won't need to keep downloading the same repeated markup. However, a router library will need to be added to your JavaScript bundle, so keep this in mind when doing the accounting.

Pitfalls
Ironically, single-page applications can harm UX if certain pitfalls aren't avoided:

1. SPAs **break native navigation functionality**. e.g. scroll position, history, back button etc. Once a router has hijacked page navigation, these features may not work as expected. An SPA must restore the functionality with JavaScript and web APIs like history.pushState. Most good router libraries will help you do this, but there will still be some manual implementation required.

2. SPAs have a **large initial download size**. Since the router and multi-purpose page elements must be downloaded first for the app to work, SPAs require an upfront download before they run. Build tools like Webpack can help by lazy-loading any code not needed before the first render.

3. SPAs will need **custom loading states and errors messages**. Browsers give visual cues that a page is being loaded, and a web server can return a 404 page. The result of an AJAX request, on the other hand, is hidden from the user by design. SPAs must find a way to let users know if the app has successfully responded to their actions or not.

4. With a naive implementation of the SPA architecture, **page content may not be included in the initial page download**, which means a user may have to wait for JavaScript to run and AJAX calls to complete. Server-side rendering or pre-rendering is a solution to this but often requires a complex setup.

Conclusion
The purpose of the SPA architecture is to provide superior user experience, but unless proper care is taken, it can have the opposite effect!

Here are the key things to keep in mind if you choose the SPA architecture:

- Configure your router software so native navigation features aren't broken
- Employ build tool features like code-splitting and lazy-loading to ensure the initial code bundle isn't too big
- Implement loading states and error messages so that the user knows the page is responding to their actions
- Use prerendering or server-side rendering to ensure your SPA shows content as early as possible

Above all, make sure you have budgeted for the extra work required for building, testing and maintaining an SPA.

https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

Pros of the Single-Page Application:

- **SPA is fast,** as most resources (HTML+CSS+Scripts) are only loaded once throughout the lifespan of application. Only data is transmitted back and forth.

- **The development is simplified and streamlined**. There is no need to write code to render pages on the server. It is much easier to get started because you can usually kick off development from a file file://URI, without using any server at all.

- SPAs are **easy to debug with Chrome**, as you can monitor network operations, investigate page elements and data associated with it

- **It's easier to make a mobile application** because the developer can reuse the same backend code for web application and native mobile application.

- **SPA can cache any local storage effectively**. An application sends only one request, store all data, then it can use this data and works even offline.

Cons of the Single-Page Application:

- **It is very tricky and not an easy task to make SEO optimization of a Single-Page Application**. Its content is loaded by AJAX (Asynchronous JavaScript and XML)—a method of exchanging data and updating in the application without refreshing the page.
  UPDATE 27.09.2017: In her comment, Iris Shaffer correctly pointed out that it can be done on server side as well. Indeed, it is easier today than it used to be.

- **It is slow to download because heavy client frameworks are required** to be loaded to the client.

- **It requires JavaScript to be present and enabled**. If any user disables JavaScript in his or her browser, it won't be possible to present application and its actions in a correct way.
  UPDATE 27.09.2017: In her comment, Iris Shaffer has noticed that with isomorphic rendering / server side rendering, you can render the page on the server already. When the initial render is on the server and can be cached, disabling JS would not be a concern for getting a rendered page. Theoretically, that's right. Obviously you can render on server side. But lack of JS can be a concern for other functionalities. Lots of things can be done in HTML & CSS but from my experience it would be hell to do it this way instead of use JavaScript.

- Compared to the "traditional" application, **SPA is less secure.** Due to Cross-Site Scripting (XSS), it enables attackers to inject client-side scripts into web application by other users.

- Memory leak in JavaScript can even cause powerful system to slow down

- In this model, **back and forward buttons become dysfunctional**
  UPDATE 27.09.2017: Nowadays, it's not an issue anymore.

Multi-Page Application:

Multiple-page applications work in a "traditional" way. Every change eg. display the data or submit data back to server requests rendering a new page from the server in the browser. These applications are large, bigger than SPAs because they need to be. Due to the amount of content, these applications have many levels of UI. Luckily, it's not a problem anymore. Thanks to AJAX, we don't have to worry that big and complex applications have to transfer a lot of data between server and browser. That solution improves and it allows to refresh only particular parts of the application. On the other hand, it adds more complexity and it is more difficult to develop than a single-page application.

Pros of the Multiple-Page Application:

- It's the perfect approach for users who need a visual map of where to go in the application. Solid, few level menu navigation is an essential part of traditional Multi-Page Application.

- Very good and easy for proper SEO management. It gives better chances to rank for different keywords since an application can be optimized for one keyword per page.

Cons of the multiple-page application:

- There is no option to use the same backend with mobile applications.
  UPDATE 27.09.2017: Back then, when I was writing this article, I didn't have much experience with backend and with mobile apps. It's obvious for me not that you can use the same backend for both. And I'd like to thank all the readers who pointed that out.

- Frontend and backend development are tightly coupled.
- The development becomes quite complex. The developer needs to use frameworks for either client and server side. This results in the longer time of application development.

SPA or MPA?

Before deploying a web application, you need to consider the goal of it. If you know you need multiple categories (because, for instance, you run an online shop or publish a lot of other content)—use a multi-page site. If you are sure that your site is appropriate for a pure single-page experience—go for it. And if you like SPA but can just barely fit everything into a single page, consider the hybrid site instead. This is another way I haven't mentioned before. A hybrid application takes what is the best in both approaches and try to minimize the disadvantages. It is, in fact, a single page application which uses URL anchors as synthetic pages enabling more in build browser navigation and preference functionality. But this is the topic for another article.

Perhaps in the future, everyone will use Single Page Application model (including a hybrid app), as it seems to bring a lot of advantages. Many apps on the market are migrating towards this model. However, as some projects simply cannot fit into SPA, the MPA model is still vivid.