**(Slide 1)**

Hi everyone.  
Today I'll talk about the **History API** and **Routing in SPA** (Single Page Applications).  
Imagine you open your messenger, and every time you click on a user, the whole page reloads. Annoying, right?

**(Slide 2)**  
As a result, we encounter a problem: traditional websites reload the page,

**(Slide 2-1)**  
while in SPAs, there's no way to go back using the browser's back button or to send someone a direct link to a specific "screen" in the app.  
The solution has already been invented:

**(Slide 3)**  
The **History API**, which allows us to change the URL and manage browser history.  
Today it’s actively used in frameworks like **React** and **Vue**.

Let's dive deeper into the History API.

**(Slide 3-1)**  
It contains three main methods:

* history.pushState – Adds a new entry to the browser history **without reloading the page**.
* history.replaceState – Replaces the current entry in the history (without adding a new one).

Both methods take 3 parameters: (state, title, url)

* **state** – the data you want to save.
* **title** – should set the page title in history (but is often ignored in practice).
* **url** – changes the URL in the address bar; it must be an **absolute** URL.

Additionally, there’s:

* history.go(n) – Moves **n** steps forward or backward (if n is negative) in history.

And one event:

* popstate – triggered when navigating through the history (back/forward buttons).

**Important nuances:**

* pushState does **not** trigger popstate (only navigation with browser buttons does).
* The URL must have the **same origin** (otherwise a security error occurs).

There are also two properties:

* **state** – the data passed into pushState or replaceState.
* **length** – the number of entries in the session history.

**(Slide 4)**  
Now I’ll show you how to build a basic version of a simple router.

**Step 1: Set up an event handler for links.**  
**(Slide 4-1)**

* Check if the link has an href. If not — exit the function.
* Otherwise, **prevent the default behavior** and **change the URL** using pushState (without reloading).
* Then call the function responsible for rendering the page.

**Step 2: Render the page.**  
**(Slide 4-2)**

* Make a request to the server to get the necessary data to render the page.
* here we get the response body, we have html here. and we replace the html pages on this line.
* In our case, rendering is simply replacing innerHTML with the server's response (for example, if the server sends a ready-made HTML page).
* Also, reset the scroll position if needed.

**Step 3: Handle the Back/Forward buttons.**  
**(Slide 4-3)**

* Attach an event listener for the popstate event on the global window object, which will render the correct page.
* Since we changed the URL in Step 1, now window.location.href contains the updated URL.

**(Slide 5)**  
Now I'll tell you a few ways to **improve your Routing**, to help you reach "hacker" level as a programmer.

**(Slide 5-1)**

1. **Saving scroll position** so that when users go back, they land exactly where they left off.

* You can simply pass window.scrollY into the state parameter, and when handling popstate, scroll the page back to the saved position (first checking if scrollY exists in the state — it could be 0).

**(Slide 5-2)**  
2. **Updating page metadata** (title and description).

* This is crucial for **SEO optimization** and for social networks (otherwise, previews might show metadata from the homepage instead of the current page).
* Also useful for updating the **tab title** when switching pages.

**Limitations and Pitfalls**  
**(Slide 6)**

* **SEO**: Without Server-Side Rendering (SSR) or dynamic metadata updates, search engines won't see your content.  
  *(Solution: use Next.js, Nuxt, etc.)*
* **Direct URL access**: The server must return index.html for **all paths**.
* **No JavaScript**: You need a fallback (e.g., normal links for older browsers).

**(Slide 7)**  
Now I’ll show you a demo version of simple routing using the **History API**.  
When you click on a link, the URL updates and the History API changes accordingly.

**Conclusion:**  
**(Slide 8)**

* The **History API** is the foundation of **client-side navigation** in SPAs.
* It allows for creating **fast and seamless** applications.
* It’s the **core** for libraries like **React Router**, **Vue Router**, and others.

**(Slide 9)**  
**Thank you for your attention!**