Lazy loading and code splitting

🚀 Lazy Loading and Code Splitting in JavaScript

Lazy loading and **code splitting** are techniques used to **optimize performance** in web applications by loading only the necessary code and resources when they are actually needed, rather than all at once.

📦 Code Splitting

📌 What is Code Splitting?

Code splitting is the practice of breaking your codebase into multiple smaller bundles ("chunks") that can be loaded on demand.

Instead of shipping one large `bundle.js`, we break it into smaller parts like:

- `home.js`
- `dashboard.js`
- `about.js`

✅ Benefits:

- Faster initial page load
- Reduced memory usage
- Improved performance on slower networks/devices

⚙️ How It's Done:

- **Manually** using dynamic imports:

  ```
  import('./path/to/module.js').then(module => {
    module.doSomething();
  });
  ```

- **Automatically** using bundlers like:

  - **Webpack**: via dynamic `import()`
  - **Vite**, **Parcel**, etc.

💤 Lazy Loading

📌 What is Lazy Loading?

Lazy loading refers to delaying the loading of non-critical resources (like images, scripts, or components) until they are actually needed.

> It’s a runtime behavior that loads the code/resource only when it’s required, e.g., when a route is visited or a component is visible.

✅ Benefits:

- Speeds up initial load
- Reduces resource consumption
- Improves user experience by loading only necessary content

🔄 Lazy Loading + Code Splitting (In Practice)

They often **work together**:

- **Code splitting** creates separate files
- **Lazy loading** loads those files **on demand**

⚛️ Example in React

Using `React.lazy()` and `Suspense`:

```
import React, { Suspense } from 'react';

const About = React.lazy(() ⇒ import('./About'));

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <About />
      </Suspense>
    </div>
  );
}
```

> About.js will only be downloaded when About is rendered.

🖼️ Lazy Loading Images (HTML)

```html
<img src="thumbnail.jpg" loading="lazy" alt="Lazy loaded image">
```

> Native support in modern browsers using the loading="lazy" attribute.

🚗 Route-Based Code Splitting (React Router Example)

```
const Home = React.lazy(() => import('./Home'));
const Profile = React.lazy(() => import('./Profile'));

<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/profile" element={<Profile />} />
</Routes>
```

⚙️ Code Splitting in Webpack

Webpack automatically splits code when using dynamic `import()`:

```
function loadFeature() {
  import('./feature').then(module => {
    module.init();
  });
}
```

This generates a separate chunk like `feature.chunk.js`.

⏱️ Lazy Loading in Vanilla JS

```
button.addEventListener('click', () => {
  import('./modal.js').then(({ showModal }) => {
    showModal();
  });
});
```

🎯 Summary

Technique	Purpose	When Used
Code Splitting	Breaks code into smaller bundles	Build time (e.g., Webpack)
Lazy Loading	Loads code/resources on demand	Runtime (e.g., when needed)

Both techniques are essential for modern, scalable, high-performance web apps.

Let me know if you want examples in frameworks like Vue, Angular, or Next.js!