**Follow:** [**https://github.com/amarkumarram/MicrofrontendTutorial/tree/host-app?tab=readme-ov-file**](https://github.com/amarkumarram/MicrofrontendTutorial/tree/host-app?tab=readme-ov-file) **and Youtube** [**https://www.youtube.com/playlist?list=PL1UWPWOy5UlfLWfSstHBW-iVu6RWJtrG-**](https://www.youtube.com/playlist?list=PL1UWPWOy5UlfLWfSstHBW-iVu6RWJtrG-)

**What is Webpack 5?**

**Webpack 5 is the latest major version of Webpack, released in 2020. It introduces several new features, optimizations, and changes aimed at improving performance, reducing bundle sizes, and enhancing module management.**

**Key Features and Improvements in Webpack 5:**

1. **Module Federation:**
   * **A significant new feature that enables sharing code and dependencies between different applications at runtime.**
   * **Allows multiple projects or microfrontends to share modules without bundling everything together, making dynamic code loading possible.**
2. **Improved Caching:**
   * **Webpack 5 comes with better caching capabilities, making builds faster by using persistent caching of modules.**
3. **Automatic Dependency Management:**
   * **Webpack 5 can automatically manage shared dependencies between different parts of an application, reducing duplicate code.**
4. **Tree Shaking Improvements:**
   * **Enhances the process of removing unused code from the final bundle, leading to smaller bundle sizes.**
5. **Better Asset Management:**
   * **It introduces an improved way to handle assets like images, fonts, and other static files, making asset optimization more efficient.**
6. **WebAssembly (Wasm) Support:**
   * **Webpack 5 improves support for WebAssembly modules, allowing developers to integrate Wasm more easily with JavaScript.**

**In Summary:**

* **Webpack is a tool for bundling and optimizing web applications.**
* **Webpack 5 is the latest version that brings new features like Module Federation for microfrontends, better caching, and performance improvements. It significantly simplifies the sharing of code between different parts of a web application.**

**1. What is a Microfrontend?**

* **Microfrontend** is an architectural style where a large web application is divided into smaller, independent parts (micro-applications or microfrontends), each responsible for a specific feature or section of the user interface.
* Each microfrontend can be developed, tested, and deployed separately, which is similar to the **microservices** approach on the backend, where different services handle different parts of an application.

**2. Why Use Microfrontends?**

Microfrontends address challenges that arise when building large web applications, such as:

* **Scaling Development**: When many developers work on the same codebase, it can become difficult to manage changes. By breaking the app into smaller pieces, teams can work independently.
* **Independent Deployments**: Each microfrontend can be updated or deployed separately without affecting the entire application.
* **Technology Flexibility**: Different teams can use different frameworks or libraries (e.g., React, Angular, Vue) for their parts of the app.
* **Easier Maintenance**: Smaller codebases are easier to maintain, test, and debug.

**3. How Do Microfrontends Work?**

Microfrontends work by splitting the user interface into multiple mini-applications, each responsible for a specific part of the app. Here’s how they typically function:

1. **Each microfrontend is built as a separate app**: It can have its own repository, tech stack, and build process.
2. **The microfrontends are then integrated into a single UI**: This is done using various techniques, such as embedding them into a main application shell or using client-side composition techniques.
3. **Communication and state sharing**: Microfrontends can communicate with each other when necessary, using shared services, events, or custom APIs.

**What is Module Federation?**

**Module Federation** is a feature introduced in **Webpack 5** that allows microfrontends to share code and resources at runtime. It facilitates the dynamic loading of modules between applications without needing to bundle all code together.

**How Module Federation Works in Microfrontends**

1. **Expose and Consume Code Dynamically**: With Module Federation, you can configure one microfrontend to expose components, modules, or services, and another microfrontend can consume these.
2. **Share Dependencies**: Avoids loading multiple versions of the same library by sharing dependencies between microfrontends.
3. **Remote Modules**: Microfrontends can act as "remotes" that provide code, while others act as "hosts" that load and use that code.

**6. How Are Microfrontends and Module Federation Related?**

Module Federation provides the mechanism for implementing a microfrontend architecture:

* **Microfrontends use Module Federation to share components, libraries, or even entire features between different parts of the application.**
* It allows dynamic code loading at runtime, making it possible to integrate microfrontends seamlessly without rebuilding the whole app.

In microfrontend architecture, **monorepo** and **multi-repo** are two different ways of organizing and managing the codebase. They describe how the code for different microfrontends is stored and handled in source control.

**1. Monorepo (Single Repository)**

* **Description**: All microfrontends are stored in a **single repository**, with each microfrontend existing as a separate folder or package. The monorepo contains the entire codebase for the application, including all microfrontends and shared libraries.
* **Advantages**:
  + **Easier Code Sharing**: Shared code and libraries can be easily reused across different microfrontends since everything is in one place.
  + **Consistent Development Practices**: Ensures consistent tooling, build processes, and code standards across all microfrontends.
  + **Simplified Dependency Management**: Managing dependencies is more straightforward since everything is in the same repository.
  + **Atomic Changes**: Changes that affect multiple microfrontends can be made in one commit, reducing coordination overhead.
* **Challenges**:
  + **Scaling the Repository**: As the application grows, the size of the repository can become difficult to manage.
  + **Longer Build Times**: Large monorepos may have longer build and test cycles.
  + **Complex CI/CD Pipelines**: Ensuring the right parts of the code are built and tested can be more complex in a monorepo.

**2. Multi-Repo (Multiple Repositories)**

* **Description**: Each microfrontend is stored in its **own separate repository**. This means there is a different repository for each microfrontend, as well as possibly for shared libraries.
* **Advantages**:
  + **Independent Development and Deployment**: Teams can work independently on different microfrontends, with their own release cycles.
  + **Better Isolation**: Issues in one microfrontend are less likely to affect others since they are developed separately.
  + **Scalable Codebase**: Since each repository is smaller, it is easier to manage as the application scales.
* **Challenges**:
  + **Code Duplication**: Sharing code between different microfrontends is more difficult, leading to potential duplication.
  + **Dependency Management**: Ensuring consistent versions of shared libraries across multiple repositories can be challenging.
  + **Complex Coordination**: Changes that impact multiple microfrontends may require coordination across different repositories, making it harder to maintain consistency.

**How They Relate to Microfrontend Architecture**

* In a **monorepo**, the microfrontends share the same repository, making it easier to share code and maintain consistency. However, build processes may need to be optimized to avoid building everything at once.
* In a **multi-repo**, each microfrontend is isolated, allowing for independent development and deployment, but making it harder to maintain consistency and share code.

**Which Approach to Use?**

* **Monorepo**: Suitable when teams are small, the codebase is relatively simple, or there is a need for tight integration and consistent tooling.
* **Multi-Repo**: Better for large teams working on separate parts of the application, where each microfrontend can have its own lifecycle and tech stack.

In a microfrontend setup, you can still share code using tools like **Module Federation** in **Webpack 5**, whether using a monorepo or a multi-repo structure

**SIMPLE:**

Here’s a simple explanation of **monorepo** and **multi-repo** in the context of microfrontends:

**Monorepo**

* **All microfrontends are stored in a single repository.**
* **Advantages**: Easier to share code and manage dependencies. Consistent development environment.
* **Challenges**: Can become large and harder to manage as the project grows.

**Multi-Repo**

* **Each microfrontend has its own separate repository.**
* **Advantages**: Teams can work independently and deploy separately. Easier to scale.
* **Challenges**: Sharing code is harder. Keeping dependencies in sync can be tricky.

Both approaches can be used in microfrontend architecture, depending on the team's needs and project structure.

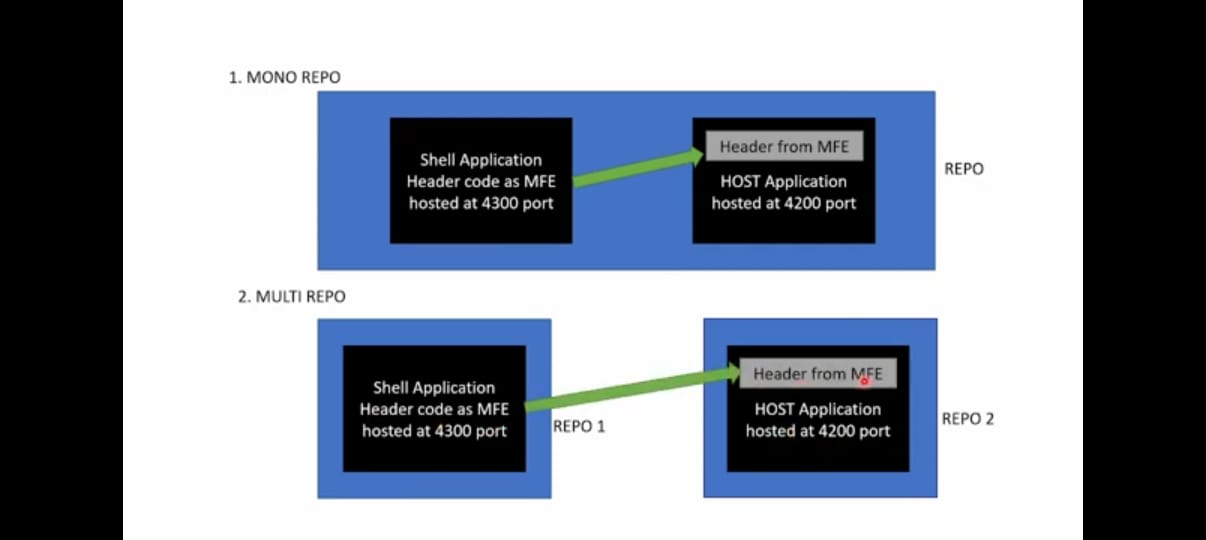
Monorepo and multi-repo describe **how the codebase is managed** in a microfrontend architecture, whereas the different **types of microfrontends** (like component-based, page-based, etc.) describe **how the microfrontends are integrated** into the application.

**How Monorepo/Multi-Repo and Microfrontend Types Differ:**

1. **Monorepo vs. Multi-Repo**:
   * **Monorepo**: All microfrontends are kept in a single repository. This approach is about storing all the code in one place.
   * **Multi-Repo**: Each microfrontend has its own separate repository. This approach is about splitting the code across multiple places for better isolation.
2. **Microfrontend Types**:
   * **Component-Based**: Different microfrontends provide individual components (e.g., buttons, forms).
   * **Page-Based**: Each microfrontend represents an entire page or section.
   * **Route-Based**: Different routes load different microfrontends.
   * **Layout-Based**: Different microfrontends fill different sections of a common layout.

**How They Relate:**

* **Monorepo/Multi-Repo** is about **how the code is stored and managed**.
* **Microfrontend types** are about **how the application is built and integrated**.



**Summary of Commands:**

* **Create workspace**: ng new mono-workspace --createApplication="False" *(It will create a workspace, not an application.)*
* **Navigate to workspace**: cd mono-workspace
* **Generate host app**: ng g application host-app --routing --style=scss *(1st app)*
* **Generate microfrontend app**: ng g application mfe-app --routing --style=scss *(2nd app)*
* **Run app**: ng serve host-app -o *(Use -o to open the app in the default browser.)*
* **Install Webpack**: npm install webpack webpack-cli --save-dev *(Inside workspace)*
* **Add Module Federation**: ng add @angular-architects/module-federation --project host-app --port 4220 *(After running this command, we will get one file webpack.config.js.)*
* **Create component**: ng generate component component-name --project=project-name *(Replace component-name and project-name with your desired names.)*