1. **Shared Module Approach:**

**Description:**

* Host a centralized shared module containing common models, services, and utilities as a separate micro-frontend. Other micro-frontends (e.g., admin and account) consume this module.

**Pros:**

* Centralized ***updates ensure consistency across micro-frontends.***
* ***Easy to*** maintain a single source of truth.

**Cons:**

* ***Increased bundle size:*** The entire shared module is loaded even if only a portion of it is needed.
* ***Potential performance*** hit if the shared module grows too large over time.
* ***Tight coupling*** between micro-frontends and the shared module.

**Use Case:**

* Suitable for ***tightly*** integrated ***teams***/***projects*** where all micro-frontends are deployed and versioned together.

1. **Individual Module Approach:**

**Description:**

* Each micro-frontend ***maintains its own copy*** of the shared models.

**Pros:**

* ***Maximum decoupling***: Each micro-frontend operates independently.
* ***No risk of*** unexpected ***changes*** in shared models breaking unrelated micro-frontends.

**Cons:**

* ***Duplication*** of models increases maintenance effort.
* ***Changes*** in shared models need to be ***manually*** propagated across micro-frontends, leading to inconsistency and potential errors.
* Higher risk ***of version mismatches*** over time.

**Use Case:**

* Suitable for ***teams*** with ***completely*** ***independent*** ***workflows*** or when the likelihood of shared model updates is very low.

1. **Package as External Libraries:**

**Description:**

* Extract shared models into a separate library (e.g., an NPM package). Publish this library to a private NPM registry or host it as part of your monorepo. Each micro-frontend includes the library as a dependency.

**Pros:**

* Centralized management with versioning: Changes in models can be propagated to micro-frontends via controlled version updates.
* Smaller micro-frontend bundles: Only the library parts used by a micro-frontend are included in its build.
* Decoupled deployment: Micro-frontends can update the library independently when needed.
* Encourages modularity and reusability.

**Cons:**

* Requires additional setup for library versioning and publishing.
* Potential for version mismatches if micro-frontends do not upgrade simultaneously (solvable with semantic versioning and communication).

**Use Case:**

* Ideal for teams adopting micro-frontend architecture with shared models that evolve over time. Ensures scalability and maintainability.