

Routing e Axios

REACT AVANZATO



React Routing




React Routing + Axios

- Obiettivi
 - Gestione di **routing**
 - Integrazione di **Axios** per chiamate API
 - Implementazione di **routes** protette con autenticazione



Routing

- **React Router**
 - Definizione di route
 - Elemento **Routes** ed elemento **Route**
 - Navigazione via codice
 - Hook **useNavigate**
- 

Definizione delle Rotte

```
<Routes>  
  <Route path="/" element={<Navigate to="/home" />} />  
  <Route path="/home" element={<Home />} />  
</Routes>
```

Rotte con Parametri

- Gestione dei parametri nelle rotte:
 - Parametri dinamici introdotti da due punti:
 - `:id`, rappresenta un parametro di nome *id*
 - I parametri dinamici sono recuperati attraverso lo hook `useParams`

Fallback Route

- Implementazione del path "*" per gestire, ad esempio, **errori 404**
- Esempio:

```
<Route path="*" element={<NotFound />} />
```

Proteggere le Rotte

- Non esiste un meccanismo di sistema
- Possiamo usare un pattern di decorazione:

```
const ProtectedRoute = ({ element, isAuthenticated }) =>  
  isAuthenticated ? element : <Navigate to="/login" />;
```

- Che usa una logica per presentare un elemento in maniera condizionata
- Nella definizione della rotta va usato il componente **ProtectedRoute** e il componente effettivo va specificato nel parametro element!




Axios





Cos'è Axios

- Libreria per gestire richieste HTTP
 - Supporto per **Promises**
 - Consente l'implementazione di intercettori e configurazioni personalizzate
- 

Axios

Installazione e uso “Statico”

- Installazione

```
npm install axios
```

- Uso dell'oggetto statico della libreria


```
axios.get('https://jsonplaceholder.typicode.com/posts')  
  .then(response => {  
    console.log(response.data); // Stampa i dati ricevuti  
  })  
  .catch(error => {  
    console.error('Errore nella richiesta:', error);  
  });
```

Funzionalità Integrate

- Metodi principali dell'oggetto axios
 - `axios.get(url, config)`
 - Per recuperare dati da un server
 - `axios.post(url, data, config)`
 - Per inviare dati al server
 - `axios.put(url, data, config)`
 - Per aggiornare dati esistenti
 - `axios.delete(url, config)`
 - Per eliminare dati



Caratteristiche

- Supporta la gestione di **header** personalizzati
 - Consente il **parsing** automatico del JSON
 - Implementa la logica di gestione degli errori attraverso un costrutto **try-catch**
- 

Istanze Customizzate

- Creazione di un'istanza di **Axios**

```
const apiClient = axios.create({  
  baseURL: process.env.REACT_APP_API_URL,  
  headers: { 'Content-Type': 'application/json' }  
});
```

- Vantaggi:
 - Consente una configurazione specifica per un determinato contesto
 - **BASE_URL**
 - **headers**

Intercettori

- Intercettori di **request**

```
apiClient.interceptors.request.use(config => {  
  const token = localStorage.getItem('token');  
  if (token) config.headers['Authorization'] = `Bearer ${token}`;  
  return config;  
});
```

- Intercettori di **response**

```
apiClient.interceptors.response.use(  
  response => {  
    // Gestisci qui la risposta (ad esempio, log o trasformazione dei dati)  
    console.log('Response:', response);  
    return response.data; // Restituisce solo i dati  
  },  
  error => {  
    // Gestisci errori (es. visualizza un messaggio o logga l'errore)  
    if (error.response) {  
      console.error('Errore HTTP:', error.response.status, error.response)
```

Intercettori

- In request con gli intercettori interagiamo con la richiesta che il client fa verso il server
 - Ad esempio aggiungiamo header
 - Es. `Authorization=Bearer token`
- In response possiamo trasformare i dati ricevuti o applicare log
- Inoltre possiamo gestire varie tipologie di errori da errori HTTP ad errori di connessione o configurazione



Domande e Discussione



Grazie per l'attenzione!