📌 Week 8 – TypeScript & API-Driven Frontend: Building a Messenger App

FH University of Applied Sciences
TECHNIKUM WIEN

📌 Week 8 – TypeScript & API-Driven Frontend: Building a Messenger App

Learning Objectives

✅ Understand **REST APIs** and how they interact with frontend applications.
✅ Learn **TypeScript interfaces, type safety, and dependency injection** for modular applications.
✅ Explore **modular web application architecture** and **state management**.
✅ Study **best practices in TypeScript API integration** and error handling.
✅ Prepare for **Angular (Week 10)** by reinforcing **component-based thinking and modularization**.

🌍 Understanding REST APIs in Modern Web Applications

1️⃣ What is an API?

- An **Application Programming Interface (API)** allows applications to communicate.
- **REST APIs (Representational State Transfer)** provide structured data exchange using HTTP.
- APIs **abstract the complexity** of the backend, enabling **scalability** and **flexibility**.

2️⃣ Why Do We Need APIs?

- 🔹 **Decoupling frontend and backend**
- 🔹 **Enabling mobile and web applications to use the same services**
- 🔹 **Providing reusable data structures across multiple platforms**

✅ APIs **define a contract** between services, making software **more modular and maintainable**.

🔄 HTTP Methods in REST APIs

Method	Description	Example in a Chat App
`GET`	Fetch data from the server	Retrieve chat messages
`POST`	Send new data	Send a new message
`PUT`	Update existing data	Edit a message
`DELETE`	Remove data	Delete a message

💡 **A frontend chat app fetches messages using `GET` and sends messages using `POST`.**

📡 How Frontend Applications Communicate with APIs

1️⃣ The Request-Response Cycle

- The **client (frontend)** sends an HTTP request to the API.
- The **server (backend)** processes the request and returns a response.
- The **frontend processes the response** and updates the UI.

✅ **This cycle is repeated every time the user interacts with data.**

🛠️ JSON – The Universal Data Format

- APIs **typically send and receive data in JSON (JavaScript Object Notation)**.
- JSON is a **lightweight, human-readable format**.

📌 **Example JSON API Response for a Chat App**

```
[
  {
    "id": 1,
    "sender": "Alice",
    "content": "Hello, how are you?",
    "timestamp": "2024-02-06T12:00:00Z"
  }
]
```

💡 JSON **uses key-value pairs** and supports nested structures.

✅ **TypeScript uses interfaces to enforce structure** for API data.

📜 TypeScript & API Data

1️⃣ Why Use TypeScript with APIs?

- TypeScript **ensures that API responses match expected data types**.
- Prevents **runtime errors caused by unexpected data**.
- **Improves maintainability and debugging**.

✅ **TypeScript adds structure to API responses through interfaces**.

2️⃣ TypeScript Interface for a Chat Message

```
interface Message {
    id: number;
    sender: string;
    content: string;
    timestamp: Date;
}
```

✅ This interface **ensures consistency** when handling API responses.

🧩 Modular Application Architecture in TypeScript

1️⃣ Why Modular Code?

- **Tightly coupled code leads to maintenance issues.**
- **Code reusability and separation of concerns make scaling easier.**
- **Each module has a clear responsibility.**

✅ **Modular design is the foundation of modern frontend frameworks like Angular.**

2️⃣ Messenger App Modular Design

1️⃣ **API Service** → Handles API communication.
2️⃣ **User Interface (UI)** → Displays messages dynamically.
3️⃣ **State Management** → Tracks user sessions and chat history.

✅ **Each module is independent and can be replaced or extended.**

📡 Handling API Calls in TypeScript

1️⃣ Making a `GET` Request

- The frontend fetches messages from the API.
- JSON data is **converted into TypeScript objects**.

✅ Fetching Data from an API

```
async function fetchMessages(): Promise<Message[]> {
    const response = await fetch("https://api.example.com/messages");
    return await response.json();
}
```

✅ Type safety ensures correct structure.

📡 Sending Data to an API (`POST`)

1️⃣ Why Use POST Requests?

- The frontend **sends new messages to the server**.
- The server **stores them and returns confirmation**.

✅ Example API Request

```
async function sendMessage(msg: Message): Promise<void> {
    await fetch("https://api.example.com/messages", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(msg),
    });
}
```

💡 **Always specify `Content-Type: application/json` for API requests.**

⚠️ Handling API Errors

1️⃣ Common API Issues

1️⃣ **Network failures** → The server is unreachable.
2️⃣ **Invalid data** → The API rejects incorrect input.
3️⃣ **Unauthorized access** → The user lacks permission.

✅ **Use `try/catch` to handle errors properly.**

📌 **Example**

```
try {
    const response = await fetch("https://api.example.com/messages");
    if (!response.ok) throw new Error("Failed to fetch messages");
} catch (error) {
    console.error("API Error:", error);
}
```

✅ **Never assume an API request will always succeed.**

🎭 The Role of State Management

1️⃣ Why Do We Need State Management?

- In **single-page applications (SPAs)**, the **frontend must track and manage data** efficiently.
- State management **ensures that the UI stays synchronized with data**.

2️⃣ Types of State Management

1️⃣ **Component State** → Data exists within the UI components.
2️⃣ **Application State** → A central store manages all data.
3️⃣ **Session Storage** → Data persists even after a page reload.

✅ **State management avoids unnecessary API calls and improves performance**.

📌 Hands-On Exercise: Build Your Messenger Frontend

🚀 What You Will Do

✅ **Fetch messages from an API & display them.**
✅ **Allow users to send new messages.**
✅ **Organize the project using TypeScript modules.**

📁 Project Structure

```
/chat-app
├── /src
│   ├── ApiService.ts      // Handles API calls
│   ├── ChatUI.ts          // Manages the frontend UI
│   ├── StateManager.ts    // Manages chat state
├── /public
│   ├── index.html         // The main HTML file
│   ├── styles.css         // Chat app styling
```

✅ **You will implement a simple UI connected to a REST API.**

💡 What’s Next?

🟢 **Week 9:** UI Enhancements & Advanced State Management.
🟢 **Week 10:** Introduction to **Angular & Frontend Frameworks**.

🚀 **Get ready for Angular by understanding modular TypeScript design.**

🙋 Questions?

📩 Feel free to ask!
🎯 **Good luck with your exercise!** 🚀

Why this version?

- **More theory** on REST APIs, modular design, and state management.
- **Fewer coding examples**, but **small, meaningful snippets** to reinforce concepts.
- **More structured explanations** preparing students for **Angular**.