📌 Week 9 – Advanced TypeScript & Real-Time Web Applications

FH University of Applied Sciences TECHNIKUM WIEN

📌 Week 9 – Advanced TypeScript & Real-Time Web Applications

Learning Objectives

✅ Understand **advanced TypeScript concepts** (Generics, Type Inference, Utility Types).
✅ Learn **best practices for API communication** (caching, retries, performance optimization).
✅ Explore **real-time communication with WebSockets**.
✅ Study **state management patterns** and how they scale in modern web applications.
✅ **Compare JavaScript vs TypeScript vs Angular**, preparing for **Week 10**.

🌍 Why Do We Need Advanced TypeScript?

1️⃣ TypeScript vs JavaScript

Feature	JavaScript	TypeScript
Static Typing	❌ No	✅ Yes
Code Scalability	❌ Harder	✅ Easier
Debugging	❌ More Errors	✅ Compile-Time Checks
Maintainability	❌ Prone to Bugs	✅ Clear Contracts

💡 **TypeScript reduces runtime errors**, making applications **safer and more scalable**.

🛠️ The Role of APIs in Web Applications

1️⃣ How Do Frontend & Backend Communicate?

- Web applications rely on **APIs** to exchange data.
- **REST APIs** are the standard for structured communication.
- **WebSockets** enable real-time updates, crucial for messaging apps.

✅ **A Messenger App needs both REST (for fetching old messages) and WebSockets (for new messages).**

🔄 The HTTP Request-Response Cycle

Step	Description
1️⃣ **Client Request**	The browser sends a request to an API.
2️⃣ **Server Processing**	The backend processes the request and fetches data.
3️⃣ **Server Response**	The API returns a structured response (usually JSON).
4️⃣ **Frontend Updates**	The UI updates dynamically based on the data received.

💡 **Optimizing API requests reduces network load and improves performance.**

📡 API Communication: REST vs WebSockets

1️⃣ REST APIs: Request-Based Communication

✅ **Best for fetching data periodically.**
❌ **Not ideal for real-time applications.**

📌 **Example of a REST API Response**

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

📡 WebSockets: Real-Time Communication

2️⃣ WebSockets: Event-Based Communication

✅ **Best for real-time updates (e.g., chat messages).**
❌ **Requires persistent connections, increasing server load.**

📌 Example WebSocket Communication

```
const socket = new WebSocket("wss://chat.example.com");

socket.onmessage = (event) => {
    console.log("New Message:", event.data);
};
```

💡 **WebSockets allow bidirectional, real-time data flow.**

📜 TypeScript & API Data

1️⃣ Why Use TypeScript for API Communication?

- **Enforces data consistency**.
- **Prevents runtime errors** by **defining expected structures**.

📌 Example: TypeScript Interface for Chat Messages

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

✅ **Ensures API responses match expected types.**

🧩 TypeScript Generics: Why Do We Need Them?

1️⃣ Reusable Code Across Different Data Types

- Avoids code duplication.
- Works with **any data type while maintaining type safety**.

📌 **Example: Generic Function**

```
function identity<T>(value: T): T {
    return value;
}
```

💡 **Used for utility functions, APIs, and modular components.**

🔄 Optimizing API Requests

1️⃣ Performance Challenges

Problem	Solution
Too many requests	Use caching & throttling
Slow API responses	Implement retries
High data usage	Use efficient data structures

📌 Example: Caching API Responses

```
class ApiService {
    private cache: { [key: string]: any } = {};

    async fetchData(url: string): Promise<any> {
        if (this.cache[url]) return this.cache[url];

        const response = await fetch(url);
        const data = await response.json();
        this.cache[url] = data;
        return data;
    }
}
```

✅ **Reduces unnecessary API calls, improving performance.**

🧠 State Management in Web Applications

1️⃣ Why Do We Need State Management?

✅ Keeps **UI synchronized** with data.
✅ Avoids **unnecessary API calls**.
✅ Enables **scalability and modularity**.

💡 **Session storage, local storage, and Redux-like patterns manage state efficiently.**

🔗 Preparing for Angular

1️⃣ Why Angular?

Feature	JavaScript	TypeScript	Angular
Component-Based	❌ No	✅ Yes	✅ Yes
Dependency Injection	❌ No	✅ Yes	✅ Yes
Two-Way Binding	❌ No	❌ No	✅ Yes
Built-in State Management	❌ No	❌ No	✅ Yes

✅ **Angular extends TypeScript concepts into a powerful frontend framework.**
🚀 **Week 10 introduces Angular!**

🎭 Final Hands-On Task: Messenger App Completion

Students Must:

✅ **Implement API optimizations (caching, retries).**
✅ **Integrate WebSockets for real-time updates.**
✅ **Store session data to persist user logins.**

📌 **The final TypeScript project before moving to Angular!**

💡 What's Next?

🟢 **Week 10:** Introduction to **Angular & Frontend Frameworks**.
🟢 **Prepare by reviewing modular TypeScript concepts.**

🚀 **Get ready for Angular by mastering TypeScript structure!**

🙋 Questions?

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