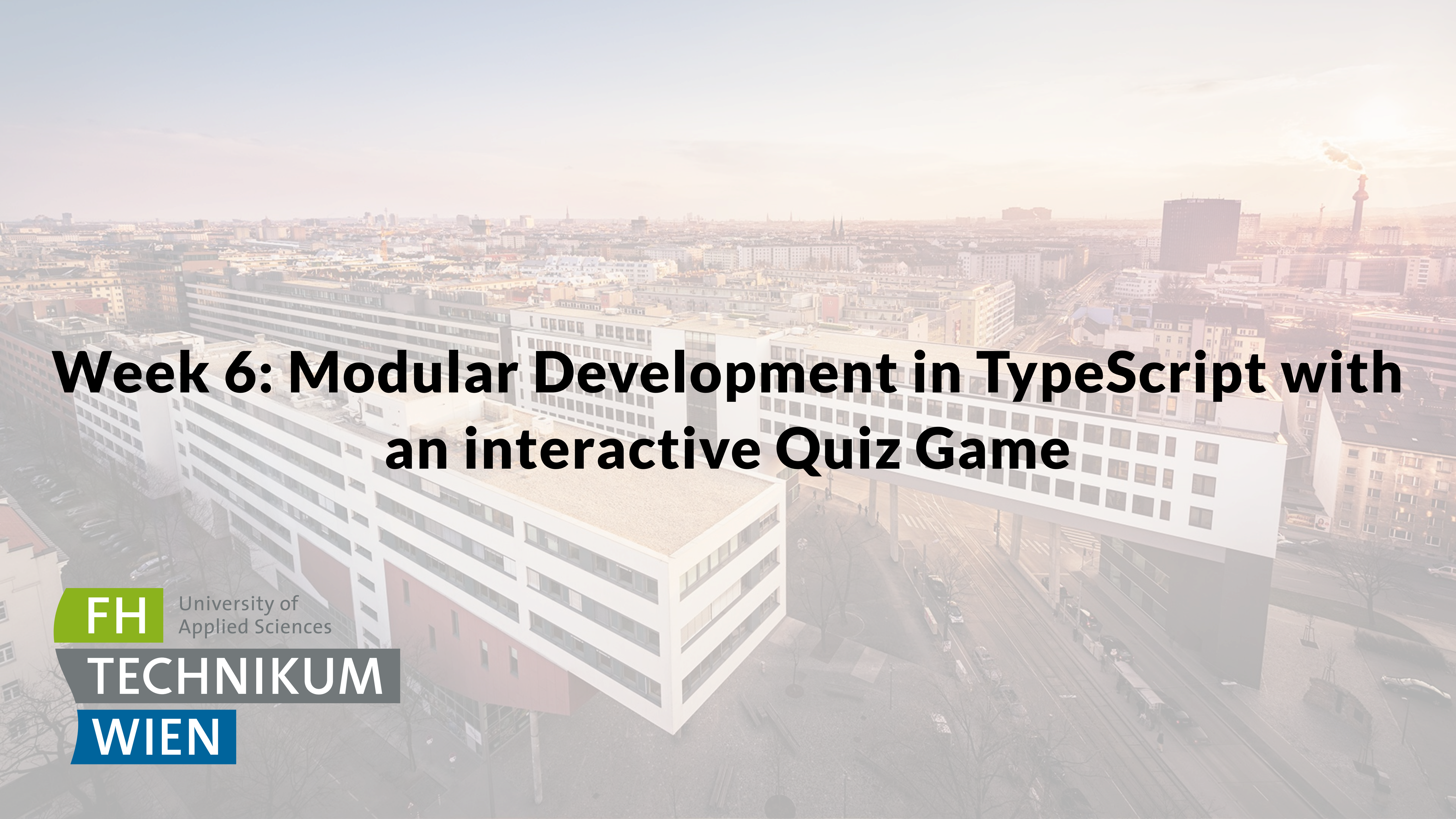

Week 6: Modular Development in TypeScript with an interactive Quiz Game

FH University of Applied Sciences TECHNIKUM WIEN

Week 6: Modular Development in TypeScript with an interactive Quiz Game

Topics for This Week

1. Introduction to Modular Development in TypeScript
2. Core Quiz application architecture
3. Implementing a Quiz Game with TypeScript
4. Extensibility for future features

1. Introduction to modular development

What is modular development?

- Breaking down an application into smaller, reusable components.
- Each module has a clear responsibility and interacts with others through well-defined interfaces.

Benefits:

- **Scalability**: Easily add new features.
- **Reusability**: Use modules in multiple applications.
- **Maintainability**: Easier debugging and testing.

Example: Modular architecture for a Quiz Gool

Components:

1. **QuizManager**: Logic for managing the quiz flow (loading and preparing questions).
2. **QuestionRenderer**: Handles rendering questions and capturing user input.
3. **ScoreManager**: Tracks and displays user scores.

Data Flow:

1. Questions are loaded by `QuizManager` from a JSON file where the questions are stored.
2. `QuestionRenderer` displays questions to the user.
3. User answers are processed, and scores are updated by `ScoreManager`.

2. Core Quiz application architecture

Requirements for the Quiz Game

1. Load and display multiple-choice questions which are stored in a JSON file.
2. Create a user which is playing the game and display 5 questions for the user.
3. Every user should get different questions but always the same number of the difficulty (easy, medium, hard).
4. Capture user answers and track the points the user gets, depending on the difficulty of the question.
5. Track and display scores dynamically after finishing a round of 5 questions.
6. Put the result into a leaderboard and line up the players due to the reached points.

High-Level architecture

Separation of concerns:

1. **Frontend**:
   - Dynamic UI rendering with HTML, CSS, and TypeScript.
   - Responsive design for mobile and desktop.
2. **Backend (Future Weeks)**:
   - RESTful APIs for quiz data management.
   - User authentication and score persistence.

Modular design example

QuizManager Module:

Manages the quiz flow:

- Randomly loads questions from an existing JSON file and track progress.
- Submit and validate answers.

Interface for Questions:

```typescript
export interface Question {
  category: string;
  question: string;
  options: (string | number)[];
  answer: string | number;
  difficulty: 'easy' | 'medium' | 'hard';
}
```

Modular design example (Cont.)

QuestionRenderer Module:

- Dynamically renders questions and options.
- Captures user input.
- Fills up the leaderboard.

ScoreManager Module:

- Tracks scores based on the difficulty of the question.
- Displays scores dynamically after finishing a round of 5 questions.

3. Implementing a Quiz Game with TypeScript

Step 1: Define the project structure

```
/quiz-game
├── /src
│   ├── /modules
│   |   ├── questions.ts    // Module that handles the questions
│   |   ├── scoring.ts      // Module that handles the scoring
│   |   ├── ui.ts           // Module that handles the user interface
│   ├── main.ts             // Entry point
├── index.html              // Main HTML file
├── questions.json          // Questions collection file
├── styles.css              // Styles for the quiz
├── tsconfig.json           // TypeScript configuration file
```

Step 2: Implement Core Modules

questions.ts

Manages quiz data and logic:

```
export async function fetchQuestions(): Promise<Question[]> {
  const response = await fetch('./questions.json');
  ...
}

function shuffleArray(array: any[]): any[] {
  ...
  return array;
}
```

ui.ts

Dynamically renders questions:

```typescript
import { fetchQuestions, Question } from './questions.js';
import { calculateScore } from './scoring.js';

export function displayQuestion(index: number): void {
    ...
}

function handleAnswer(index: number, selectedOption: string | number): void {
    ...
}

function displayScore(categoryScores: { [category: string]: { correct: number; total: number; points: number; } }, totalScore: number, totalPoints: number, maxPoints: number): void {
    ...
}
```

4. Extensibility for future features

Preparing for future extensions

1. **Real-Time Multiplayer**:
    - Add WebSocket integration and manage multiple users' responses.
2. **Advanced Quiz Configurations**:
    - Allow quiz creators to add custom questions and implement a backend API for question storage.
3. **Improved Scoring System**:
    - Add time-based scoring and save scores persistently using local storage or a database.

Weekly exercise

Task: Build a modular Quiz application

1. Implement core modules:
   - `QuizManager`
   - `QuestionRenderer`
   - `ScoreManager`
2. Use mock data to display and validate quiz questions.
3. Create a responsive and dynamic UI.

Summary and Q&A

- **Key Concepts**:
  - Modular application design.
  - Core TypeScript principles.
  - Extensible architecture.

Questions?