# Aufgabe: Mono-Repository mit Multi-Root Workspaces einrichten

## Lernziele

Nach dieser Aufgabe können Sie:

- Ein Mono-Repository für ein Fullstack-Projekt strukturieren
- Multi-Root Workspace-Dateien für verschiedene Entwicklungsszenarien erstellen
- Globale und workspace-spezifische VS Code-Einstellungen konfigurieren
- Extensions gezielt für unterschiedliche Technologien definieren
- Ein professionelles Entwicklungsumfeld für Teamarbeit aufsetzen
- GitHub-Issue-Templates (Issue Forms in YAML) erstellen, anpassen und korrekt platzieren (`.github/ISSUE_TEMPLATE/`)
- Pull-Request-Templates projektspezifisch strukturieren (Checklisten, betroffene Komponenten, Tests, Breaking Changes) und nutzen (`.github/PULL_REQUEST_TEMPLATE/`)
- Standardisierte Workflows mit Templates unterstützen (Labels, Assignees, Meilensteine/Projects über Frontmatter konfigurieren)
- Den Zweck und die Unterschiede zwischen klassischen Issue-Templates und Issue Forms verstehen und anwenden

## Kontext

Sie starten ein neues Fullstack-Projekt mit drei Komponenten:

- **Mobile App** (z.B. Flutter)
- **Web-Admin-Oberfläche** (z.B. React + TypeScript + Vite)
- **Backend-Server** (Spring Boot)

Alle drei Komponenten sollen in einem gemeinsamen Git-Repository verwaltet werden, um Code-Sharing zu vereinfachen und eine koordinierte Entwicklung zu ermöglichen.

## Voraussetzungen

- Git ist installiert
- VS Code ist installiert
- Grundkenntnisse in der Arbeit mit VS Code
- Kenntnisse über die Konzepte von Mono-Repositories und Multi-Root Workspaces (*siehe bereitgestelltes Scriptum*)

## Aufgabenstellung

### Teil 1: Repository-Struktur erstellen

Erstellen Sie ein neues Git-Repository mit folgender Ordnerstruktur:

```
your-project/
├── .github/                  # GitHub-Konfiguration
│   ├── ISSUE_TEMPLATE/
│   │   ├── 01_user_story.yml
│   │   ├── 02_bug_report.yml
│   │   ├── 03_enhancement.yml
│   │   └── 04_refactor.yml
│   └── PULL_REQUEST_TEMPLATE/
│       ├── mobile_pr_template.md
│       ├── web_pr_template.md
│       └── server_pr_template.md
├── mobile/                   # Flutter-App (noch leer)
├── admin-web/                # React-Web-Admin (noch leer)
├── server/                   # Spring Boot Backend (noch leer)
├── shared-resources/         # Gemeinsame Ressourcen
│   ├── documentation/
│   │   └── README.md
│   └── design-tokens/
│       └── README.md
├── docs/
│   └── project-overview.md
├── .gitignore
└── README.md
```

**Anforderungen:**

**Repository initialisieren**

- Erstellen Sie einen neuen Ordner `[your-project]` (z.B. `eco-track`)
- Initialisieren Sie ein Git-Repository
- Erstellen Sie einen initialen Commit

**Ordnerstruktur anlegen**

- Erstellen Sie alle oben genannten Ordner
- Fügen Sie in jeden leeren Ordner eine `.gitkeep`-Datei ein (damit Git leere Ordner trackt)

**README.md erstellen**

- Projektbeschreibung
- Komponenten-Übersicht (Mobile, Web, Server)
- Kurze Anleitung zum Öffnen der Workspaces

**.gitignore konfigurieren**

- Ignorieren Sie Node.js-spezifische Dateien
- Ignorieren Sie Flutter-spezifische Dateien
- Ignorieren Sie Java/Maven-spezifische Dateien
- Ignorieren Sie IDE-spezifische Dateien (außer `.vscode/`)

**Beispiel `.gitignore`:**

```
# Node / Web
dist/
build/
...

# Flutter / Dart
mobile/build/
mobile/ios/Pods/
...

# Java / Maven
target/
.mvn/wrapper/maven-wrapper.jar

# OS
.DS_Store
Thumbs.db
...
```

---

## Teil 2: GitHub Issue & Pull Request Templates erstellen

Erstellen Sie GitHub-Templates für Issues und Pull Requests, um eine standardisierte Zusammenarbeit im Team zu ermöglichen.

### 2.1 Issue Templates (YAML-Format)

Erstellen Sie den Ordner `.github/ISSUE_TEMPLATE/` und darin vier Issue-Templates im YAML-Format:

- `user-template.yaml`
- `refactor-template.yaml`
- `enhancement-template.yaml`
- `bugfix-template.yaml`

### 2.2 Pull Request Templates

Erstellen Sie den Ordner `.github/PULL_REQUEST_TEMPLATE/` und darin drei projektspezifische PR-Templates:

- `server_pr-template.md`
- `mobile_pr-template.md`
- `web-app_pr-template.md`

## Teil 3: Globale VS Code-Konfiguration erstellen

Erstellen Sie im Root-Verzeichnis einen `.vscode/`-Ordner mit globalen Konfigurationsdateien.

### 3.1 `settings.json` - Globale Einstellungen

Erstellen Sie `.vscode/settings.json` mit folgenden Einstellungen:

**Anforderungen:**

- Automatisches Formatieren beim Speichern aktivieren
- Ausblenden von Build-Artefakten (z.B. `build/`, `dist/`, `target/`)
- Dart-spezifische Analyse-Ausschlüsse für Build-Ordner
- Java Build-Konfiguration auf "interactive" setzen
- ESLint Flat Config aktivieren

**Beispiel-Struktur:**

```
{
  "files.exclude": {
    "**/.git": true,
    "**/node_modules": true,
    // Weitere Excludes...
  },
  "editor.formatOnSave": true,
  // Weitere Einstellungen...
}
```

### 3.2 `extensions.json` - Empfohlene Extensions

Erstellen Sie `.vscode/extensions.json` mit Extensions für **alle** Technologien (z.B. `github.vscode-github-actions`)

**Beispiel-Struktur:**

```
{
  "recommendations": [
    "github.vscode-github-actions",
    // Weitere Extensions...
  ]
}
```

### 3.3 `tasks.json` - Projektübergreifende Tasks

Erstellen Sie `.vscode/tasks.json` mit Tasks für alle drei Komponenten:

**Anforderungen:**

- **Mobile Tasks**: `mobile: pub get`, `mobile: run`, `mobile: test`
- **Web Tasks**: `web: install`, `web: dev`, `web: build`
- **Server Tasks**: `server: mvn verify`, `server: run`
- Alle Tasks müssen das korrekte `cwd` (Working Directory) setzen

**Beispiel-Task:**

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "mobile: pub get",
      "type": "shell",
      "command": "flutter pub get",
      "options": { "cwd": "${workspaceFolder}/mobile" },
      "problemMatcher": []
    }
    // Weitere Tasks...
  ]
}
```

---

## Teil 4: Workspace-Dateien erstellen

Erstellen Sie vier verschiedene Workspace-Dateien im Root-Verzeichnis.

### 4.1 Full Workspace

**Datei:** `myproject-full.code-workspace`

**Anforderungen:**

- Öffnet alle vier Ordner: Root (`.`), `mobile`, `admin-web`, `server`
- Blendet Build-Artefakte aus
- Dokumentieren Sie im Kommentar, wofür dieser Workspace gedacht ist

**Beispiel:**

```
{
  "folders": [
    { "path": "." },
    { "path": "mobile" },
    { "path": "admin-web" },
    { "path": "server" }
  ],
  "settings": {
    "files.exclude": {
      // Excludes definieren...
    }
  }
}
```

### 4.2 Mobile Workspace

**Datei:** `myproject-mobile.code-workspace`

**Anforderungen:**

- Öffnet nur den `mobile/`-Ordner
- Blendet andere Projekt-Ordner aus (`../admin-web`, `../server`, `../infra`)
- Definiert Flutter/Dart-spezifische Einstellungen:
  - `dart.flutterSdkPath` (falls FVM verwendet wird, optional)
- Empfiehlt nur Dart/Flutter-Extensions:
  - `Dart-Code.dart-code`
  - `Dart-Code.flutter`
  - ...

#### 4.3 Web Workspace

**Datei:** `myproject-web.code-workspace`

**Anforderungen:**

- Öffnet nur den `admin-web/`-Ordner
- Blendet andere Projekt-Ordner aus
- Definiert TypeScript/React-spezifische Einstellungen:
  - `editor.defaultFormatter: "esbenp.prettier-vscode"`
  - ...
- Empfiehlt nur Web/TypeScript-Extensions:
  - `esbenp.prettier-vscode`
  - ...

#### 4.4 Server Workspace

**Datei:** `myproject-server.code-workspace`

**Anforderungen:**

- Öffnet nur den `server/`-Ordner
- Blendet andere Projekt-Ordner aus
- Definiert Java/Spring-spezifische Einstellungen:
  - `java.saveActions.organizeImports: true`
  - ...
- Empfiehlt nur Java/Spring-Extensions:
  - `vscjava.vscode-maven`
  - ...

---

## Teil 5: Demo-Applikationen erstellen

Für jede der drei Komponenten muss eine lauffähige Demo-App erstellt werden, die mit den definierten Tasks gestartet werden kann.

**Anforderungen:**

#### 5.1 Mobile Demo-App (Flutter)

- Erstellen Sie ein neues Flutter-Projekt im `mobile/`-Ordner
- Erstellen sie die Default - Demo App für Flutter (Counter-App mit Floating Action Button)
- Stellen Sie sicher, dass der Task `mobile: run` die App erfolgreich startet
- Der Task `mobile: pub get` muss alle Dependencies installieren

#### 5.2 Web Demo-App (React + TypeScript + Vite)

- Erstellen Sie ein neues Vite-Projekt mit React und TypeScript im `admin-web/`-Ordner
- Die App soll eine einfache Startseite mit "Admin Dashboard" anzeigen
- Stellen Sie sicher, dass der Task `web: dev` den Dev-Server startet
- Der Task `web: install` muss alle Dependencies installieren
- Der Task `web: build` muss ein Production-Build erstellen

#### 5.3 Server Demo-App (Spring Boot)

- Erstellen Sie ein neues Spring Boot-Projekt im `server/`-Ordner
- Der Server soll einen einfachen REST-Endpoint `/api/health` bereitstellen, der `{"status": "ok"}` zurückgibt
- Stellen Sie sicher, dass der Task `server: run` den Server erfolgreich startet
- Der Task `server: mvn verify` muss erfolgreich durchlaufen

**Hinweis:** Sie können den Spring Initializr verwenden oder die Spring Boot Extension in VS Code nutzen.

#### 5.4 Validierung der Demo-Apps

- Öffnen Sie den Full Workspace
- Führen Sie für jede Komponente die entsprechenden Tasks aus:
  - `mobile: pub get` → `mobile: run`
  - `web: install` → `web: dev`
  - `server: mvn verify` → `server: run`
- Testen Sie, ob alle Apps erfolgreich starten
- Dokumentieren Sie Probleme und deren Lösungen in `docs/project-overview.md`

---

## Teil 6: Dokumentation

Erstellen Sie in `docs/project-overview.md` eine Dokumentation mit folgenden Inhalten:

**Anforderungen:**

**Projektstruktur**

- Beschreiben Sie den Aufbau des Mono-Repositories
- Erklären Sie den Zweck jedes Hauptordners

**Workspace-Guide**

- Beschreiben Sie die vier Workspace-Dateien
- Erklären Sie, wann welcher Workspace verwendet werden soll
- Fügen Sie Befehle zum Öffnen hinzu (`code myproject-mobile.code-workspace`)

**GitHub Templates**

- Erklären Sie die vier Issue-Template-Typen und deren Verwendung
- Beschreiben Sie die drei PR-Templates und wann sie verwendet werden
- Fügen Sie Beispiele für Issue- und PR-Erstellung hinzu

**Development Workflow**

- Beschreiben Sie, wie Tasks ausgeführt werden
- Erklären Sie den Vorteil von Focused Workspaces für die Performance
- Geben Sie Tipps zur Extension-Verwaltung

**Beispiel-Struktur:**

```
# MyProject - Entwicklerdokumentation

## Projektstruktur

Das Projekt ist als Mono-Repository organisiert:
- `mobile/`: Flutter-App für iOS/Android
- `admin-web/`: React-Webanwendung
- `server/`: Spring Boot Backend
- `shared-resources/`: Gemeinsame Ressourcen

## Workspace-Übersicht

### Full Workspace (`myproject-full.code-workspace`)
**Verwendung:** Cross-Platform-Entwicklung, API-Änderungen
...

[weitere Workspaces]

## Entwicklungsworkflow

### Tasks ausführen
1. `Ctrl+Shift+P` → "Tasks: Run Task"
2. Task auswählen (z.B. `mobile: pub get`)
...
```

---

## Teil 7: Testing & Validierung

**Anforderungen:**

**Git-Historie überprüfen**

- Führen Sie `git log --oneline` aus
- Stellen Sie sicher, dass alle Dateien committed sind

**GitHub Templates testen**

- Erstellen Sie ein Test-Issue mit jedem Template-Typ
- Überprüfen Sie, ob alle Felder korrekt angezeigt werden
- Testen Sie die Labels und Validierungen

**Workspaces testen**

- Öffnen Sie jeden Workspace einzeln in VS Code
- Überprüfen Sie, ob die richtigen Ordner sichtbar sind
- Überprüfen Sie, ob die Extension-Empfehlungen angezeigt werden

**Tasks testen**

- Öffnen Sie den Full Workspace
- Führen Sie alle Tasks mindestens einmal aus:
  - `mobile: pub get`, `mobile: run`
  - `web: install`, `web: dev`
  - `server: mvn verify`, `server: run`
- Überprüfen Sie, ob die Demo-Apps erfolgreich starten
- Dokumentieren Sie eventuelle Probleme

**Demo-Apps validieren**

- Mobile App: Läuft die App auf einem Emulator/Simulator?
- Web App: Öffnet sich die App im Browser unter `http://localhost:5173` (oder ähnlich)?
- Server: Gibt der Endpoint `/api/health` die erwartete Antwort zurück?

---

## Abgabe

- **Zugriff** auf Repository für Lehrer einrichten

---

## Hilfreiche Ressourcen

- Multi-root Workspaces - VS Code Dokumentation
- Kapitel 2.3 des Lernskripts: "Mono-Repository und Multi-Root Workspaces"
- VS Code Tasks Dokumentation
- VS Code Extension Marketplace
- GitHub Issue Forms Documentation
- GitHub Pull Request Templates

---

## Tipps für die Bearbeitung

- **Schritt für Schritt vorgehen:** Beginnen Sie mit der Ordnerstruktur, dann globale Konfiguration, dann Workspaces.
- **Testen Sie zwischendurch:** Öffnen Sie Workspaces bereits während der Erstellung, um Fehler früh zu erkennen.

- **JSON-Syntax beachten:** Verwenden Sie einen JSON-Validator oder VS Code's eingebaute Syntaxprüfung.
- **Kommentare nutzen:** JSON erlaubt in VS Code Kommentare – nutzen Sie diese zur Dokumentation!
- **Extension-IDs finden:** Öffnen Sie eine Extension im Marketplace und kopieren Sie die ID aus der URL oder dem Detailbereich.
- **Relative Pfade in Workspaces:** Bei Focused Workspaces mit `"path": "mobile"` wird `mobile/` als Root betrachtet – daher `../admin-web` für Excludes.

---

## Erweiterte Herausforderung (Optional)

Erstellen Sie zusätzlich eine `launch.json` mit Debug-Konfigurationen:

- **Flutter Debug**: Startet die Mobile App
- **Vite Dev Server**: Startet die Web-App
- **Spring Boot Debug**: Startet den Server
- **Compound Configuration**: Startet Web + Server gleichzeitig

**Hinweis:** Diese Konfiguration wird erst funktionieren, wenn die Projekte tatsächlich erstellt sind.