Setup und Start

Das Projekt benötigt zur Ausführung eine JDK11. Wir empfehlen als Entwicklungsumgebung eine aktuelle Version von IntelliJ. Weitere lokale Installationen sind nicht notwendig.

Lombok

In der verwendeten IDE muss noch Lombok installiert und ggf. konfiguriert werden. Hinweise zur Installation bietet zum Beispiel https://www.baeldung.com/lombok-ide.

Lösung der Aufgaben

Grundsätzlich kann das Projekt gestartet werden, zur Lösung der Aufgaben ist es aber ausreichend, die Unit-Tests laufen zu lassen. Dies kann aus der IDE geschehen oder per Maven auf der Kommandozeile:

```
./mvnw clean test
```

bzw. auf Windows Systemen

```
./mvnw.cmd clean install
```

Aufgabe 1: Filtern der Personenliste

In der Klasse `PersonService` sind mehrere Methoden definiert, aber nicht fertig implementiert. Bitte ergänze eine Implementierung, so dass die Tests in der Klasse `PersonServiceTest` erfolgreich durchlaufen.

Die Filter können auf Datenbankebene oder durch Java-Sprachkonstrukte umgesetzt werden. Es können beliebige Hilfsklassen und -methoden erstellt werden.

Aufgabe 2: Synchronisation Urlaubskalender

In dieser Aufgabe sollen zwei externe Systeme miteinander synchronisiert werden: In einem HR-System sind Urlaube und Urlaubsanträge gespeichert. Außerdem gibt es ein Projekt-Planungssystem, in das für jeden beantragten und genehmigten Urlaub eine Abwesenheit eingetragen werden soll. So wird verhindert, dass Personen für Projekte verplant werden, obwohl sie im Urlaub sind.

Es werden an dieser Stelle keine realen Fremdsysteme angebunden, die Systeme sind durch Spring Komponenten gekapselt und würden im Realbetrieb die externe Schnittstelle aufrufen.

Der Unit-Test `SyncServiceTest` definiert Tests, um die Funktionalität des Abgleichs zwischen den Systemen zu überdecken.

Der tatsächliche Abgleich findet in der Klasse `SyncService` statt. Die Methode `makeChanges` erhält die Liste der vom HR-System gelesenen Urlaube und hat die Aufgabe, die notwendigen Änderungen im Planungssystem durchzuführen. Die entsprechende Schnittstelle für das Planungssystem steht in der Klass e in der Variable `planningSystem` zur Verfügung.

Bitte implementiere die Aktualisierung des Planungssystems mit folgenden Restriktionen:

- Es werden von beiden Systemen jeweils alle zur Verfügung stehenden Daten geliefert, nicht nur ein bestimmter Zeitraum.
- Urlaube von Personen, die nicht im Planungssystem erfasst sind, werden ignoriert.
- Der Abgleich zwischen Urlaub und Person läuft über die employeeId des HR-Systems mit der ID der Person im Planungssystem.
- Die Daten müssen nicht validiert werden, es kann davon ausgegangen werden, dass die Systeme nur valide Daten liefern (z. B. Start des Urlaubs immer kleiner als Ende).
- Abgelehnte Urlaube werden aus dem Planungssystem gelöscht/nicht importiert.

- Da das Planungssystem bei jeder Änderung eine Mail verschickt, sollen so wenig Änderungen an das Planungssystem geschickt werden, wie möglich. D. h. es dürfen nicht einfach alle alten Abwesenheiten aus dem Projekt-Planungssystem gelöscht und neu angelegt werden.
- Leider lassen sich die Urlaube im HR-System und die Abwesenheiten im Planungssystem nicht per ID abgleichen. Es muss hier jeweils mit Start- und Enddatum sowie der Person abgeglichen werden.
- Ändert sich das End- oder das Startdatum eines Urlaubs, darf die Abwesenheit im Planungssystem gelöscht und neu angelegt werden, sie muss nicht per update-Methode aktualisiert werden. Wenn sich der Status eines Antrags ändert (Urlaub wurde genehmigt), soll das per update im Planungssystem geschrieben werden.
- Es können noch weitere Tests hinzugefügt werden, um die Implementierung zu unterstützen.
- Es muss keine Fehlerbehandlung für die Schnittstellenaufruf implementiert werden um den Rahmen der Aufgabe nicht zu sprengen.

Aufgabe 3: REST Ressourcen

Bitte setze REST Ressourcen für eine einfache Taskliste um. Die Daten sollen im JSON Format ausgetauscht werden, ein Task hat dabei die folgende Struktur:

```
{
    „title“: string,
    „description“: string,
    „done“: boolean,
    „dueDate“: timestamp
}
```

Ressource „/api/tasks“

Methode GET: Liefert alle gespeicherten Tasks als JSON Array.

Methode POST: Legt einen neuen Task an (title und description sind Pflichtfelder).

Ressource „/api/tasks/{id}“

Methode GET: Liefert Einzeldatensatz per ID.

Methode PUT: Überschreibt die Ressource.

Methode PATCH: Überschreibt die angegebenen Teile der Ressource.

Methode DELETE: Löscht die Ressource.

Hinweise zur Umsetzung

- Die Umsetzung soll testgetrieben erfolgen. Bitte ergänze die Testklasse `TaskControllerTest` um die notwendigen Methoden.
- Die Methode POST soll den URI der angelegten Ressource im Location-Header zurückliefern.
- Der HTTP Status ist gemäß der gängigen Praxis für REST Ressourcen zu wählen.
- Die Daten können per JPA in der vorkonfigurierten H2 Datenbank persistiert werden, hierzu kann ein Flyway-Skript zur Generierung der Datenbankstruktur hinzugefügt werden. Alternativ genügt es, die Daten im Speicher zu halten, zum Beispiel in einer Map.
- Die Eigenschaft dueDate soll auf der Java Seite vom Typ LocalDateTime sein.
- Eine zusätzliche Schicht für Businesslogik kann eingeführt werden, muss aber nicht.