**מערכת CRUD לניהול סטודנטים - הנחיות לביצוע**

**שלבי הכנה ראשוניים**

✓ **הגדרת הפרויקט**

צור פרויקט Spring Boot חדש באמצעות Spring Initializr

בחר את התלויות הבאות: Lombok, Spring Web

בחר Java 17 או גרסה חדשה יותר

בחר Maven כמנהל החבילות

- **הגדרת מבנה החבילות**

צור את מבנה החבילות בהתאם לדרישה:

`***your-Package.model***`
`***your-Package.service***`
`***your-Package.controller***`

**<u>שלבים לביצוע</u>**

**שלב 1: יצירת מחלקת המודל**

✓ **הגדרת מחלקת Model**

צור מחלקה בשם Student בחבילת model

הוסף את כל השדות הנדרשים (id, firstName, lastName, age)

השתמש ב-Lombok כדי להקל על הקוד:

הוסף את האנוטציות הבאות:

@Data - ליצירת getters, setters, toString, equals ו-hashCode אוטומטיים

@NoArgsConstructor - ליצירת קונסטרקטור ריק

@AllArgsConstructor - ליצירת קונסטרקטור עם כל השדות

**שלב 2: יצירת שירות**

✓ **הגדרת ממשק Service**

צור ממשק בשם StudentService בחבילת service

הגדר את החתימות של כל המתודות הנדרשות:

**getAllStudents()**
**addStudent(Student student)**
**updateStudent(Student student)**
**deleteStudent(Long id)**

✓ **יישום ממשק Service**

צור מחלקת יישום בשם StudentServiceImpl שמיישמת את הממשק StudentService

הוסף אנוטציית Service@

יצור מאגר נתונים פנימי מסוג ArrayList לאחסון סטודנטים עם מספר סטודנטים התחלתיים:

```
List<Student> students = new ArrayList<>(Arrays.asList(
  new Student(1L, "Alice", "Moskovitz", 21.3),
  new Student(2L, "Bob", "Smith", 22.3),
  new Student(3L, "Charlie", "Brown", 23.3),
  new Student(4L, "David", "Miller", 24.3)
));
```

יישם את כל המתודות הנדרשות לפי הדוגמה:

```
// addStudent
public String addStudent(Student student) {
  // check if a student already exists
  if (students.stream().anyMatch(s -> s.getId().equals(student.getId()))) {
    return ("Student with id " + student.getId() + " already exists");
  }
  students.add(student);
  return "Student added successfully";
}
```

וודא שבמתודות אחרות יש גם בדיקות תקינות דומות

**שלב 3: יצירת בקר**

✓ **הגדרת בקר (Controller)**

צור מחלקה בשם StudentController בחבילת controller

הוסף אנוטציית @RestController

הגדר prefix מתאים באמצעות @RequestMapping("/student")

צור שדה פרטי מסוג StudentService והשתמש בהזרקת תלויות דרך קונסטרקטור

יישם את כל נקודות הקצה (endpoints) הנדרשות עם אנוטציות מתאימות:

```
@GetMapping("/getAllStudents")
@PostMapping("/addStudent")
@PutMapping("/updateStudent")
@DeleteMapping("/deleteStudent/{id}")
```

למשל:

```
@DeleteMapping("/deleteStudent/{id}")
public String deleteStudent(@PathVariable Long id) {
  return studentService.deleteStudent(id);
}
```

## נושאים לתשומת לב

**עיצוב קוד**

✓ **הזרקת תלויות באמצעות קונסטרקטור**:

השתמש בהזרקת תלויות דרך קונסטרקטור ולא דרך שדות (field injection)

זה מאפשר בדיקות יחידה טובות יותר וגישה יותר נקייה

✓ **בדיקות תקינות**:

הוסף בדיקות תקינות בשכבת השירות

בדוק האם סטודנט קיים לפני עדכון או מחיקה

בדוק האם ID כבר קיים לפני הוספת סטודנט חדש

**תגובות ונקודות קצה (REST API)**

✓ **החזרת מידע נכון**:

החזר מחרוזות מידע ברורות למשתמש בתגובה לפעולות

השתמש בהחזרת ערכים ישירים (Strings ו-Collections) - נלמד בהמשך עטיפה ב-ResponseEntity

- ✓ **טיפול בשגיאות פשוט**:

ממש טיפול בשגיאות בתוך מתודות הבקר באמצעות try-catch

החזר הודעות שגיאה ברורות למשתמש

**<u>פקודות HTTP לבדיקת המערכת</u>**

להלן סט פקודות HTTP שתוכל להשתמש בהן לבדיקת המערכת שפיתחת:

- ✓ **קבלת כל הסטודנטים (תחילה, המערכת כבר מכילה סטודנטים התחלתיים)**

***GET http://localhost:8080/student/getAllStudents***
***Accept: application/json***

**2. הוספת סטודנט חדש**

***POST http://localhost:8080/student/addStudent***
***Content-Type: application/json***
***Accept: application/json***

```
{
  "id": 5,
  "firstName": "ישראל",
  "lastName": "ישראלי",
  "age": 22.5
}
```

**3. הוספת סטודנט נוסף**

```
POST http://localhost:8080/student/addStudent
Content-Type: application/json
Accept: application/json

{
  "id": 6,
  "firstName": "רחל",
  "lastName": "כהן",
  "age": 24.0
}
```

**4. ניסיון להוסיף סטודנט עם ID שכבר קיים**

```
POST http://localhost:8080/student/addStudent
Content-Type: application/json
Accept: application/json

{
  "id": 1,
  "firstName": "שרה",
  "lastName": "לוי",
  "age": 21.5
}
```

**5. בדיקה שהסטודנטים נוספו בהצלחה**

```
GET http://localhost:8080/student/getAllStudents
Accept: application/json
```

**6. עדכון פרטי סטודנט קיים**

```
PUT http://localhost:8080/student/updateStudent
Content-Type: application/json
Accept: application/json

{
  "id": 1,
  "firstName": "ישראל",
  "lastName": "ישראלי",
  "age": 23.0
}
```

**7. ניסיון לעדכן סטודנט שלא קיים**

```
PUT http://localhost:8080/student/updateStudent
Content-Type: application/json
Accept: application/json

{
  "id": 99,
  "firstName": "לא",
  "lastName": "קיים",
  "age": 25.0
}
```

**8. מחיקת סטודנט לפי ID**

```
DELETE http://localhost:8080/student/deleteStudent/2
Accept: application/json
```

**9. ניסיון למחוק סטודנט שכבר נמחק**

```
DELETE http://localhost:8080/student/deleteStudent/2
Accept: application/json
```

**10. בדיקה סופית של הסטודנטים במערכת**

```
GET http://localhost:8080/student/getAllStudents
Accept: application/json
```

**<u>הערות:</u>**

- **פורט הפעלת השרת**: הפקודות מניחות שהשרת רץ על פורט 8080 (ברירת המחדל של Spring Boot). אם השתמשת בפורט אחר, יש לשנות את הכתובת בהתאם.
- **תגובות צפויות**:

  לפקודות GET: רשימה של אובייקטים JSON

  לפקודות POST, PUT, DELETE: הודעת טקסט על הצלחה או כישלון
- **סדר הבדיקות**: חשוב לבצע את הבדיקות בסדר הנכון, מכיוון שהן מסתמכות על מצב המערכת מהפעולות הקודמות.
- **שימוש ב-HTTP Client של Intellij Ultimate Edition**: אם אתה משתמש ב-IntelliJ, תוכל להריץ את כל הפקודות ישירות מתוך הסביבה.
- **שימוש ב-Postman**: תוכל להעתיק את תוכן הגוף של הבקשות והנתיבים ישירות ל-Postman ולבצע את הבדיקות משם.