Jackson:
https://www.baeldung.com/jackson

Jackson annotation:
ספרייה שמאפשרת סרליזציה של אובייקטים לjson, ודסרליזציה של json לאובייקט בjava.
השימוש באנוטציות מאפשר גמישות במבנה האובייקטים והjson שמוחזר.
בתהליך הסרליזציה המשתנים והפונקציות שהן public נכתבות לjson שחוזר.

jackson **Serialization** annotation:
(serialization = java object -> JSON)

@JsonAnyGetter:
אנטוציה ברמת הפונקציה. מאפשר להשתמש בשדה מסוג map כשדות של המחלקה, כלומר בסרליזיה של מחלקה עם שדה map שמשתמש באנוטציה הזאת, האובייקט יוצג עם השדות של ה map כאילו היו חלק מהשדות במחלקה, כאשר הם יהיו מהצורה של "key":"vaule"
ניתן לעביר לאנטוציה ארגומנט false לבטל את האנוטציה, במקרה הזה ה map יומר לJson, ויוצג בצורת json כחלק מהשדות של המחלקה.

@JsonGetter: alternative to @JsonProperty
אנטוציה ברמת השדה. מגדיר שהשדה הזה יופיע ב json שנקבל.

@JsonGetter: alternative to @JsonProperty
אנטוציה ברמת הפונקציה. מאפשר להגדיר מפתח לשדה.

@JsonPropertyOrder:
אנטוציה ברמת המחלקה. מאפשרת לקבוע את הסדר בוא יופיע השדות בסרליזציה.
לדוגמא (@JsonPropertyOrder({ "name", "age" }, מציין שבסרליזציה השם יופיע לפני הגיל.
ניתן להעביר לאנוטציה ארגונט שקובע שסדר השדות יהיה אלפבטי.

@JsonRootName:
אנוטציה ברמת המחלקה. מאפשר לקבוע ערך שיעטוף את הגייסון שיוצא

@JsonSerialize:
רנטוציה ברמת השדה. מאפשר להגדיר מחלקה שתהיה אחראי על סרליזציה של שדה.
המחלקה צריכה לרשת מ StdSerializer<T>.

jackson **Deserialization** annotation:
(deserialization = JSON -> java object)

@JsonCreator:
אנטוציה ברמת הפונקציה הבונה (constructor). מאפשר לגשר על פער בין השדה במחלקה לבין הערך בגייסון. כלומר, אם יש לנו ב json ערך theNAme, ואנחנו רוצים שהוא יכנס לשדה name נשתמש בJsonCreator וב JsonProperty כך:

```
public class BeanWithCreator {
    public int id;
    public String name;

    @JsonCreator
    public BeanWithCreator(
      @JsonProperty("id") int id,
      @JsonProperty("theName") String name) {
        this.id = id;
        this.name = name;
    }
}
```

כפי שניתן לראות, הjsonProperty מאפשר לגשר על הפער הזה

@JacksonInject:
אנוטציה ברמת השדה. מציין שהשדה הזה יקבל ערך מה Injection ולא מהמידע בjson.
הinject מתבצע כך:

```
InjectableValues inject = new InjectableValues.Std()
    .addValue(int.class, 1);
 BeanWithInject bean = new ObjectMapper().reader(inject)
    .forType(BeanWithInject.class)
    .readValue(json);
```

@JsonSetter
אנוטציה ברמת המטודה. מסמל פונקציה כ setter בשביל ה

jackson **Serialization and Deserialize** annotation:

@JsonIgnore:
אנטוציה על שדה. מאפשר להתעלם משדה בזמן סרליזציה ודסרליזיציה, כלומר השדה לא יופיע בjson שנקבל, או באובייקט שנבנה.

@JsonIgnoreProperties:
אנטוציה על מחלקה. מאפשר להתעלם מאוסף של שדות.

@JsonIgnoreType:

אנטוציה על מחלקה. מאפשר להתעלם מהמחלקה, כלומר מכל השדות של המחלקה.
דוגמא: נגדיר מחלקה address, במצב כזה בזמן סרליזציה הכתובת לא תופיע ב json שנקבל. (יכול להתאים להסתרת מידע רגיש כמו סיסמאות)

@JsonAutoDetect:

אנטוציה ברמת המחלקה. מאפשר לדרוס את הגישה של השדות במחלקה, למשל לאפשר סרליזציה לשדה שהוא פרטי במחלקה.
אפשר להגדיר את האנוטציה על האלמנטים הבאים:

- creatorVisibility
- fieldVisibility
- getterVisibility
- setterVisibility
- isGetterVisibility

וערכים אפשריים לאנטוציה מהמחלקה JsonAutoDetect:

- ANY
- DEFAULT
- NON_PRIVATE
- NONE
- PROTECTED_AND_PRIVATE
- PUBLIC_ONLY

אנוטציות נוספות:

@JsonFormat:

אנוטציה על שדה או מטודה. האנוטציה מגדירה את הפורמט של השדה בתהליך הסרליזציה. למשל הגדרה של תאריך בפורמט String ותבנית yyyy-MM-dd