

# מסמך דרישות - מיני מערכת רישום משתמשים:

## FastAPI → Kafka → MongoDB

### (1) מטרת הפרויקט

לבנות מערכת מינימלית המאפשרת:

1. קבלת פרטי משתמש דרך **endpoint יחיד** בשרת **FastAPI**.
2. ביצוע **ולידציה באמצעות Pydantic**.
3. פרסום (publish) של המשתמש שנרשם ל-**Kafka**.
4. שירות נוסף (consumer) שקורא הודעות מ-**Kafka** ושומר את המשתמש במסד **MongoDB**.

הפרויקט חייב לרוץ מקצה לקצה באמצעות **Docker Compose**.

מומלץ להיעזר [בגיטהאב שנלמד ביום שני](#) שהציג את השימוש ב-**Kafka**

---

### (2) ארכיטקטורה כללית

המערכת תכלול **4 שירותים** (services) ב-**Docker Compose**:

1. **api** – שרת **FastAPI**, שמתפקד גם כ-**Producer** ל-**Kafka**. (מכיל **endpoint** אחד בלבד).
  2. **consumer** – שירות **Python** שקורא באופן רציף מ-**Kafka** וכותב ל-**MongoDB**.
  3. **kafka** – **Kafka broker** (ניתן להשתמש ב-**image** ציבורי).
  4. **mongodb** – **MongoDB** (**image** ציבורי).
-

## 3) דרישות פונקציונליות

### 3.1 שרת FastAPI (שירות api)

- חייב לכלול endpoint יחיד:  
1. `POST /register`
- ה־endpoint מקבל גוף בקשה (JSON) עם פרטי משתמש, מבצע ולידציה באמצעות **Pydantic**, ואם תקין:
  1. יוצר אובייקט משתמש תקני (לפי הסכמה).
  2. שולח את הנתונים ל־Kafka כ־message (בפורמט JSON).

#### תגובות ה-API

- במקרה הצלחה:
  - קוד: `201 Created` (או `200 OK` אם בחרתם, אך מומלץ 201)

גוף תשובה לדוגמה:

```
}  
  "status": "accepted"  
  "message": "user published to kafka"  
{
```

- 
- במקרה ולידציה נכשלת:
  - קוד: `422 Unprocessable Entity` (ברירת המחדל של FastAPI/Pydantic)
  - הגוף יוחזר אוטומטית ע"י FastAPI בהתאם לשגיאות.

### 3.2 שירות Consumer (שירות consumer)

- מאזין ל־Kafka topic מוגדר.
- עבור כל הודעה שמתקבלת:
  1. מפרסר את ה־JSON.
  2. מבצע ולידציה בסיסית/רובסטיות (לפחות בדיקה שהשדות הנדרשים קיימים).
  3. כותב את המשתמש ל־MongoDB collection מוגדר.
- השירות צריך לרוץ ברציפות (loop) ולהמשיך לצרוך הודעות.

### 3.3 MongoDB (שירות mongodb)

- יש לשמור כל משתמש למסד ב־collection בשם קבוע (לדוגמה: `users`).
- הוספת מסמך חדש לכל הודעה.
- אין צורך ב־authentication ל־MongoDB בשלב זה (אלא אם תרצו – לא חובה).

## Pydantic – וולידציה (Schema) סכמה (4)

### 4.1 מודל משתמש (User)

ה-API חייב לקבל את השדות הבאים:

- `full_name` (string) – חובה, מינימום 2 תווים
- `email` (string) – תקין email חובה, חייב להיות
- `age` (integer) – חובה, חייב להיות בין 13 ל-120
- `phone` (string) – אופציונלי, אם קיים: מינימום 9 תווים, מקסימום 15
- `city` (string) – אופציונלי, אם קיים: מינימום 2 תווים

#### דרישות נוספות

- יש להוסיף שדה `created_at`, למשל ISO datetime. (יכול להישלח על ידי הלקוח או להיווצר בצד השרת)
- מומלץ: להוסיף `user_id` שנוצר בשרת (UUID) כדי לזהות משתמשים (לא חובה, אבל מומלץ).

---

## Kafka – דרישות (5)

- חייב להיות Topic אחד מוגדר מראש, לדוגמה:
  - `users.registered`
- הודעת Kafka תישלח בפורמט JSON (stringified JSON).
- יש להשתמש בספריית `confluent-kafka` לצורך שימוש ב-Kafka

---

## Docker Compose – דרישות הרצה (6)

### 6.1 חובה

- הפרויקט חייב לכלול קובץ:
  - `docker-compose.yml`
- חובה להשתמש ב-MongoDB Docker image ציבורי.
- חובה לבצע קונטיינריזציה לשני שירותי Python:
  - `api` עם Dockerfile
  - `consumer` עם Dockerfile
- כל השירותים חייבים להיות באותה רשת Compose (ברירת מחדל מספיקה).
- יש להגדיר משתני סביבה (environment variables) כדי לא לקודד כתובות בקוד.

## 6.2 משתני סביבה (דוגמה)

- `KAFKA_BOOTSTRAP_SERVERS` (למשל: `kafka:9092`)
  - `KAFKA_TOPIC` (למשל: `users.registered`)
  - `MONGO_URI` (למשל: `mongodb://mongodb:27017`)
  - `MONGO_DB` (למשל: `appdb`)
  - `MONGO_COLLECTION` (למשל: `users`)
- 

## 7 מבנה פרויקט מומלץ

מינימום:

```
/project
├── docker-compose.yml
├── /api
│   ├── Dockerfile
│   ├── Requirements.txt
│   ├── main.py
│   ├── models.py
│   └── /consumer
│       ├── Dockerfile
│       ├── Requirements.txt
│       └── consumer.py
```

---

## 8 דרישות איכות ותיעוד (חובה)

- קובץ `README.md` שמכיל:
  1. תיאור קצר של המערכת
  2. איך מריצים:
    - `docker compose up --build`
    - 3. דוגמת בקשת `curl` ל-`register/`
    - 4. איך לוודא שהנתונים נשמרו ב-MongoDB (פקודה/הסבר קצר)

---

## 9 קריטריונים להצלחה (Acceptance Criteria)

המערכת תיחשב עובדת אם מתקיימים כל אלו:

1. `docker compose up --build` מרים את כל השירותים ללא קריסות מתמשכות.
2. קריאה ל-`POST /register` עם JSON תקין מחזירה הצלחה.
3. המשתמש שפורסם ל-Kafka נקלט ע"י consumer.
4. המשתמש נשמר ב-MongoDB ב-collection הנדרש.
5. קריאה עם JSON לא תקין מחזירה שגיאת ולידציה (422).

---

## 10 המשך המשימה

לאחר הגעה למצב של מערכת עובדת מקצה לקצה, יינתנו הנחיות נוספות (שלב 2), שיישענו על הפרויקט הבסיסי הזה.

---