

## **מסמך דרישות - מינוי מערכת רישום משתמשים:**

### **FastAPI → Kafka → MongoDB**

#### **1) מטרת הפרויקט**

לבנות מערכת מינימלית המאפשרת:

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

הפרויקט חייב לרווח מקצה לפחות באמצעות Docker Compose.

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

---

#### **2) ארכיטקטורה כללית**

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

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

### 3) דרישות פונקציונליות

#### 3.1 שירות FastAPI (שירות api)

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

#### תగובות 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 נכשלה:
  - קוד: 422 Unprocessable Entity (ברירת המחדל של FastAPI/Pydantic)
  - הגוף יחזיר אוטומטית ע"י FastAPI בהתאם לשגיאות.

#### 3.2 שירות Consumer (שירות consumer)

- amazon ל-Kafka topic מוגדר.
- עברו כל הודעה שמתقبلת:
- 1. מפרשר את ה-JSON.
- 2. מבצע ולידציה בסיסית/רוביוטית (פחות בדיקה שהנדשים קיימים).
- 3. כותב את המשתמש ל-MongoDB collection מוגדר.
- השירות צריך להריץ ברציפות (loop) ולהמשיך לצורך הודעות.

#### 3.3 MongoDB (שירות db)

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

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

### 4.1 מודל משתמש (User)

ה-API חייב לקבל את השדות הבאים:

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

#### דרישות נוספת

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

## 5 – Kafka – דרישות

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

## 6 – Docker Compose – דרישות הרצה

### 6.1 חובה

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

## 6.2 משתני סביבה (דוגמה)

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

## 7) בניית פרויקט מומלץ

מינימום:

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

---

## 8) דרישות אינטראקטיביות (חובה)

- קובץ `README.md` שמכיל: •
  - תיאור קצר של המערכת
  - איך מרכיבים:
- `docker compose up --build` ■
  - דוגמת בקשת `curl -L register/`
  - איך לוודא שהנתונים נשמרו ב-DB MongoDB (פקודה/הסבר קצר)

---

## 9) קритריונים להצלחה (Acceptance Criteria)

המערכת תיחסב עובדת אם מתקיימים כל אלו:

1. קריאה ל-`docker compose up --build` מרים את כל השירותים ללא קרייסות 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, ינתנו הנחיות נוספות (שלב 2)**, שיישענו על הפרויקט הבסיסי זהה.

---