

תכנון ארכיטקטורת מערכת לניהול בתי דין מבוססת מיקרו-שירותים דורש גישה שיטתית שמפרידה את המערכת לשירותים עצמאיים ומוגדרים היטב. להלן תכנון מפורט:

**שלב 1: הגדרת שירותים עיקריים**

נחלק את המערכת לשירותים לפי תחומי אחריות:

1. **שירות ניהול תיקים (Case Management Service):**

טיפול ביצירת תיקים, עדכון ומחזור חיי התיק.

פונקציות עיקריות:

* Task<bool> CreateCase(CreateCaseDto caseDto);
* Task<CaseDetailsDto?> GetCaseDetails(int caseId);
* Task<bool> UpdateCase(int caseId, UpdateCaseDto updateCaseDto);
* Task<List<CaseDetailsDto>> GetCases();

1. **שירות חיפוש (Search Service):**

ניהול מיון וסינון של תיקים.

פונקציה עיקרית:

Task<List<CaseSummaryDto>> SearchCases(SearchCriteriaDto criteria);

1. **שירות הודעות ודיוור (Notification Service):**

שליחת מיילים לגורמים רלוונטיים.

פונקציה עיקרית:

Task SendEmail(EmailDto emailDto);

1. **שירות מסמכים (Document Management Service):**

ניהול מסמכים קשורים לתיק.

פונקציות עיקריות:

* Task<bool> UploadDocument(UploadDocumentDto documentDto);
* Task<DocumentDetailsDto?> GetDocumentDetails(int documentId);
* Task<bool> UpdateDocument(int documentId, UpdateDocumentDto updateDto);
* Task<bool> DeleteDocument(int documentId);
* Task<List<DocumentDetailsDto>> GetDocuments();

1. **שירות משתמשים והרשאות (User Management Service):**

ניהול משתמשים, הרשאות וזיהוי.

פונקציות עיקריות:

Task<AuthResponse> AuthenticateUser(AuthDto authDto);

**שלב 2: הגדרת קשרים בין שירותים**

**תרשים ארכיטקטורה (Pseudocode):**

Client -> API Gateway -> [Case Service, Search Service, Notification Service, User Service, Document Service]

* **API Gateway:** שער יחיד לניהול תקשורת בין הלקוח לשירותים.
* **Event Bus (למשל, RabbitMQ או Kafka):** לתקשורת אסינכרונית בין שירותים (למשל, הפצת מייל בעת פתיחת תיק).

**דוגמה לתהליך פתיחת תיק:**

1. לקוח שולח בקשה ל-Case Management Service דרך ה-API Gateway.
2. Case Management Service:
   * יוצר את התיק בבסיס הנתונים.
   * שולח הודעה ל-Notification Service דרך Event Bus להפצת מייל.
3. Notification Service שולח מייל לכל הגורמים המעורבים.

**שלב 3: תכנון בסיס נתונים**

**מבנה בסיסי לפי שירותים (Database per Service):**

1. **Case Management Service:**

טבלה Cases:

CaseId (PK), Title, Description, Status, CreatedDate.

טבלה Participants:

ParticipantId (PK), CaseId (FK), Name, Role.

1. **Document Management Service:**

טבלה Documents:

DocumentId (PK), CaseId (FK), FilePath, UploadedBy, UploadedDate.

1. **Search Service:**

אינדקסים מותאמים למיון וסינון לפי קריטריונים (DateOpened, CaseNumber, Status).

**שלב 4: שיקולים לארכיטקטורה**

1. **סינכרוניות מול אסינכרוניות**:

* פעולות קריטיות כמו יצירת תיק יתבצעו באופן סינכרוני.
* פעולות לא קריטיות כמו שליחת מייל יבוצעו אסינכרונית.

1. **אבטחה**:

* שימוש ב-Authentication מבוסס JWT.
* הרשאות מבוססות תפקידים בכל שירות.

1. **אינטגרציה**:

* שימוש ב-HTTP API לתקשורת בין שירותים.
* Event Bus לניהול אירועים (Events) בין השירותים.

**שלב 5: טכנולוגיות מומלצות**

**Backend:**

* .NET Core ליישום מיקרו-שירותים.
* Entity Framework לניהול בסיסי נתונים.

**Frontend:**

* Angular לפיתוח הממשק.
* Angular Material לשיפור חוויית המשתמש.

**Infrastructure:**

* Docker לניהול מכולות.
* Kubernetes לניהול השירותים.

**Messaging:**

* RabbitMQ או Kafka לניהול Event Bus.

**שלב 6: פיתוח ויישום**

1. **הקמת שירותים בסיסיים**:

* יצירת פרויקט לכל שירות בנפרד.
* כתיבת פונקציות CRUD לכל שירות.

1. **הקמת Event Bus:**

* הגדרת נושאים (Topics) והודעות (Messages) לשירותי הודעות.

1. **חיבור Frontend ל-API Gateway:**

* בניית רכיבי Angular המתקשרים עם השירותים.

**שלב 7: בדיקות**

1. **Unit Tests:** בדיקת כל פונקציה בנפרד.
2. **Integration Tests:** בדיקת תקשורת בין שירותים.
3. **UI Tests:** וידוא מיון, סינון, והצגת נתונים.