Text

Description automatically generated

פרויקט גמר

**Skyline CRS**

Skyline Computer Reservation System

מכללת אורט סינגאלובסקי

שנה"ל תשפ"ב

|  |  |
| --- | --- |
| **שם התלמיד:** | עידו סבן |
| **ת.ז. התלמיד:** | 212619787 |
| **שם המנחה:** | סרן אופק אוחיון |
| **תאריך הגשה:** | 15/5/2022 |

תוכן עניינים

[1. הצעת פרויקט 3](#_Toc103476864)

[1.1. תיאור הפרויקט 3](#_Toc103476865)

[1.2. רקע תיאורטי 4](#_Toc103476866)

[1.2.1. תהליך הזמנת הטיסות 4](#_Toc103476867)

[1.2.2. מונחים 5](#_Toc103476868)

[1.3. תהליכים עיקריים בפרויקט 5](#_Toc103476869)

[1.4. טכנולוגיות בפרויקט 6](#_Toc103476870)

[1.5. תרשימים 7](#_Toc103476871)

[1.5.1. תרשים המערכת 7](#_Toc103476872)

[1.5.2. תרשים מסכים (User Flow) 8](#_Toc103476873)

[1.6. לוחות זמנים 8](#_Toc103476874)

[1.7. אישורים 8](#_Toc103476875)

[2. מבוא 9](#_Toc103476876)

[2.1. רקע 9](#_Toc103476877)

[2.2. תהליך המחקר 9](#_Toc103476878)

[2.3. סקירת ספרות 10](#_Toc103476879)

[2.4. אתגרים מרכזיים 10](#_Toc103476880)

[2.4.1. הבעיות איתן התמודדתי 10](#_Toc103476881)

[2.4.2. הסיבות לבחירת הנושא 11](#_Toc103476882)

[2.4.3. מוטיבציה לעבודה 11](#_Toc103476883)

[2.4.4. הצורך בפרויקט 11](#_Toc103476884)

[2.4.5. הפתרונות לבעיה 11](#_Toc103476885)

[3. מטרות ויעדים 12](#_Toc103476886)

[4. אתגרים 13](#_Toc103476887)

[5. מדדי הצלחה 13](#_Toc103476888)

[6. רקע תיאורטי 13](#_Toc103476889)

[6.1. רקע כללי 13](#_Toc103476890)

[6.2. רקע טכנולוגי 15](#_Toc103476891)

[7. תיאור מצב קיים 16](#_Toc103476892)

[8. ניתוח חלופות מערכתיות 17](#_Toc103476893)

[9. החלופה המערכתית הנבחרת 18](#_Toc103476894)

[10. אפיון המערכת 18](#_Toc103476895)

[10.1. ניתוח דרישות המערכת 18](#_Toc103476896)

[10.2. מודולים במערכת 19](#_Toc103476897)

[10.3. אפיון פונקציונלי 19](#_Toc103476898)

[10.4. ביצועים עיקריים 20](#_Toc103476899)

[10.5. אילוצים 20](#_Toc103476900)

[11. תיאור הארכיטקטורה 21](#_Toc103476901)

[11.1. ארכיטקטורת המערכת 21](#_Toc103476902)

[11.2. תיאור הרכיבים במערכת 21](#_Toc103476903)

[11.3. ארכיטקטורת הרשת 22](#_Toc103476904)

[11.4. פרוטוקולי התקשורת 23](#_Toc103476905)

[11.5. תיאור השרת והלקוח 24](#_Toc103476906)

[12. ניתוח מקרי השימוש של המערכת 24](#_Toc103476907)

[12.1. תיאור מקרי השימוש במערכת 24](#_Toc103476908)

[12.1.1. מקרה שימוש: Search for Flights 24](#_Toc103476909)

[12.1.2. מקרה שימוש: Get Flight Details 26](#_Toc103476910)

[12.1.3. מקרה שימוש: Book Flight 27](#_Toc103476911)

[12.1.4. מקרה שימוש: Log In 28](#_Toc103476912)

[12.1.5. מקרה שימוש: Get Booking Details 29](#_Toc103476913)

[12.1.6. מקרה שימוש: Update Booking 29](#_Toc103476914)

[12.1.7. מקרה שימוש: Cancel Booking 29](#_Toc103476915)

[12.1.8. מקרה שימוש: Check In 30](#_Toc103476916)

[12.2. מקרי השימוש עבור הפונקציות העיקריות במערכת 30](#_Toc103476917)

[12.3. מבני הנתונים שהושמשו 30](#_Toc103476918)

[12.4. הקשרים בין היחידות השונות 31](#_Toc103476919)

[12.5. עץ מודולים 31](#_Toc103476920)

[12.6. דיאגרמת UML Use Case 32](#_Toc103476921)

[12.8. תרשים UML Class Diagram 33](#_Toc103476922)

[12.9. תרשים Deign Class 34](#_Toc103476923)

[12.10. תרשים מחלקות 34](#_Toc103476924)

[12.11. תיאור המחלקות המוצעות 34](#_Toc103476925)

# 1. הצעת פרויקט

**שם הסטודנט:** עידו סבן

**ת.ז. הסטודנט:** 212619787

**סמל מוסד:** 570077

**שם המוסד:** מכללת אורט סינגאלובסקי

**שם רכז המגמה:** שחר אוחנה

**שם המנחה:** סרן אופק אוחיון (צה"ל), שחר אוחנה (מכללה)

**מקום ביצוע הפרויקט:** חיל התקשוב, צה"ל

**שם הפרויקט:** Skyline CRS

## 1.1. תיאור הפרויקט

**מערכת הזמנות ממוחשבת** (Computer/Central Reservation System או (CRS היא מערכת שמטרתה לנהל את ההזמנות והעסקאות של ארגון, כמו חברות טיסה, מלונות, וכדומה. מערכות אלו קיימות כיום בכל מקום, בין אם ארגונים גדולים כמו חברות טיסה בינלאומיות ובין אם בתי קולנוע מקומיים.

מערכות הזמנות ממוחשבות הופיעו לראשונה בקרב חברות התעופה, שחיפשו פתרון ממוחשב על מנת לייעל את תהליך ההזמנות לטיסה. בעקבותיהן הלכו גם המלונות ושירותים דומים, עד שהפך הדבר לנפוץ כפי שהוא היום.

על מערכות אלו להיות בעלות זמינות גבוהה (high availably) עם סובלנות גבוהה לתקלות (fault tolerance), ולכם כך על המערכת להיות סקלבילית (scalable) על פי דרישה לשירותיה ולעומסים.

**Skyline CRS** היא מערכת לניהול טיסות עבור חברת תעופה בדיונית Skyline שמטרתה לנהל את כל הקשור לתהליך ההזמנה של הטיסה והעליה אליה. המערכת תעוצב בארכיטקטורת microservices על מנת לאפשר horizontal scaling של המערכת ובכך לממש את הדרישות של זמינות גבוהה, סובלנות לתקלות וסקלביליות.

המערכת תציע בנוסף ממשק משתמש אינטרנטי ללקוחות, היתקשר עם ה-API של המערכת, דרכו ניתן יהיה לחפש טיסות, להזמין טיסות (booking) ולבצע online check-in. יש לציין שלמערכת יהיה API ציבורי, כך שגם סוכנויות נסיעות חיצוניות (Online Travel Agency או OTA), לדוגמת Booking.com, Skyscanner, וכדומה, יוכלו לגשת אליו.

חשוב לציין **שלא אתייחס** בפרויקט לנושא הכבודה לטיסה וכן לעניינים כלכליים (כמו תשלום על הזמנות), אלא אתמקד בנושאי הזמנות הכרטיסים למקומות בטיסה וכל הכרוך בכך.

## 1.2. רקע תיאורטי

### 1.2.1. תהליך הזמנת הטיסות

Diagram

Description automatically generated

שלבים בתהליך:

1. **חיפוש טיסות באתר חברת התעופה** – המשתמש בוחר שדה תעופה למוצא ושדה תעופה של היעד עבור הטיסה שהוא מחפש בין טווח תאריכים שהוא בוחר. כמו כן הוא יכול לבחור גם במספר הנוסעים לטיסה. לאחר שהוא מבצע חיפוש, תוצג לו רשימת טיסות מתאימות לפרמטרים שנתן. מהן יוכל המשתמש לבחור אחת מהטיסות המוצעות ולבצע הזמנה.
2. **הזמנת הטיסה** – לאחר בחירת הטיסה מרשימת הטיסות, המשתמש ימלא פרטים אישיים, ולאחר מכן אף יוכל לבחור במקומות בטיסה. לאחר שיבצע את ההזמנה הוא יקבל מספר הזמנה (PNR locator או Booking number) המזהה את הזמנתו. באמצעות מספר ההזמנה יוכל המשתמש לבדוק בכל עת את מצב ההזמנה וכן לגשת לכרטיס ה-e-ticket שיקבל במייל גם דרך האתר. כעת ההזמנה מחכה לאישור ותישאר במצב זה עד שישלח לנוסע כרטיס דיגיטלי.
3. כרטיס הטיסה הדיגיטלי (e-ticket) ישלח לנוסע באימייל לאחר כמה זמן (לרוב עד 3 ימים ברוב חברות הטיסה), שם יהיו פרטי ההזמנה השונים. רק לאחר שהתקבל כרטיס הטיסה ניתן יהיה לעשות צ'ק-אין.
4. **צ'ק-אין מקוון (online check-in)** – הצ'ק-אין לטיסה נפתח 72 שעות לפני זמן העלייה למטוס. תהליך זה הכרחי על מנת לקבל את כרטיס העלייה למטוס (boarding pass) – בלעדיו לא יתאפשר לעלות על הטיסה! בתהליך הצ'ק-אין על הנוסעים יהיה למלא פרטים חסרים (כמו בחירת מקומות במקרה שלא בחרו בשלב ההזמנה) ולענות על שאלות אבטחה שונות. את תהליך הצ'ק-אין המקוון יבצעו הנוסעים דרך אתר חברת התעופה, על ידי הכנסת מספר ההזמנה ומספר פרטי אימות נוספים, ובתומו יקבלו את כרטיס העלייה למטוס דרך האימייל והאתר.
5. **עליה למטוס** – הנוסעים יגיעו לשדה התעופה עם כרטיס העלייה למטוס שקיבלו (דיגיטלי או מודפס), יבצעו את ההליכים הדרושים בשדה, ויעלו על הטיסה בזמן שצוין בכרטיס העלייה למטוס.

### 1.2.2. מונחים

* **Passenger Name Record (PNR)** – קובץ הנוצר עבור כל הזמנה, המתאר את פרטי כל הנוסעים וכן פרטים נוספים (כמו פרטי התקשרות, מצב ההזמנה, וכדומה). לאור זאת, ניתן לומר ש-PNR הוא מונח טכני המקביל להזמנה (booking).
* **PNR Locator (Booking Number)** – מספר המזהה את ה-PNR שנוצר עבור ההזמנה. מספר זה הכרחי על מנת לבדוק את מצב ההזמנה, לבצע שינויים בהזמנה, לבצע צ'ק-אין לטיסה, וכדומה.
* **E-ticket** – כרטיס (מסמך) דיגיטלי המתקבל לאחר אישור ההזמנה (במציאות לאחר בדיקת פרטים וחיוב האשראי) המתאר את ההזמנה ופרטיה. בין הפרטים העיקריים שמכיל ה-e-ticket הוא מספר ההזמנה (PNR locator) שיאפשר לבצע check-in בשלב מאוחר יותר.
* **צ'ק-אין (Check-in)** – שלב שניתן לביצוע או באופן מקוון (online check-in), לרוב בין 24-72 שעות לפני הטיסה, או בשדה התעופה לפני העלייה למטוס. בשלב זה נבדקים הפרטים האישיים של הנוסעים (למשל מספר דרכון, תאריך לידה, וכולי) לצורכי אימות, וכן ניתנת האפשרות לנוסעים לרוב לבחור מקומות (עבור תשלום נוסף), להגדיל את כמות הכבודה, להוסיף אוכל לטיסה, וכולי. לבסוף, יונפק עבור על נוסע כרטיס עליה למטוס (boarding pass). למעשה ללא שלב זה לא ניתן יהיה לקבל את כרטיס העלייה למטוס, וכן, לא יהיה ניתן לעלות לטיסה.
* **כרטיס עליה למטוס (Boarding Pass)** – הכרטיס באמצעותו נוסע עולה למטוס. הכרטיס יכיל פרטים כמו מושב בטיסה, זמן העלייה למטוס, פרטי נוסע, טרמינל, וכדומה. כרטיס זה יכול להיות או דיגיטלי על ידי ביצוע צ'ק-אין מקוון או מודפס אם בוצע בצ'ק-אין בשדה התעופה.

## 1.3. תהליכים עיקריים בפרויקט

* חיפוש טיסות על פי פרמטרים נתונים (מוצא, יעד, טווח תאריכים, מספר נוסעים, מחלקת טיסה).
* הצגת פרטי טיסה (פרטי הטיסה, מבנה המטוס, מקומות פנויים).
* הזמנת טיסה (יצירת PNR).
* הצגת פרטי הזמנה.
* עדכון או ביטול הזמנה.
* ביצוע צ'ק-אין לטיסה וקבלת כרטיס עליה למטוס..
* שליחת אימיילים (e-ticket, כרטיס עליה למטוס, אישורי הזמנה וביטול).
* ייעול מסד הנתונים של ה-PNR-ים על ידי העברת PNR-ים לא פעילים לארכיון.

## 1.4. טכנולוגיות בפרויקט

* **Docker** – פלטפורמה המאפשרת הרצת קונטיינרים (containers) ובניית תמונות (images) ליצירתם. בפרויקט, כל microservice ייארז כתמונה נפרדת וירוצו בתוך קונטיינרים נפרדים המתקשרים זה עם זה על גבי הרשת.
* **Kubernetes** – מערכת שפותחה על ידי גוגל המשתמשת לניהול ופריסה אוטומטית של קונטיינרים (אורכסטרציה של קונטיינרים – container orchestration). למעשה היא זו שתאפשר לקשר בין כל חלקי הפרויקט השונים ותעזור לממש את עקרונות הזמינות הגבוהה, סובלנות לתקלות וסקלביליות.
* **RabbitMQ** – פלטפורמת תיווך הודעות (message broker) המשמשת לניהול תורים תחת פרוטוקולים שונים, בעיקרAdvanced Message Queuing Protocol (AMQP). מטרת השימוש במתווך הודעות היא מניעת עומסים על ה-backend על ידי דחיית עבודות שלוקחות זמן ובכך לאפשר זמינות גבוהה יותר של המערכת ומניעת שגיאות במקרה של תקלות במערכת.
* **PostgreSQL** – מערכת לניהול בסיסי נתונים רציונליים (RDBMS). בפרויקט תשמש לאחסן את נתוני הטיסות והמקומות בהן.
* **Hasura** – מוצר שיוצר הפשטה על גבי בסיס נתונים רציונאלי, כמו PostgreSQL, וחושף באופן אוטומטי ממשק GraphQL על פי סכמת מסד הנתונים. ישמש להפשטת ההתממשקות עם מסד הנתונים של הטיסות וכן לשליטה על הגישה אליו.
* **MongoDB** – מערכת לניהול בסיסי נתונים לא רציונליים (NoSQL database) המשתמשת במסמכים (documents) דמויי JSON לאחסון הנתונים באוספים (collections), בניגוד למסדי נתונים רציונליים המשתמשים בטבלאות לייצוג הנתונים. בפרויקט תשמש לאחסן את ה-PNR-ים של ההזמנות השונות, שלרוב מאוחסנים כקבצים נפרדים, שכן אופי הנתונים תואם לאופי אחסונם במערכת זו.
* **Git** – מערכת לניהול גרסאות מבוזרת (distributed VCS) הפופולרית בעולם. תשתמש לנהל את הגרסאות השונות במהלך הפיתוח של הפרויקט.
* **GitHub** – שירות ניהול ואחסון אינטרנטי עבור Git repositories. יחד עם Git היא תשמש לניהול הקוד של הפרויקט לאורך הפיתוח.
* **ספריות עבור ה-backend לשפה Python:**
  + **FastAPI** – Web framework אסינכרונית ליצירת REST APIs. תשתמש בתוכנה Uvicorn להרצת השרת. ספרייה זו תשמש ליצירת microservices החושפים REST API במערכת.
  + **aio-pika** – ספרייה להתממשקות עם RabbitMQ. תשמש microservices שצריכים לשלוח הודעות לתורים או לקבל מהם הודעות.
  + **Strawberry** – ספרייה לתקשורת עם ממשקי GraphQL ויצירתם. תשמש microservices לתקשורת עם Hausra.
  + **Motor** – ספריית ODM לתקשורת אסינכרונית עם MongoDB. תשמש microservices שצריכים להתממשק עם מסד הנתונים של ה-PNR-ים.
* **ספריות עבור ה-backend לשפות JavaScript/Typescript:**
  + **Express** – ספרייה ליצירת שרת web אסינכרוני החושף REST API**.** ספרייה זו תשמש ליצירת microservices החושפים REST API במערכת.
  + **amqplib** – ספרייה להתממשקות עם RabbitMQ. תשמש microservices שצריכים לשלוח הודעות לתורים או לקבל מהם הודעות.
  + **graphql-request** – ספרייה לתקשורת עם ממשקי GraphQL ויצירתם. תשמש microservices לתקשורת עם Hausra.
  + **Mongoose** – ספרייה ODM לתקשורת עם MongoDB. תשמש microservices שצריכים להתממשק עם מסד הנתונים של ה-PNR-ים.
  + **Nodemailer** – ספרייה לשליחת אימיילים, על גבי פרוטוקול SMTP ופרוטוקולים נתמכים אחרים. תשמש את Email Service.
* **ספריות עבור ה-frontend לשפות JavaScript/Typescript:**
  + **React** – ספרייה ליצירת אפליקציות web שפותחה על ידי פייסבוק. הספרייה תשמש ליצירת אתר חברת התעופה.
  + **Axios** – ספריית JavaScript/Typescriptמבוססת Promises המפשיטה יצירת בקשות HTTP. תשמש לתקשורת בין ממשק המשתמש ל-API של המערכת.

## 1.5. תרשימים

### 1.5.1. תרשים המערכת

Diagram

Description automatically generated

### 1.5.2. תרשים מסכים (User Flow)

Diagram

Description automatically generated

## 1.6. לוחות זמנים





## 1.7. אישורים

**חתימת הסטודנט: A picture containing text, clipart

Description automatically generated**

**חתימת רכז המגמה:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**אישור ממשרד החינוך:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 2. מבוא

## 2.1. רקע

במדור בו אני נמצא בחיל התקשוב יש שימוש בטכנולוגיות הקשורות לענן. בין הטכנולוגיות בהן משתמש המדור, קיים השימוש ב-Docker ו-Kubernetes, טכנולוגיות שנועדו להפשיט להפוך את תחום הריצה בענן לסטנדרטי, ולהקנות שליטה רחבה יותר לארגון במערכות שרצות עליו.

מאחר שהמדור עובד על פרויקטים ברמת סיווג גבוהה, את הפרויקט שלי ביצעתי למטרת למידה והכנה לתפקיד בלבד, על דגש בטכנולוגיות שמשתמשים בהן בתפקיד על מנת שאבוא עם ניסיון וידע מוקדם.

לשם כך, הוצע שאבנה מערכת עבור ארגון כלשהו, שתדאג לטפל במידע עבור הארגון מא' ועד ת', בדגש על כך שתהיה scalable ותוכל לעמוד בעומסים, על ידי שימוש בטכנולוגיות שהוצגו לעיל.

לאחר מחשבה רבה, החלטתי על רעיון לפרויקט: מערכת Computer Reservation System (CRS) עבור חברת תעופה דמיונית בשם Skyline, שתדאג לתהליך ההזמנות המלא של החברה מול לקוחות אפשריים. המערכת תהיה תחשוף REST API ציבורי, דרכו יוכלו גורמי צד שלישי, כדוגמת שירותים כמו Skyscanner, לתקשר עם המערכת ולבצע הזמנות עבור לקוחות אפשריים. המערכת תבנה בצורה שתאפשר horizontal scaling, בדגש על ריצה בסביבה של ענן (מאשר שרתים ייעודיים).

הפרויקט ישתמש כמו כן בשפת התכנות Python וב-Node.js (JavaScript/Typescript) לכתיבת ה-services של המערכת, לאור השימוש בהן במדור. כמו כן, יהיה שימוש במסדי נתונים לשמירת המידע, במקרה זה PostgreSQL ו-MongoDB).

## 2.2. תהליך המחקר

תהליך המחקר לפרויקט התחלק לשלושה חלקים עיקריים:

1. לפני שהתחלתי את המחקר על הפרויקט, התכנון והעבודה עליו, למדתי על העקרונות של פיתוח לסביבת ענן. מנחה הפרויקט הכווין אותי ללמוד תחילה על Docker, עקרונותיו, הבעיות שהוא פותר, כיצד להשתמש בו, וכדומה. כמו כן, נתבקשתי גם לחקור על התחום של ארכיטקטורת microservices, הבעיה שהיא פותרת, עקרונות הקשורים בה (לדוגמא service discovery), ואופן פעולתה.

לאחר מכן, למדתי על Docker Compose, כיצד להשתמש בו על מנת לפשט עבודה מקומית עם קונטיינרים, ואף בניתי פרויקט קטן שמשתמש ב-Docker Compose ב-Python ו-React יחד עם בסיס נתונים של PostgreSQL (פרויקט מסוג user management system), על מנת לבסס את הידע בנושא.

לבסוף, למדתי גם לגבי Kubernetes, הסיבות לשימוש בו ואופן השימוש, וכחלק מהלמידה גם עשיתי deployment לפרויקט הקטן שתיארתי לעיל כתרגול.

1. לאחר בניית הבסיס בנושאים, השלב הבא היה מחקר לגבי התחום של הפרויקט – מערכות Computer Reservation Systems. מאחר שאין המון מידע בתחום באינטרנט, הייתי צריך לעשות סקירה ולחפש מקורות מידע שונים ברחבי האינטרנט, על מנת למצוא דוגמאות ולקרוא לגב אופן הפעולה של מערכות אלו, בדגש על אלו שנועדו לחברות טיסה קיימות. כמו כן, לדמתי לגבי תהליך ההזמנה של כרטיסי טיסה וכרטיסי עליה למטוס, וכיצד זה נעשה בדרך כלל מאחורי הקלעים.
2. לאחר המחקר הרב והמעמיק, עברתי לתכנון הפרויקט עצמו. מאחר שמדובר בפרויקט בעל סדר גודל בינוני עד גדול (למפתח יחיד), נדרש הרבה תכנון ראשוני ואיטרציות עליו לפני תחילת בנייתו. תכננתי את ארכיטקטורת המערכת, את ה-API שחלקים במערכת חושפים זה לזה, את הסכמות של בסיסי הנתונים, וגם את ה-REST API הציבורי שתחשוף המערכת. כמו כן, כתבתי תיעוד לחלקים רבים במערכת, במהלכו עברו איטרציות נוספות על התכנון של המערכת.

## 2.3. סקירת ספרות

הפרויקט עוסק בתחום בו לא קיימת ספרות רבה, אם בכלל. לכן, היה צורך לחפור במעמקי האינטרנט ולאסוף מידע מאתרים רבים על מנת לגבש דעה וידע כללי בכל הקשור למערכות CRS של חברות תעופה. לפיכך, אין לי מידע נוסף לספק בפרק זה, שכן חלק גדול מהפרויקט והאופן בו הוא בנוי נובעים מצורת החשיבה והתכנון שלי בלבד.

## 2.4. אתגרים מרכזיים

### 2.4.1. הבעיות איתן התמודדתי

לאורך תכנון ובניית הפרויקט עלו בעיות שונות מסוגים שונים.

הבעיה העיקרית איתה התמודדתי בשלב תכנון המערכת הוא הפיזור והמחסור במידע לגבי מערכות CRS באינטרנט, דבר שדרש ממני לאסוף מידע מהרבה מקומות, וכן גם להשלים את החוסרים בעצמי.

כמו כן, עלו בשלב התכנון בעיות כמו ארכיטקטורת המערכת, שדרשה מספר איטרציות עד שהגיעה לאופן בו היא פועלת כיום. הסיבה לכך היא מורכבותה, ודרישות או צרכים שעלו בזמן כתיבת הפרויקט.

קושי נוסף הוא השימוש בטכנולוגיות רבות, שדרש ממני להשקיע גם זמן רב ללמידתן ואופן פעולתן באופן מעמיק, לצד מגבלות הזמן הקיימות לביצוע הפרויקט. הדבר דרש ממני השקעה ותכנון רב לפני בניית חלקים בפרויקט גם בזמן בנייתו, על מנת לשלב אותן בצורה הטובה ביותר ולמנוע בעיות עתידיות.

לבסוף, בעיה שעלתה היא ה-deployment של הפרויקט ל-OpenShift (הפצה של Kubernetes של Red Hat עם תוספות), שבמהלכה עלו שגיאות שונות הקשורות להרשאות ודרשו ממני למצוא דרכים לעקוף אותן או למצוא פתרונות אחרים (בעיות שעלו עם השימוש במסדי הנתונים בעיקר).

### 2.4.2. הסיבות לבחירת הנושא

כפי שכבר ציינתי בפרקים קודמים, מטרת המערכת היא בעיקר להכין אותי לעבודה במדור, על מנת שאלמד ואצבור ניסיון בטכנולוגיות בהן עושים שימוש שם. עם זאת, הבחירה בבניית מערכת CRS לחברת תעופה היא בחירה אישית.

בחרתי דווקא במערכת כזו מאחר שפרויקט זה גרם לי להתנסות בכל הקשור של פיתוח מערכת גדולה, הרי שהיא מנהלת את הנתונים של בעצמה לגמרי, ואינה תלויה במערכת חיצונית לעבודתה.

כמו כן, מערכות מסוג זה לרוב מיושנות ולא בנויות לעבודה בענן, כך שזאת הייתה הזדמנות לנסות לראות כיצד אני יכול לפתח מערכת כזו בסביבת ענן.

### 2.4.3. מוטיבציה לעבודה

אני אדם שאוהב ללמוד ולהתנסות בדברים חדשים, במיוחד בעולם פיתוח התוכנה. לכן, כאשר הוצגה בפני ההזדמנות לעבוד על פרויקט מסוג כזה, התרגשתי מאוד לקראת הדרך שעבוד לאורך המחקר, הפיתוח והלמידה, בנוסף לניסיון שאצבור שיסייע לי גם בזמן הישרות וגם לאחריו.

לא אשקר ואגיד שלא היו שלבים שהרגשתי חסר מוטיבציה, אבל מאחר שהפרויקט רחב ממדים ומשתמש בכל כך הרבה טכנולוגיות, תמיד הרגשתי שיש עניין פרויקט ורציתי להמשיך לעבוד עליו, כך שבפן הכולל של הדברים, הייתי מלא מוטיבציה לאורך הפרויקט, ואפילו עכשיו יש עדיין דברים שאני ממשיך ללמוד הקשורים אליו.

### 2.4.4. הצורך בפרויקט

הפרויקט אינו מספק צורך קיים כלשהו למערכת מסוג זה, במיוחד שלא עבור הצבא. לפיכך, אפשר לומר שמטרתו היא ללמידה וצבירת ניסיון בלבד.

למרות זאת, הפרויקט מספק פתרון של מערכת CRS לחברת תעופה שהיא scalable שנבנתה לרוץ בענן, מה שיכול לסייע למערכות שאמורות להתפרס על פני משתמשים ולקוחות רבים.

### 2.4.5. הפתרונות לבעיה

מתחילת הפרויקט היו דרישות ספציפיות לגבי ריצתו, ביניהן שימוש ב-Kubernetes על מנת להפוך את הפרויקט שיהיה scalable. למרות זאת, כן היה צורך למצוא טכנולוגיות שונות לניהול היבטים שונים במערכת:

1. היה צורך במערכת לניהול תורים על מנת לנהל הודעות בין services במערכת, על מנת שתהיה כמה שיותר אמינה ולמנוע מ-services להיות תלויים לגמרי אחד בשני בצורה ישירה. לשם כך, נבחנו אופציות שונות, בהן: Redis, Kafka ו-RabbitMQ. לבסוף, בחרתי להשתמש ב-RabbitMQ, לאור אופן פעולתו, שכן הוא תאם לאופן מעבר המידע בין ה-services בפרויקט.
2. היה צורך במסדי נתונים לניהול הנתונים במערכת. לשם כך נבחנו כל מיני סוגים של מסדי נתונים, אך החלטתי לבסוף על שניים. הראשון הוא PostgreSQL, שכן הוא ידוע כמסד נתונים אמין שעומד טוב בכמות נתונים גדולה, וגם כי משתמשים בו אצלנו במדור. השני הוא MongoDB, לאור האופן בו הוא שומר נתונים במסמכים, שתאם לאופן בו נשמרים Passenger Name Records (PNR) במערכת.

# 3. מטרות ויעדים

הפרויקט היה צריך לעמוד במספר יעדים שנקבעו מראש, וגם כאלו שעלו תוך כדי במהלך בנייתו.

מבחינת היעדים הטכניים הכלליים של הפרויקט:

* מערכת שתהיה scalable אך גם אמינה.
* מערכת שיכולה לרוץ בענן ובנויה סביב קונטיינריזציה וריצה ב- Kubernetes cluster.
* חשיפת REST API ציבורי מאובטח.
* כתיבת בדיקות יחידה ל-services במערכת.
* כתיבת deployment manifest files עבור Kubernetes על OpenShift.
* דוקומנטציה רחבה למערכת (גם סטטית וגם אינטראקטיבית דרך Swagger UI).
* שימוש ב-Git ו-GitHub בשביל version control.

מבחינת היעדים של השימוש במערכת:

* חיפוש טיסות בתאריכים נתונים וקבלת הפרטים שלהן (פרטי טיסה, זמינות הטיסה והמקומות בה, צורת הישיבה במטוס, וכו').
* הזמנת טיסה.
* גישה מאובטחת להזמנה (צפיה בהזמנה, עדכון פרטי ההזמנה, וביטול ההזמנה).
* ביצוע צ'ק-אין מקוון לטיסה.
* שליחת אי-מיילים כתגובה לפעולות שעשו המשתמשים (הזמנת טיסה, צ'ק-אין, וכדומה).

# 4. אתגרים

במהלך פיתוח הפרויקט עלו מספר אתגרים. האתגר המרכזי שעלה הוא עיצוב המערכת ושמירת ה-services שיוכלו לעבוד באופן עצמאי, כך שיהיה ניתן להרחיב את המערכת על ידי horizontal scaling מבלי לפגוע בשלמותה.

אתגר נוסף שעלה היה ניהול הזמן ביחד לכמות העבודה, שכן הפרויקט דרש ממני להשקיע זמן רב בלמידה ובפיתוח של הפרויקט, כך שהיה צורך שאוותר על דברים מסויימים על מנת לסיים את הפרויקט בזמן. עם זאת, הצלחתי לעמוד בזמנים לאורך רוב הפרויקט, אם כי לקראת הסוף נוצרה בעיה של זמנים עקב הערכה לא נכונה שלי לכמות הזמן שייקח לפתח חלקים מסויימים במערכת, דבר שגרם לוויתור על דברים מסויימים.

האתגר האחרון שעלה הוא לגרום למערכת לרוץ בענן (OpenShift cluster), מה ששונה מאוד מעבודה מקומית על פרויקט ודורש הסתגלות.

# 5. מדדי הצלחה

מדדי ההצלחה שלי בפרויקט מורכבים מכמה היבטים.

בהיבט הטכני, הפרויקט מוצלח כל עוד המערכת רצה בצורה מאובטחת וללא בעיות ושגיאות פנימיות, ושהיא תהיה אמינה כמה שאפשר גם לאור נתונים שגויים שיכולים להיכנס מהמשתמש.

בהיבט האמינות, הפרויקט מוצלח אם קיימים מספיק בדיקות אוטומטיות שיכולות לבדוק את תקינותו, וכן, שניתן להרחיב אותו כמה שצריך ללא בעיה.

בהיבט הספרותי, שהפרויקט יהיה מתועד לגמרי, ובמיוחד ה-API החיצוני שמוצר למשתמשי המערכת, מה שיתרום גם למשתמשי המערכת שירצו להשתמש בה וגם לפיתוח עתידי.

# 6. רקע תיאורטי

## 6.1. רקע כללי

מערכת הזמנות ממוחשבת (Computer/Central Reservation System או (CRS היא מערכת שמטרתה לנהל את ההזמנות והעסקאות של ארגון, כמו חברות טיסה, מלונות, וכדומה. מערכות אלו קיימות כיום בכל מקום, בין אם ארגונים גדולים כמו חברות טיסה בינלאומיות ובין אם בתי קולנוע מקומיים.

מערכות הזמנות ממוחשבות הופיעו לראשונה בקרב חברות התעופה, שחיפשו פתרון ממוחשב על מנת לייעל את תהליך ההזמנות לטיסה. בעקבותיהן הלכו גם המלונות ושירותים דומים, עד שהפך הדבר לנפוץ כפי שהוא היום.

על מערכות אלו להיות בעלות זמינות גבוהה (high availably) עם סובלנות גבוהה לתקלות (fault tolerance), ולכם כך על המערכת להיות סקלבילית (scalable) על פי דרישה לשירותיה ולעומסים.

לתהליך ההזמנה של טיסה והעליה אליה קיים סדר קבוע וברור, המוכר לכל מי שנוהג לטוס. אפרט על אופן פעולת התהליך:

Diagram

Description automatically generated

שלבים בתהליך:

1. **חיפוש טיסות** – המשתמש בוחר שדה תעופה למוצא ושדה תעופה של היעד עבור הטיסה שהוא מחפש בין טווח תאריכים שהוא בוחר. כמו כן הוא יכול לבחור גם במספר הנוסעים לטיסה. לאחר שהוא מבצע חיפוש, יקבל המשתמש רשימת טיסות זמינות לאותו המועד, מהן יוכל המשתמש לבחור ולבצע הזמנה.
2. **הזמנת הטיסה** – לאחר בחירת הטיסה, יבצע המשתמש הזמנה של הטיסה, עם פרטיו האישיים והמקומות בהם בחר לטיסה. לאחר שיבצע את ההזמנה הוא יקבל מספר הזמנה (PNR locator או Booking number) המזהה את הזמנתו. באמצעות מספר ההזמנה יוכל המשתמש לבדוק בכל עת את מצב ההזמנה.
3. כרטיס הטיסה הדיגיטלי (e-ticket) ישלח לנוסע באימייל לאחר כמה זמן (לרוב עד 3 ימים ברוב חברות הטיסה), שם יהיו פרטי ההזמנה השונים. רק לאחר שהתקבל כרטיס הטיסה ניתן יהיה לעשות צ'ק-אין.
4. **צ'ק-אין מקוון (online check-in)** – הצ'ק-אין לטיסה נפתח 72 שעות לפני זמן העלייה למטוס. תהליך זה הכרחי על מנת לקבל את כרטיס העלייה למטוס (boarding pass) – בלעדיו לא יתאפשר לעלות על הטיסה! בתהליך הצ'ק-אין על הנוסעים יהיה למלא פרטים חסרים (כמו מספר דרכון וכו').
5. **עליה למטוס** – הנוסעים יגיעו לשדה התעופה עם כרטיס העלייה למטוס שקיבלו (דיגיטלי או מודפס), יבצעו את ההליכים הדרושים בשדה, ויעלו על הטיסה בזמן שצוין בכרטיס העלייה למטוס.

קיימים גם מספר מונחים שכדאי לדעת הקשורים לתחום של ה-CRS של חברות התעופה:

* **Passenger Name Record (PNR)** – קובץ הנוצר עבור כל הזמנה, המתאר את פרטי כל הנוסעים וכן פרטים נוספים (כמו פרטי התקשרות, מצב ההזמנה, וכדומה). לאור זאת, ניתן לומר ש-PNR הוא מונח טכני המקביל להזמנה (booking).
* **PNR Locator (Booking Number)** – מספר המזהה את ה-PNR שנוצר עבור ההזמנה. מספר זה הכרחי על מנת לבדוק את מצב ההזמנה, לבצע שינויים בהזמנה, לבצע צ'ק-אין לטיסה, וכדומה.
* **E-ticket** – כרטיס (מסמך) דיגיטלי המתקבל לאחר אישור ההזמנה (במציאות לאחר בדיקת פרטים וחיוב האשראי) המתאר את ההזמנה ופרטיה. בין הפרטים העיקריים שמכיל ה-e-ticket הוא מספר ההזמנה (PNR locator) שיאפשר לבצע check-in בשלב מאוחר יותר.
* **צ'ק-אין (Check-in)** – שלב שניתן לביצוע או באופן מקוון (online check-in), לרוב בין 24-72 שעות לפני הטיסה, או בשדה התעופה לפני העלייה למטוס. בשלב זה נבדקים הפרטים האישיים של הנוסעים (למשל מספר דרכון, תאריך לידה, וכולי) לצורכי אימות, וכן ניתנת האפשרות לנוסעים לרוב לבחור מקומות (עבור תשלום נוסף), להגדיל את כמות הכבודה, להוסיף אוכל לטיסה, וכולי. לבסוף, יונפק עבור על נוסע כרטיס עליה למטוס (boarding pass). למעשה ללא שלב זה לא ניתן יהיה לקבל את כרטיס העלייה למטוס, וכן, לא יהיה ניתן לעלות לטיסה.
* **כרטיס עליה למטוס (Boarding Pass)** – הכרטיס באמצעותו נוסע עולה למטוס. הכרטיס יכיל פרטים כמו מושב בטיסה, זמן העלייה למטוס, פרטי נוסע, טרמינל, וכדומה. כרטיס זה יכול להיות או דיגיטלי על ידי ביצוע צ'ק-אין מקוון או מודפס אם בוצע בצ'ק-אין בשדה התעופה.

## 6.2. רקע טכנולוגי

בפרויקט יש שימוש בטכנולוגיות רבות. להלן הסברים על הטכנולוגיות השונות ומונחים קשורים:

* **Microservices Architecture** – ארכיטקטורה שנולדה מתוך ארכיטקטורת SOA (Service Oriented Architecture), המורכבת מ-microservices – שירותים/תהליכים קטנים, המטפלים בחלק מסוים של המערכת, ומתקשרים זה עם זה על מנת להרכיב מערכת שלמה. ארכיטקטורה זו נוגדת את הארכיטקטורה המונוליתית, בה נבנה שרת אחד מרכזי המטפל בכל המערכת.

היתרון העיקרי בשימוש ב-microservices הוא ה-scalability של המערכת בתוצאה מהשימוש בהם: היות שכל אחד מהם מבטא יחידת עבודה, ניתן לבצע horizontal scaling (למעשה יצירת עותקים שרצים במקביל של אותו microservice) בקלות לחלקים ספציפיים במערכת שקיים עליהם עומס. כמו כן, הוא מאפשר לארגונים גדולים לחלק את העבודה לצוותים קטנים יותר, כך שכל צוות יעבוד על microservice אחד או יותר, בלי תלות בשאר הצוותים.

* **API Gateway** – דרך לחשוף API של מערכת לגורם חיצוני מבלי לתת גישה ישירה לחלקים במערכת. בארכיטקטורת microservices ה-API gateway ממומש ברך כלל על ידי שימוש ב-reverse proxy (למשל Nginx, Envoy וכדומה) המתקשר עם ה-microservices של המערכת.
* **Docker** – פלטפורמה המאפשרת הרצת קונטיינרים (containers) ובניית תמונות (images) ליצירתם. קל לחשוב על קונטיינר כתהליך מבודד עם מערכת קבצים משלו (כמו מכונה וירטואלית קלה), בעוד שתמונה מתארת כיצד ניתן ליצור את התהליך ואת מערכת הקבצים שלו. בפרויקט, כל microservice נארז כתמונה נפרדת וירוץ בתוך קונטיינר נפרד המתקשר עם שאר ה-microservices על גבי הרשת.
* **Kubernetes** – מערכת שפותחה על ידי גוגל המשמשת לניהול ופריסה אוטומטית של קונטיינרים (אורכסטרציה של קונטיינרים – container orchestration). למעשה היא זו שתאפשר לקשר בין כל חלקי הפרויקט השונים ותעזור לממש את עקרונות הזמינות הגבוהה, סובלנות לתקלות וסקלביליות. טכנולוגיה זו מאוד מסובכת להסבר לעומק לאור הגודל שלה והדברים בה היא תומכת, כך שאני ממליץ מאוד לחפש עליה באינטרנט.
* **Cluster** – מונח כללי שמתאר אוסף (או אשכול) של מחשבים שנמצאים ברשת אחת ומתקשרים זה עם זה. בהקשר של Kubernetes cluster, הכוונה לאוסף מחשבים שמנהל אותו Kubernetes. לרוב Kubernetes cluster יכיל מספר namespaces שמהווים מערכות לוגיות נפרדות (למשל פרויקטים) שנמצאות ב-cluster, מעין תת-cluster וירטואליים. ליחידת מחשוב אחת ב-cluster קוראים **node** (צומת), שכן הוא חלק מהרשת/הגרף שיוצר ה-cluster.
* **RabbitMQ** – פלטפורמת תיווך הודעות (message broker) המשמשת לניהול תורים תחת פרוטוקולים שונים, בעיקרAdvanced Message Queuing Protocol (AMQP). מטרת השימוש במתווך הודעות היא מניעת עומסים על ה-backend על ידי דחיית עבודות שלוקחות זמן ובכך לאפשר זמינות גבוהה יותר של המערכת ומניעת שגיאות במקרה של תקלות במערכת.למעשה, היא משתמשת כאיש ביניים, כך ששירותים במערכת לא תלויים זה בזה ישירות, אלא עבודות שיכולות להתבצע במועד מאוחר יותר נשלחות לתור ומתבצעות ברקע.
* **PostgreSQL** – מערכת לניהול בסיסי נתונים יחסי (RDBMS) אמינה ומוכרת, שטובה עם עבודה עם נתונים רבים.
* **MongoDB** – מערכת לניהול בסיסי נתונים לא יחסי (NoSQL database) המשתמשת במסמכים (documents) דמויי JSON לאחסון הנתונים באוספים (collections), בניגוד למסדי נתונים רציונליים המשתמשים בטבלאות לייצוג הנתונים.
* **REST API** – סגנון ה-REST (Representation State Transfer) API לחשיפת ממשקי משתמש ברשת בנוי לרוב על גבי פרוטוקול HTTP. הרעיון העומד מאחורי REST API הוא ממשק משתמש חסר state, העובד על הרעיון של resources, שהם יחידות מידע/משאבים שונים אותן מנהל ה-API.
* **GraphQL** – ספציפיקציה שקמה לאחר השימוש הרחב ב-REST לחשיפת ממשקים כדרך לייעול השגת הנתונים על ידי הקטנת מספר הבקשות לשרת. שימוש ב-GraphQL טוב מאוד כאשר יש שימוש רחב בנתונים יחסיים הקשורים זה בזה, ומכאן שמו.

# 7. תיאור מצב קיים

כיום לרוב המוחלט של חברות התעופה יש מערכת Passengers Service System (PSS), הכוללת מערכות שונות, בהן מערכת ה-Computer Reservation System (CRS), שזהו התחום בו עוסק הפרויקט שלי. רוב החברות משתמשות במוצרים קיימים מאשר בפתרונות חדשים. דוגמאות למערכות PSS/CRS קיימות בשוק:

* **Amadeus** –הפופולרית ביותר בקרב חברות התעופה, בהן Air France, Lufthansa, אל על, ועוד רבות.
* **Sabre** – נמצאת בשימוש על ידי American Airlines, Alitalia, וכו'.

קיימות עוד מערכות רבות בשוק, כשכמעט כולן קמו עוד לפני שנות ה-2000. לפיכך, אפשר להיות כמעט לגמרי בטוחים שאין כיום מערכת CRS/PSS פופולרית שעושה שימוש ב-Kubernetes בענן, שכן זוהי טכנולוגיה חדשה יחסית, ששוחררה רק ב-2014.

# 8. ניתוח חלופות מערכתיות

|  |  |  |
| --- | --- | --- |
| **חלופה** | **יתרונות** | **חסרונות** |
| פיתוח מערכת CRS חדשה שבנויה לרוץ בענן עם שימוש ב-Kubernetes. | * תכנון והתאמת המערכת מההתחלה לריצה בסביבת ענן. * תמיכה בשימוש בקונטיינרים בלי קונפיגורציה נוספת. * ערך לימודי רב, במיוחד בתחום הפיתוח. * פתרון חינמי. | * זמן פיתוח ארוך יותר מאשר התעסקות בקונפיגורציה. * להמציא את הגלגל מחדש לאור הקיום הרחב של מערכות קיימות יציבות. |
| המרת מערכת CRS קיימת לריצה בענן עם שימוש ב-Kubernetes. | * צורך מינימלי בפיתוח (אם בכלל) למערכת קיימת. * מערכות קיימות ידועות ביציבותן. | * קושי בהמרת מערכות שמיועדות לריצה על מערכות הפעלה או מכונות וירטואליות לריצה על קונטיינרים, במיוחד אם מדובר במערכות מונוליתיות. * מערכות קיימות יקרות מאוד, ולא נועדו לשימוש יחידני. * הזמן שייקח להמיר מערכת קיימת לריצה על סביבה כמו Kubernetes לא מצדיק מספיק את השימוש בסביבת ענן מלכתחילה. * ללא ערך לימודי רב. |

# 9. החלופה המערכתית הנבחרת

החלופה בה בחרתי היא כמובן החלופה הראשונה – פיתוח מערכת CRS חדשה שבנויה לרוץ בענן עם שימוש ב-Kubernetes. חלופה זו הרבה יותר ריאליסטית מבחינת היקף הפרויקט והמטרה שלו, שכן הוא יספק עבורי הכי הרבה ניסיון, הן בתחום הפיתוח והן בתחום ה-DevOps.

החלופה השנייה לא ישימה כלל, שכן מערכות קיימות מיועדות לחברות גדולות ולא לשימוש אישי, בנוסף לכך שהן עולות הרבה כסף. כמו כן, אין טעם בהמרת מערכת קיימת ומוכחת במשך שנים לריצה בסביבה שהיא לא יועדה אליה, כמו שאין טעם להמיר פרויקט שרץ על הענן לריצה על שרתים ייעודיים או מכונות וירטואליות.

# 10. אפיון המערכת

## 10.1. ניתוח דרישות המערכת

למערכת דרישות בתחומים שונים, כך שאתייחס לכל תחום בנפרד.

מבחינת פריסת התוכנה (software deployment):

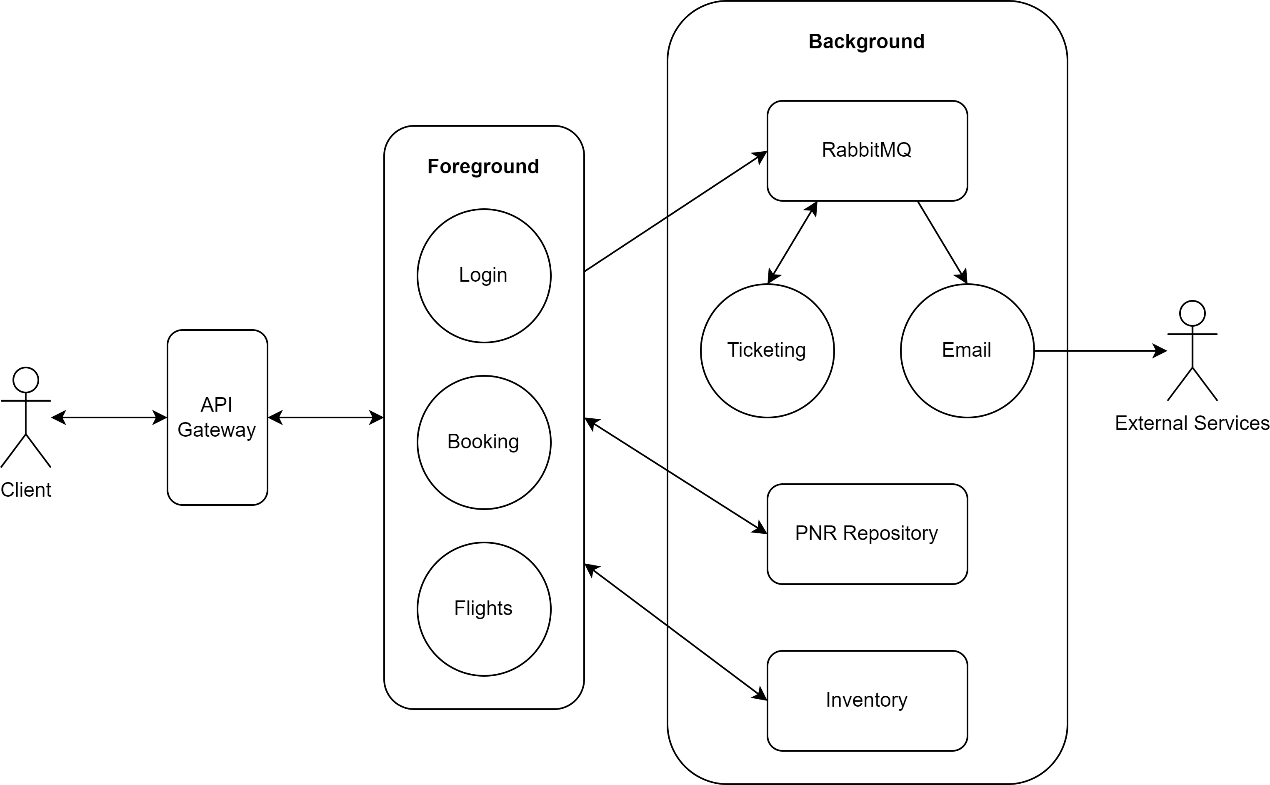
* שימוש בארכיטקטורת microservices למערכת.
* אריזת ה-microservices כתמונות והרצתם בקונטיינרים.
* שימוש ב-Kubernetes (OpenShift ספציפית) לאורכסטרציית קונטיינרים.
* נובע מהנקודות הקודמות – מערכת המאפשרת הרחבה לרוחב (horizontal scaling) ועמידות לתקלות (fault tolerance) המסוגלת לעמוד בעומסים.

מבחינה פונקציונלית:

* חיפוש טיסות בתאריך נתון בין מקור ויעד נתונים.
* קריאת פרטי טיסה (סדר הישיבה במטוס, מקומות פנויים, וכדומה).
* הזמנת טיסה (ומקומות בטיסה).
* ביצוע כניסה מאובטחת עבור מתן גישה לפרטים אישיים.
* צפייה בהזמנה (גישה מאובטחת).
* עדכון פרטי הזמנה (גישה מאובטחת).
* ביטול הזמנה (גישה מאובטחת).
* ביצוע צ'ק-אין מקוון גישה מאובטחת).
* שליחת אי-מיילים כתגובה לפעולות שעשו המשתמשים בצורה אוטומטית (הזמנת טיסה, צ'ק-אין, וכדומה).

## 10.2. מודולים במערכת

להלן תרשים המתאר את המודולים המרכיבים את המערכת. יש לשים לב שקיים דמיון בין תרשים המודולים הלוגיים לתרשים ארכיטקטורת המערכת שבהמשך, שכן לאור השימוש בארכיטקטורת microservice, שמקדמת חלוקת אחריות ועבודה לפי מודולריזציה, נוצר דמיון שכזה.



## 10.3. אפיון פונקציונלי

כפי שניתן היה לראות בסעיף הקודם, המודולים מתחלקים לשתי קטגוריות: מודולי חזית (foreground), שלמעשה מטפלים ישירות בבקשות מהמשתמש (שמתקבלות דרך ה-API Gateway), ולמודולי רקע, שאחראיים לעבודות ברקע, כמו ניהול תורים, שמירת נתונים, עבודות רקע, וכו'.

להלן הפירוט על מודולי החזית:

* **Login** – מטפל באימות (authentication) עבור המערכת, ודואג לייצר authentication tokens עבור המשתמשים, על מנת שיוכלו לגשת לחלקים מאובטחים במערכת.
* **Booking** – מטפל בהזמנת הטיסות וניהולן, וגם בבקשות צ'ק-אין לטיסות. עבור מידע רגיש, כמו גישה ושינוי פרטי הזמנה, ביטול הזמנה, וביצוע צ'ק אין לטיסה, המודול משתמש ב-token שיצר מודול ה-Login לשם אימות ומתן הרשאות גישה.
* **Flights** – חושף מידע של המערכת לגבי טיסות, ומאפשר חיפוש טיסות על פי פרמטרים נתונים. אחראי להציג למשתמש נתונים הקשורים בטיסות הזמינות ולממשק את המידע הקיים במערכת לגבי הטיסות השונות.

להלן הפירוט על מודולי הרקע:

* **RabbitMQ** – אחראי לניהול תורים בהם משתמשים שאר מודולי הרקע. באמצעותו מעבירים מודולי החזית עבודות לביצוע על ידי מודולי הרקע.
* **Ticketing** – אחראי לתהליך התיקוף (ticketing) של הזמנות שבוצעו ועדכון מידע רלוונטי הקשור לתיקוף ההזמנות. בעולם האמיתי, מודול זה אחראי גם לתקשור עם מודולים נוספים ושירותים חיצוניים, כמו חברות האשראי על מנת לחייב את הלקוחות.
* **PNR Repository** – אחראי לניהול ושמירת ה-PNR-ים הנוצרים בזמן תהליך ההזמנה.
* **Inventory** – אחראי לשמירת ופרסום נתוני טיסה של המערכת לכל שאר המודולים.

## 10.4. ביצועים עיקריים

ביצועי המערכת לא נבדקו מאחר עקב מחסור במשאבים פיזיים. מאחר שהפרויקט בנוי להתרחב לרוחב (horizontal scaling) בתיאוריה, לא קיימת מגבלת ביצועים אמיתית מלבד מגבלות פיזיות של התשתית עליה יושבת המערכת.

המודולים שכן יכולים לגרום לצוואר בקבוק הם האחראיים על שמירת הנתונים והפצתם, מאחר שלא תמיד פשוט להרחיב שירותים העוסקים ב-persistence, כמו שירותי מסדי נתונים, אך גם לכך יש פתרונות שונים, כמו database sharding.

במקרה הספציפי של הפרויקט שלי, ועם המגבלות הקיימות מבחינת תשתית שיש לי לבדיקה, אין ירידה בביצועים ברורה קיימת בבדיקות מערכת שעלו.

## 10.5. אילוצים

מאחר שהפרויקט בנוי סביב קונטיינריזציה, לא קיימים אילוצים טכנולוגיים רבים, שכן טכנולוגיה זו מבודדת את המודולים כך שכל אחד רץ בסביבה שמתאימה לו, מבלי שום תלות בטכנולוגיה חיצונית.

האילוץ היחידי עבור הרצה מקומית של הפרויקט הוא התקנה מקומית של Docker (ו-Docker Compose, אך אפשרי גם בלי), ועבור הרצה בענן הוא שיהיה Kubernetes/OpenShift cluster קיים.

# 11. תיאור הארכיטקטורה

## 11.1. ארכיטקטורת המערכת

Diagram

Description automatically generated

## 11.2. תיאור הרכיבים במערכת

המערכת מורכבת מאוסף של microservices שמתקשרים זה עם זה ברשת. חלק מהשירותים איתם מתקשרים ה-microservices הם שירותים שיכולים להיות חיצוניים למערכת, שיכולים לרוץ בקונטיינר במערכת, או יכולים להוות cluster נפרד משלהם (למשל במקרה של RabbitMQ) איתו מתקשר ה-cluster הנוכחי.

להלן תיאור לגבי כל רכיב/microservice בארכיטקטורה:

* **API Gateway** – חושף REST API ציבורי למערכת, וממומש על ידי שימוש ב-reverse proxy. במקרה של ריצה על גבי Kubernetes, יש שימוש ב-Ingress, שהוא רכיב Kubernetes שאחראי לקשר בין העולם החיצון אל ה-cluster.
* **Login Service** – מטפל באימות (authentication) עבור המערכת, על ידי בדיקת פרטי האימות עם הנתונים השמורים במערכת, ודואג לייצר authentication tokens עבור המשתמשים, על מנת שיוכלו לגשת לחלקים מאובטחים במערכת.
* **Booking Service** – מטפל הזמנת טיסות עבור משתמשים, ומאפשר גישה לנתוני ההזמנה וכן עדכון ההזמנה וביטולה. נוסף לכך, הוא מטפל בתהליך הצ'ק-אין לטיסות. הוא משתמש גם ב-authentication token שיוצר ה-login service בשביל אימות ומתן הרשאות גישה למשתמשים אל הנתונים הרלוונטיים.עבור פעולות מסוימות, הוא מתזמן גם שליחת אימיילים דרך ה-email service.
* **Flights Service** – חושף ממשק למציאת טיסות בתאריך מסוים עבור מקור ויעד נתונים ומסננים נוספים. כמו כן, מאפשר גם חיפוש מידע לגבי טיסה מסוימת, שכולל את פרטי הטיסה, המטוס, צורת הישיבה, מקומות פנויים וכדומה.
* **RabbitMQ** – שירות לניהול תורים במערכת – יכול לרוץ בקונטיינר, להיות שירות חיצוני, או אפילו להיות ב-cluster משלו. קיימים בו שני exchanges (למעשה רכיבים שאחראיים להעביר הודעות לתורות המתאימים): ticket exchange ו-email exchange. ה-ticket exchange מקבל הודעות לתיקוף הזמנות ומעביר אותן לתור המתאים (ticketing queue); ה-email exchange מקבל הודעות של אימיילים לשליחה ומעביר אותן לתור המתאים (email queue).
* **Ticketing Service** – אחראי על תיקוף ההזמנות ועדכון סטטוס ההזמנה ופרטים נוספים הקשורים בהזמנה. הוא מושך הודעות מה-ticketing queue של RabbitMQ, מעבד אותן, ומתזמן שליחת מייל עם כרטיסי הטיסה ל-email exchange של RabbitMQ.
* **Email Service** – אחראי על שליחת אימיילים שונים ללקוחות. הוא מושך הודעות מה-email queue של RabbitMQ ודואג לשליחת האימייל המתאים.
* **PNR Database** – שירות מסד הנתונים ששומר את ה-PNR-ים (ההזמנות) של המערכת – יכול לרוץ בקונטיינר, להיות שירות חיצוני, או אפילו להיות ב-cluster משלו.
* **Inventory** – שירות מסד הנתונים ששומר את כל נתוני הטיסות והנתונים הקשורים לאותן טיסות – יכול לרוץ בקונטיינר, להיות שירות חיצוני, או אפילו להיות ב-cluster משלו.
* **Inventory Manager** – אחראי על יצירת ממשק GraphQL עבור ה-inventory, בו ישתמשו כל ה-microservices במערכת.

## 11.3. ארכיטקטורת הרשת

ארכיטקטורת הרשת דומה מאוד לתרשים של ארכיטקטורת המערכת כאשר קיים עותק אחד בלבד של כל אחד מה-microservices במערכת/cluster. עבור מספר עותקים שונה, קשה לדעת שכן הרשת שבין הרכיבים ברשימת מנוהלת לגמרי על ידי ה-Kubernetes cluster, שכן אחת התכונות החשובות של Kubernetes הוא ניהול ה-cluster והרשת שהוא יוצר בצורה אוטומטית, בין אם מדובר בהקצאת כתובות IP, load balancing, וכדומה.

## 11.4. פרוטוקולי התקשורת

המערכת כולה משתמשת בפרוטוקולי תקשורת הבנויים על גבי פרוטוקול TCP של אוסף הפרוטוקולים TCP/IP. להלן פירוט של פרוטוקולי התקשורת השונים:

|  |  |  |
| --- | --- | --- |
| **פרוטוקול** | **תיאור** | **שימוש** |
| REST API (HTTP) | REST API הוא לא פרוטוקול כשלעצמו, אלא סגנון עיצוב API פופולרי המתבסס על הפרוטוקול הנפוץ HTTP. הוא משתמש בעקרונות של HTTP ומשמיש אותם לשימוש כללי עבור בקשות API. כיום הוא בין הסגנונות הפופולריים לעיצוב API מוכוון רשת. | חושפים REST API:   * API Gateway * Login Service * Booking Service * Flights Service |
| GraphQL (HTTP) | פרוטוקול ושפת שאילתות ומניפולציות לנתונים שפותחה על ידי פייסבוק. GraphQL לא מוגבלת לפרוטוקול מסוים, אך ברוב המקרים השימוש הוא על גבי פרוטוקול HTTP. פרוטוקול זה מאפשר שליטה טובה על הנתונים המתבקשים בצורה דקלרטיבית ופשוטה, ומונעת את הצורך במספר בקשות API על מנת להשיג נתונים קשורים. | חושפים GraphQL API:   * Inventory Manager   צורכים GraphQL API:   * Booking Service * Flights Service * Email Service |
| AMQP 0-9-1 | פרוטוקול עבור העברת הודעות והשמתן בתורים, ושמה הוא ראשי תיבות של Advanced Message Queuing Protocol. פרוטוקול זה נמצא בשימוש על ידי RabbitMQ על מנת קבלת והעברת הודעות לתורים וכדומה. זהו אחד מן הפרוטוקולים בהן תומך RabbitMQ, כאשר  AMQP 0-9-1 הוא הגרסה הרווחת יותר. | משתמשים ב-AMQP 0-9-1:   * RabbitMQ * Booking Service * Ticketing Service * Email Service |
| PostgreSQL Protocol | הפרוטוקול בו PostgreSQL עושה שימוש להעברת נתונים ברשת. נעשה שימוש בפרוטוקול זה כאשר יוצרים חיבור ומתקשרים עם שירות מסד הנתונים. | משתמשים בפרוטוקול:   * Inventory * Inventory Manager |
| MongoDB Protocol | הפרוטוקול בו MongoDB עושה שימוש להעברת נתונים ברשת. נעשה שימוש בפרוטוקול זה כאשר יוצרים חיבור ומתקשרים עם שירות מסד הנתונים. | משתמשים בפרוטוקול:   * PNR DB * Login Service * Booking Service * Ticketing Service * Email Service |

## 11.5. תיאור השרת והלקוח

בפרויקט קיים רק צד השרת, שהוא למעשה המערכת כולה. נכון לזמן כתיבת ספר זה, לא קיים צד לקוח, אם כי אפשרי הדבר בהחלט, מאחר שקיים API חיצוני למערכת. יכול להיות שלאחר כתיבת הספר אכתוב גם צד לקוח על מנת להראות כי אפשרי הדבר.

מבחינת צד השרת, כל microservice במערכת פותח ברובו לפי לוח הזמנים שהוגדר בהצעת הפרויקט (פרק 1). בשביל צד השרת השתמשתי בטכנולוגיות שונות כפי שכבר תואר בסעיפים ופרקים קודמים, וכפי שאתאר גם בפרק 15 בספר.

ה-microservices פותחו באופן אינדיווידואלי לפי סדר לוגי של פיתוחם, שכן קיימת תלות בין חלק מהם. למעשה אופי הפיתוח היה לפי מודל מפל המים (ולא agile), שכן בשביל הדרישות שהועלו, הארכיטקטורה, והעובדה שאני מפתח יחיד, זה היה הרבה יותר מתאים. אם היה מדובר בצוות שעובד על הפרויקט, היה אפשר גם לעבוד במודל ה-agile, שכן ניתן היה לבצע עבודה במקביל על חלקים רבים במערכת.

כמו כן, בזמן פיתוח הפרויקט עברתי לעבוד עם Trello על מנת לנהל את המשימות ולוח הזמנים של הפרויקט, שתרם לי רבות, וגם לימד אותי כיצד להשתמש בו לעתיד.

# 12. ניתוח מקרי השימוש של המערכת

## 12.1. תיאור מקרי השימוש במערכת

### 12.1.1. מקרה שימוש: Search for Flights

הלקוח שולח בקשה לחיפוש טיסות בתאריך מסוים, עבור מקור ויעד. המקום והיעד הם שדות תעופה, וניתנים על פי קודי ה-IATA שלהם (למשל TLV, JFK, BER, וכדומה).

הבקשה עוברת דרך ה-API Gateway אל ה-Flight Service. ה-Flights Service יוצר בקשה לחיפוש טיסות התואמות פרמטרים אלו אל ה-Inventory Manager.

להלן בקשת ה-GraphQL שמתבצעת:

query findFlights(

$origin: String!

$destination: String!

$from\_time: timestamptz!

$to\_time: timestamptz!

$passengers: bigint! = 1

$cabin\_classes: [String!]! = ["E", "B", "F"]

) {

service(

where: {

origin\_airport: { iata\_code: { \_eq: $origin } }

destination\_airport: { iata\_code: { \_eq: $destination } }

}

) {

id

origin\_airport {

...airportFragment

}

destination\_airport {

...airportFragment

}

flights(

where: {

departure\_time: { \_gte: $from\_time, \_lte: $to\_time }

available\_seats\_counts: { available\_seats\_count: { \_gte: $passengers } }

}

) {

id

departure\_terminal

departure\_time

arrival\_terminal

arrival\_time

aircraft\_model {

icao\_code

iata\_code

name

}

available\_seats\_counts(

where: {

cabin\_class: { \_in: $cabin\_classes }

available\_seats\_count: { \_gte: $passengers }

}

) {

cabin\_class

total\_seats\_count

available\_seats\_count

}

}

}

}

fragment airportFragment on airport {

iata\_code

icao\_code

name

subdivision\_code

city

geo\_location

}

הבקשה מחפשת את כל הטיסות **הזמינות** (שיש להן מושבים פנויים) היוצאות משדה המקור (הנתון כקוד IATA) לשדה היעד (הנתון כקוד IATA) בין טווח תאריכים שקובע ה-Flights Service, שהוא מהתאריך שהתקבל ועד 24 שעות לאחר מכן.

אם לא קיים שירות עבור המקור והיעד הנתונים, כלומר חברת התעופה לא תומכת בטיסות בין המקור והיעד כללו, ה-Flights Service יחזיר שגיאה מתאימה ללקוח. אחרת, ישלח פירוט של השירות הקיים והטיסות שיש לתאריך שהתבקש (אם יש טיסות בכלל).

### 12.1.2. מקרה שימוש: Get Flight Details

למקרה שימוש זה קיימים שתי סוגי בקשות. הסיבה לכך היא שאיחוד הבקשות לבקשה אחת יגרום לתגובה להיות מאוד גדולה, ויכלול מידע שלא תמיד רצוי. לכן, החלטתי לחלק את מקרה השימוש לשתי סוגי בקשות במערכת: אחת לבקשת פרטי הטיסה, ואחת לפרטי סדר הישיבה הנוכחי במטוס (seat map ומקומות תפוסים).

הלקוח שולח בקשה למציאת הפרטים של טיסה מסוימת לפי המזהה שלה (flight ID).

הבקשה עוברת דרך ה-API Gateway אל ה-Flight Service. ה-Flights Service יוצר בקשה לחיפוש הטיסה המתאימה והפרטים הרצויים אל ה-Inventory Manager.

להלן בקשת ה-GraphQL עבור בקשת פרטי הטיסה:

query getFlight($flight\_id: uuid!) {

flight\_by\_pk(id: $flight\_id) {

id

service {

id

origin\_airport {

...airportFragment

}

destination\_airport {

...airportFragment

}

}

departure\_terminal

departure\_time

arrival\_terminal

arrival\_time

aircraft\_model {

iata\_code

icao\_code

name

}

available\_seats\_counts {

cabin\_class

total\_seats\_count

available\_seats\_count

}

}

}

fragment airportFragment on airport {

iata\_code

icao\_code

name

subdivision\_code

city

geo\_location

}

הבקשה מחפשת את הטיסה הרצויה לפי המזהה שלה, ומחזירה את פרטי הטיסה הרצויים ללא סדר הישיבה. התגובה כן כוללת ערכים כלליים לגבי זמינות המושבים בטיסה (כמות המושבים, כמות המושבים הפנויים).

להלן בקשת ה-GraphQL עבור פרטי סדר הישיבה:

query getFlightSeats($flight\_id: uuid!) {

flight\_by\_pk(id: $flight\_id) {

id

aircraft\_model {

icao\_code

iata\_code

name

seat\_maps {

cabin\_class

start\_row

end\_row

column\_layout

}

}

booked\_seats {

seat\_row

seat\_column

}

}

}

הבקשה מחפשת את הטיסה הרצויה לפי המזהה שלה, ומחזירה את פרטי סדר הישיבה במטוס ורשימה של המקומות התפוסים.

בשני המקרים, אם לא קיימת טיסה עם המזהה הנתון ה-Flights Service יחזיר שגיאה מתאימה ללקוח. אחרת, ישלח תגובה עם הפרטים הרצויים לבקשת הלקוח.

### 12.1.3. מקרה שימוש: Book Flight

הלקוח שולח בקשה להזמנת טיסה. הבקשה כוללת את מזהה הטיסה והפרטים של הנוסעים, כולל המקומות בהם ישבו הנוסעים.

הבקשה עוברת דרך ה-API Gateway אל ה-Booking Service. ה-Booking Service יוצר בקשה לחיפוש הטיסה המבוקשת אל ה-Inventory Manager.

להלן בקשת ה-GraphQL שמתבצעת:

query findFlight($flight\_id: uuid!) {

flight\_by\_pk(id: $flight\_id) {

id

aircraft\_model {

seat\_maps {

cabin\_class

start\_row

end\_row

column\_layout

}

}

booked\_seats {

seat\_row

seat\_column

}

available\_seats\_counts {

cabin\_class

available\_seats\_count

total\_seats\_count

}

}

}

אם לא נמצאה הטיסה, מחזיר ה-Booking Service שגיאה ללקוח. אחרת, הוא בודק אם המקומות המבוקשים כבר תפוסים. אם כן, הוא מחזיר הודעה מתאימה ללקוח. אחרת, הוא שולח בקשה ל-Inventory Manager להזמנת המקומות.

להלן בקשת ה-GraphQL שמתבצעת:

mutation bookSeats($seats: [booked\_seat\_insert\_input!]!) {

insert\_booked\_seat(objects: $seats) {

returning {

id

cabin\_class

seat\_row

seat\_column

}

}

}

הבקשה שומרת את המקומות המבוקשים. אם הבקשה נכשלת, כי בזמן שבין הבקשות ל-Inventory Manager נתפסו המקומות, יחזיר ה-Booking Service שגיאה ללקוח.

אם הכל בסדר, הוא יתזמן את הטיסה לתיקוף על ידי RabbitMQ וגם יתזמן שליחת אימייל אישור הזמנה באותו האופן. לבסוף, ישלח תגובה עם פרטי ההזמנה שבוצעה ללקוח, המכילה את מספר ההזמנה (ה-Booking ID).

### 12.1.4. מקרה שימוש: Log In

הלקוח שולח בקשת התחברות, הכוללת את מספר ההזמנה שקיבל בעת הזמנת הטיסה ופרטי האימות.

הבקשה עוברת דרך ה-API Gateway אל ה-Login Service. ה-Login Service מחפש האם ההזמנה קיימת ב-PNR database. אם ההזמנה לא קיימת, מקבל הלקוח שגיאה. אחרת, הוא בודק את פרטי האימות ביחס לפרטי ההזמנה (PNR) הקיימים. אם הפרטים לא תואמים, יקבל הלקוח שגיאה. אחרת, ייווצר ה-authentication token עבור הלקוח, וישלח לו כתגובה לבקשה.

### 12.1.5. מקרה שימוש: Get Booking Details

הלקוח שולח בקשת פרטי הזמנה עם מספר ההזמנה.

הבקשה עוברת דרך ה-API Gateway אל ה-Booking Service. ה-Booking Service בודק אם הלקוח ביצע התחברות (log in). אם לא היה מחובר הלקוח כשביצע את הבקשה (אין בבקשה את ה-authentication token), יקבל הלקוח שגיאה.

במקרה שהלקוח מחובר, ה-Booking Service מחפש את ההזמנה ב-PNR database ומחזיר את פרטיה ללקוח.

### 12.1.6. מקרה שימוש: Update Booking

הלקוח שולח בקשת עדכון פרטי הזמנה עם מספר ההזמנה ופרטי ההזמנה המעודכנים.

הבקשה עוברת דרך ה-API Gateway אל ה-Booking Service. ה-Booking Service בודק אם הלקוח ביצע התחברות (log in). אם לא היה מחובר הלקוח כשביצע את הבקשה (אין בבקשה את ה-authentication token), יקבל הלקוח שגיאה.

במקרה שהלקוח מחובר, ה-Booking Service מחפש את ההזמנה ב-PNR database. אם מספר הנוסעים בבקשה לא תואם לנוסעים שבהזמנה, יקבל הלקוח שגיאה. אם ההזמנה כבר בוטלה או שהלקוח כבר ביצע צ'ק-אין, לא ניתן לעדכן את ההזמנה ויקבל הלקוח שגיאה.

אם הכל בסדר, ה-Booking Service מעדכן את ההזמנה ושולח ללקוח תגובה עם ההזמנה המעודכנת.

### 12.1.7. מקרה שימוש: Cancel Booking

הלקוח שולח בקשת ביטול הזמנה עם מספר ההזמנה.

הבקשה עוברת דרך ה-API Gateway אל ה-Booking Service. ה-Booking Service בודק אם הלקוח ביצע התחברות (log in). אם לא היה מחובר הלקוח כשביצע את הבקשה (אין בבקשה את ה-authentication token), יקבל הלקוח שגיאה.

במקרה שהלקוח מחובר, ה-Booking Service מחפש את ההזמנה ב-PNR database. אם ההזמנה כבר בוטלה או שהלקוח כבר ביצע צ'ק-אין, לא ניתן לבטל את ההזמנה ויקבל הלקוח שגיאה.

אם הכל בסדר, ה-Booking Service יסמן את ההזמנה כמבוטלת ב-PNR database. כמו כן, הוא יעדכן את המושבים באמצעות בקשה ל-inventory manager.

להלן בקשת ה-GraphQL שמתבצעת:

mutation removeBookedSeats($seats: [uuid!]!) {

delete\_booked\_seat(where: { id: { \_in: $seats } }) {

returning {

id

cabin\_class

seat\_row

seat\_column

}

}

}

ה-Booking Service מתזמן גם אימייל ביטול הזמנה באמצעות RabbitMQ ולבסוף שולח ללקוח תגובה עם ההזמנה המבוטלת.

### 12.1.8. מקרה שימוש: Check In

הלקוח שולח בקשת צ'ק-אין עם מספר ההזמנה ופרטים הנדרשים לתהליך הצ'ק-אין. הלקוח לא נדרש לבצע צ'ק-אין לכל הנוסעים, כך שניתן לבצע את התהליך מספר פעמים לנוסעים שונים בנפרד.

הבקשה עוברת דרך ה-API Gateway אל ה-Booking Service. ה-Booking Service בודק אם הלקוח ביצע התחברות (log in). אם לא היה מחובר הלקוח כשביצע את הבקשה (אין בבקשה את ה-authentication token), יקבל הלקוח שגיאה.

במקרה שהלקוח מחובר, ה-Booking Service מחפש את ההזמנה ב-PNR database. אם ההזמנה כבר בוטלה או שההזמנה עוד לא תוקפה, לא ניתן לבצע את התהליך ויקבל הלקוח שגיאה. אם לפחות אחד מהנוסעים המבקשים לבצע צ'ק-אין כבר עברו את התהליך, יקבל הלקוח שגיאה, שכן לא ניתן לבצע את התהליך פעמיים לאותו הנוסע.

אם הכל בסדר, יעדכן ה-Booking Service את ההזמנה ב-PNR database, יתזמן את שליחתם של כרטיסי העלייה למטוס (boarding passes) לנוסעים שביצעו צ'ק-אין באימייל, וישלח ללקוח תגובה עם פרטי ההזמנה המעודכנים.

## 12.2. מקרי השימוש עבור הפונקציות העיקריות במערכת

ראה סעיף 12.1 שלעיל, שכן הוא מתאר את מקרי השימוש והפונקציות העיקריות בפרויקט.

## 12.3. מבני הנתונים שהושמשו

במערכת יש שימוש במספר רב של מבני נתונים, אך אתייחס לעיקריים בהם:

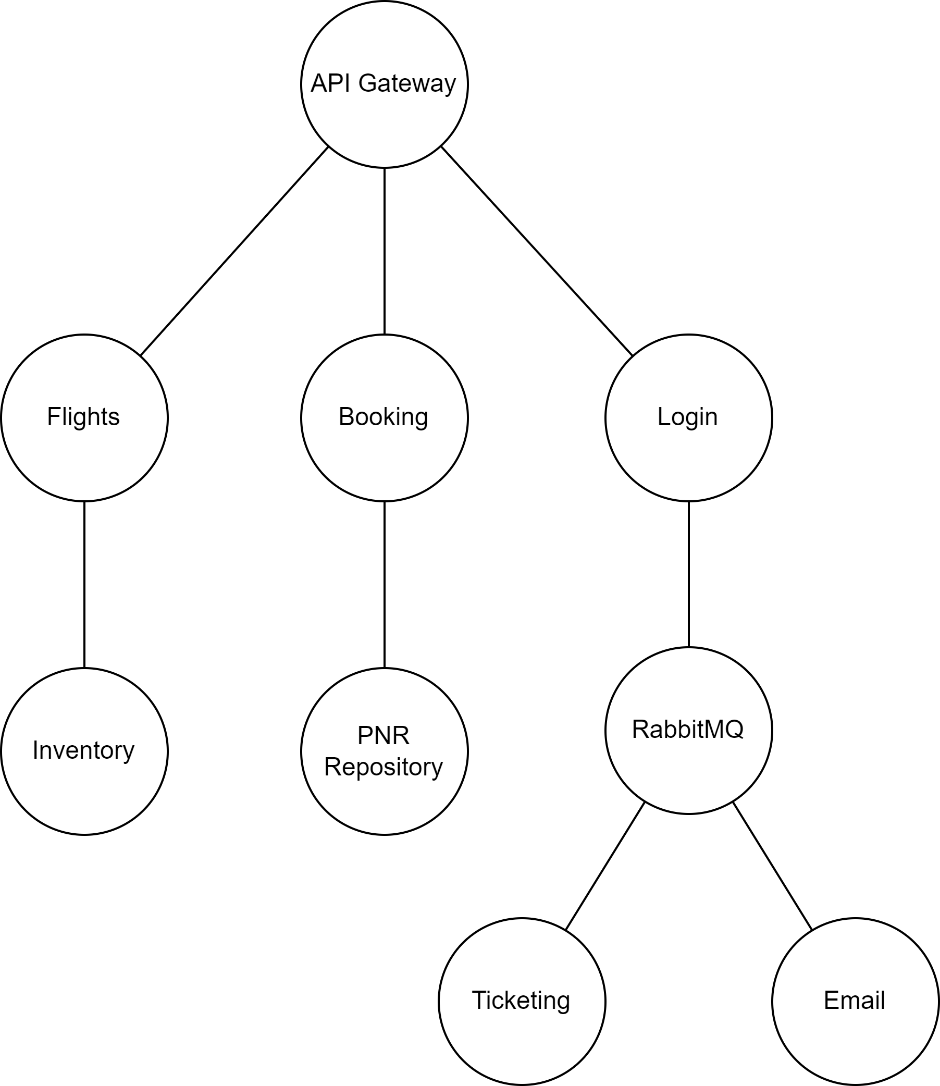
* **רשימה (list)** – בין מבני הנתונים הנפוצים. לרוב ממומש על ידי שימשו במערך או רשימה מקושרת, ומובנה בשפות התכנות בהן השתמשתי בפרויקט. מטרת מבנה הנתונים הוא לייצג אוסף של פרטים עם סדר קבוע.
* **מילון (Map/Dictionary/Associative Array)** – מבנה נתונים נפוץ, בו נעשה שימוש רחב ב-JavaScript, שכן אובייקטים בשפה זו הם למעשה מילון (קיים גם סוג שנקרא Map בשפה, שפועל בצורה מאוד דומה, אך טוב יותר לשימוש כללי ושמירת נתונים במילון). מבנה נתונים זה מקשר בין ערך מפתח (key) ל-value, ומאפשר מציאת ערכים לפי המפתח שלהם ביעילות טובה (בשימוש בטבלת גיבוב למשל היעילות היא O(1)).
* **קבוצה (Set)** – מבנה נתונים דמוי רשימה, שבדומה למבנה המתמטי ממנו בא שמו, שומר ערכים ייחודיים בלבד (אין עותקים). טוב למקרים בהם יש צורך של אוסף נתונים ללא חזרות או עותקים.
* **תור (Queue)** – מבנה נתונים מסוג FIFO (First In First Out), בו נתונים נשמרים בתור לוגי, כפי שאומר שמו. יש שימוש במבנה נתונים זה אצל RabbitMQ, והוא מושמש בפרויקט לניהול עבודות עתידיות.

## 12.4. הקשרים בין היחידות השונות

תרשים של הקשרים בין היחידות בפרויקט, שהם למעשה ה-microservices שבו, ניתן לראות בסעיף 11.1.

ההסברים על הקשרים שבין היחידות ניתן למצוא בסעיף 12.1.

## 12.5. עץ מודולים



## 12.6. דיאגרמת UML Use Case

Diagram

Description automatically generated

## 12.8. תרשים UML Class Diagram

מאחר שהפרויקט משתמש בארכיטקטורת microservices, ומאחר שהשפות בהן השתמשתי בפרויקט לא מוכוונות עצמים כמו Java או C# למשל, אתייחס לחלק זה כאפשרות להציג את ה-domain model של המערכת.

A screenshot of a computer

Description automatically generated with medium confidence

## 12.9. תרשים Deign Class

ראו סעיף 12.8 עבור תרשים זה.

## 12.10. תרשים מחלקות

ראו סעיף 12.8 עבור תרשים זה.

## 12.11. תיאור המחלקות המוצעות