Skyline

הרשת המדעית-טכנולוגית המובילה בישראל
מכללות ובתי־ספר לטכנולוגיה מתקדמת ולמדעים

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

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.

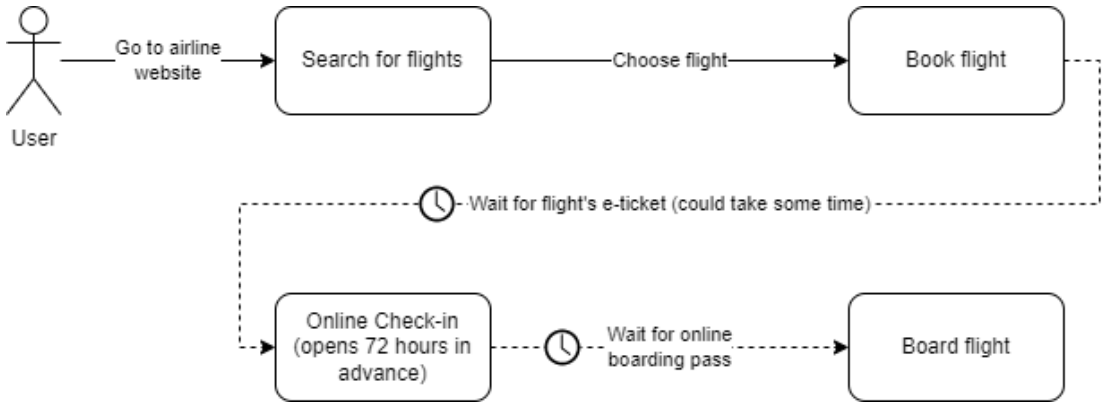
Skyline CRS היא מערכת לניהול טיסות עבור חברת תעופה בדיונית Skyline שמטרתה לנהל את כל הקשור לתהליך ההזמנה של הטיסה והעליה אליה. המערכת תעוצב בארכיטקטורת microservices על מנת לאפשר horizontal scaling של המערכת ובכך לממש את הדרישות של זמינות גבוהה, סובלנות לתקלות וסקלביליות.

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



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, בעיקר Advanced 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:**

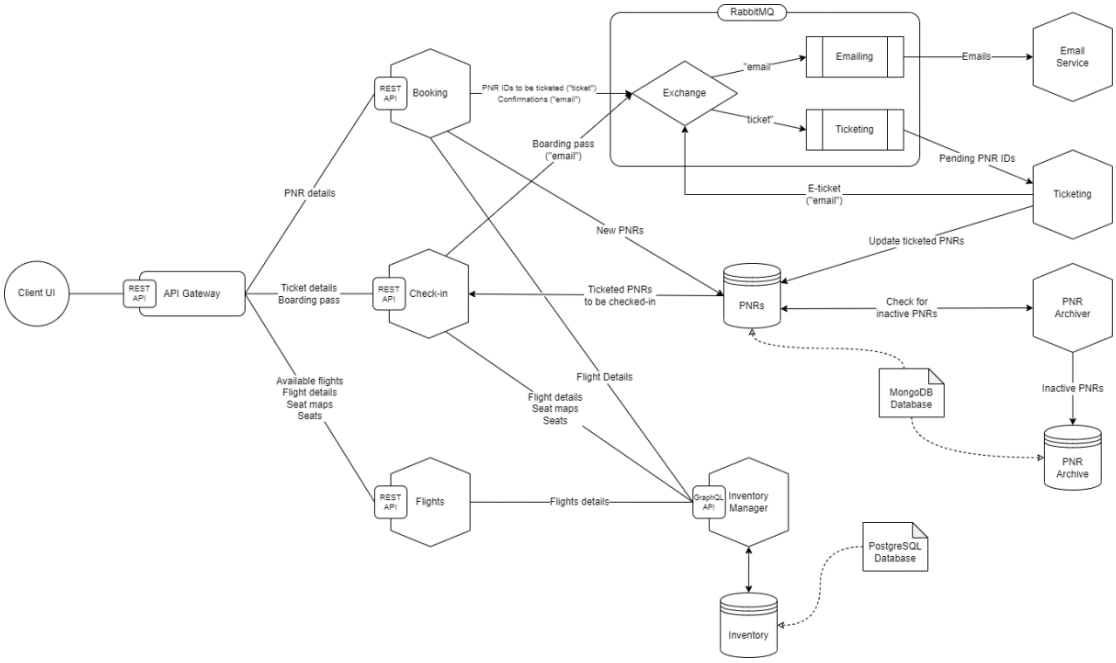
- **Express** – ספרייה ליצירת שרת web אסינכרוני החושף REST API**.** ספרייה זו תשמש ליצירת microservices החושפים REST API במערכת.
- **amqplib** – ספרייה להתממשקות עם RabbitMQ. תשמש microservices שצריכים לשלוח הודעות לתורים או לקבל מהם הודעות.
- **graphql-request** – ספרייה לתקשורת עם ממשקי GraphQL ויצירתם. 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:**

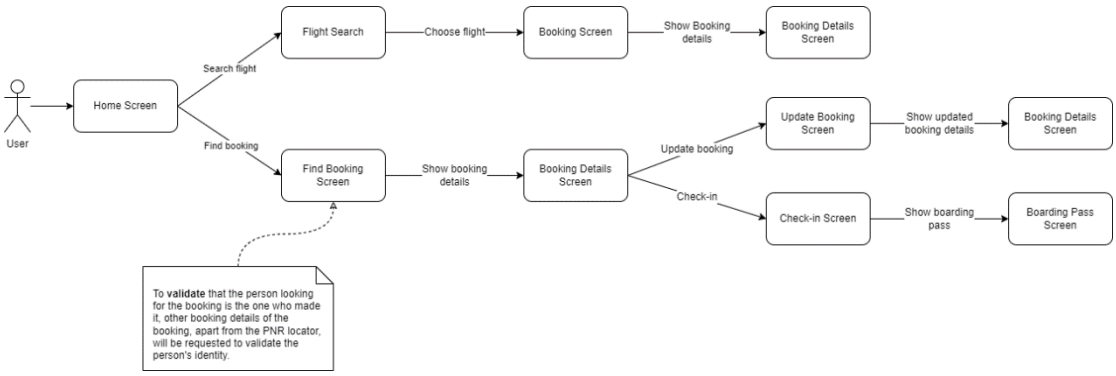
- **React** – ספרייה ליצירת אפליקציות web שפותחה על ידי פייסבוק. הספרייה תשמש ליצירת אתר חברת התעופה.
- **Axios** – ספריית JavaScript/Typescript מבוססת Promises המפשיטה יצירת בקשות HTTP. תשמש לתקשורת בין ממשק המשתמש ל-API של המערכת.

1.5. תרשימים

1.5.1. תרשים המערכת

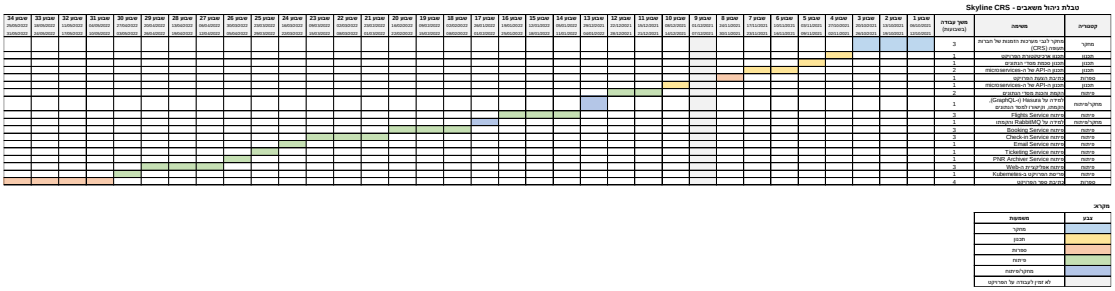


1.5.2. תרשים מסכים (User Flow)



1.6. לוחות זמנים

skyline-crs-project-gantt-chart.xlsx



טבלת ניהול משאבים - Skyline CRS

קטגוריה	משימה	מאמץ עבודה (בשבועות)	שבוע 1 (06/02/2023)	שבוע 2 (13/02/2023)	שבוע 3 (20/02/2023)	שבוע 4 (27/02/2023)	שבוע 5 (06/03/2023)	שבוע 6 (13/03/2023)	שבוע 7 (17/03/2023)	שבוע 8 (24/03/2023)	שבוע 9 (31/03/2023)	שבוע 10 (07/04/2023)	שבוע 11 (14/04/2023)	שבוע 12 (21/04/2023)	שבוע 13 (28/04/2023)	שבוע 14 (05/05/2023)	שבוע 15 (12/05/2023)	שבוע 16 (19/05/2023)	שבוע 17 (26/05/2023)	שבוע 18 (02/06/2023)	שבוע 19 (09/06/2023)	שבוע 20 (16/06/2023)	שבוע 21 (23/06/2023)	שבוע 22 (30/06/2023)	שבוע 23 (07/07/2023)	שבוע 24 (14/07/2023)	שבוע 25 (21/07/2023)	שבוע 26 (28/07/2023)	שבוע 27 (04/08/2023)	שבוע 28 (11/08/2023)	שבוע 29 (18/08/2023)	שבוע 30 (25/08/2023)	שבוע 31 (01/09/2023)	שבוע 32 (08/09/2023)	שבוע 33 (15/09/2023)	שבוע 34 (22/09/2023)
מתקדם	מתקדם לגבי הערכות הזמנות של הברות תעופה (CRS)	3																																		
תכנון	תכנון ארכיטקטורת המערכת	1																																		
תכנון	תכנון מסמכי מפרט ותיעוד	2																																		
תכנון	תכנון ה-APIים של ה-microservices	2																																		
הפרדה	הכנת הקמת הפרויקט	1																																		
תכנון	תכנון ה-APIים של ה-microservices	1																																		
פיתוח	הגדרת מבנה מסד הנתונים	2																																		
מתקדם/פיתוח	הגדרת של Hasura (GraphQL-i) מעבר, גישה למסד הנתונים	1																																		
פיתוח	פיתוח Flights Service	3																																		
מתקדם/פיתוח	הגדרת של RabbitMQ הנכונה	1																																		
פיתוח	פיתוח Booking Service	3																																		
פיתוח	פיתוח Check-in Service	3																																		
פיתוח	פיתוח Email Service	1																																		
פיתוח	פיתוח Ticketing Service	1																																		
פיתוח	פיתוח PNR Archive Service	1																																		
פיתוח	פיתוח אפליקציית Web-ה	3																																		
פיתוח	הגדרת תשתיות Kubernetes	2																																		
סגירה	הכנת סוף הפרויקט	4																																		

מקרא

צבע	משמעות
	מתקדם
	תכנון
	סגירה
	פיתוח
	מתקדם/פיתוח
	לא נדרש לעבודה על הפרויקט

1.7. אישורים

חתימת הסטודנט: עידו סבן

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 הכוון 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.

2. 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.

3. 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

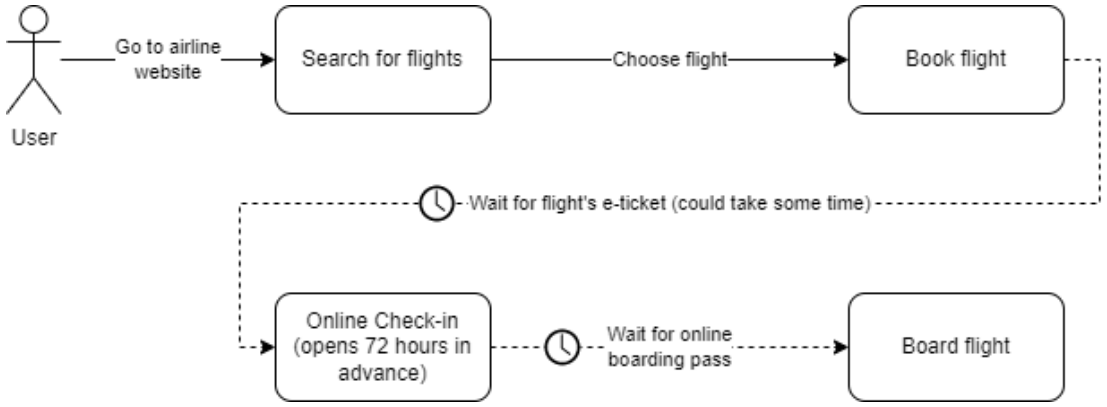
6.1. רקע כללי

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:



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, בעיקר Advanced 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-API REST (Representation State Transfer) 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):

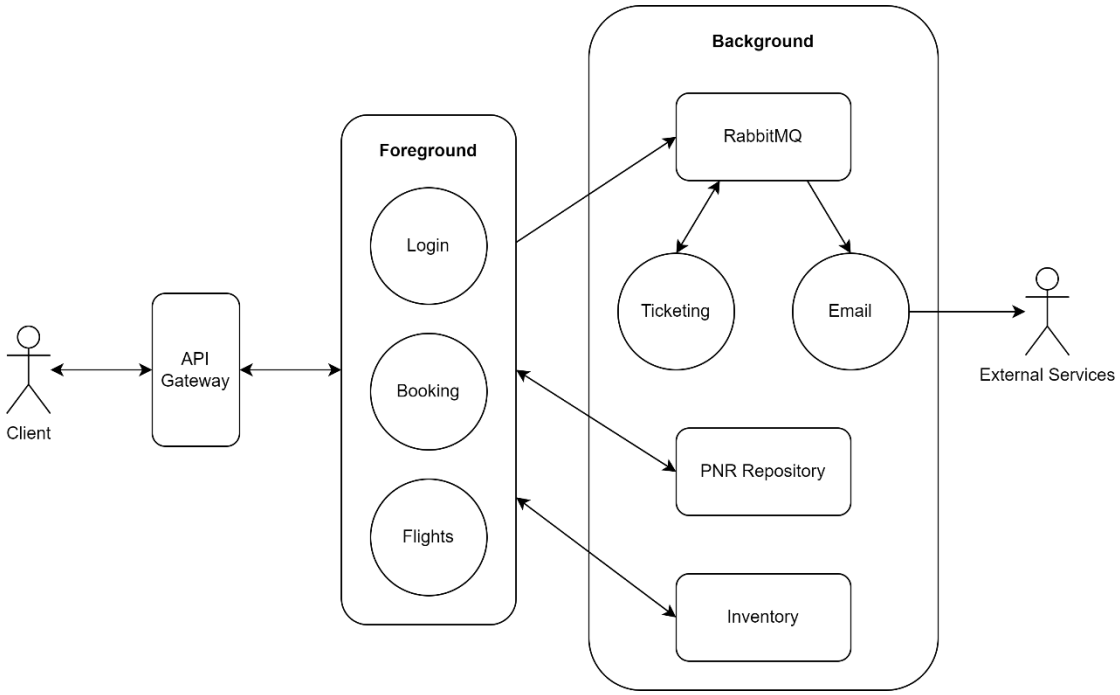
- שימוש בארכיטקטורת microservices למערכת.
- אריזת ה-microservices כתמונות והרצתם בקונטיינרים.
- שימוש ב-Kubernetes (OpenShift ספציפית) לאורכסטרציית קונטיינרים.
- נובע מהנקודות הקודמות – מערכת המאפשרת הרחבה לרוחב (horizontal scaling) ועמידות לתקלות (fault tolerance) המסוגלת לעמוד בעומסים.

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.

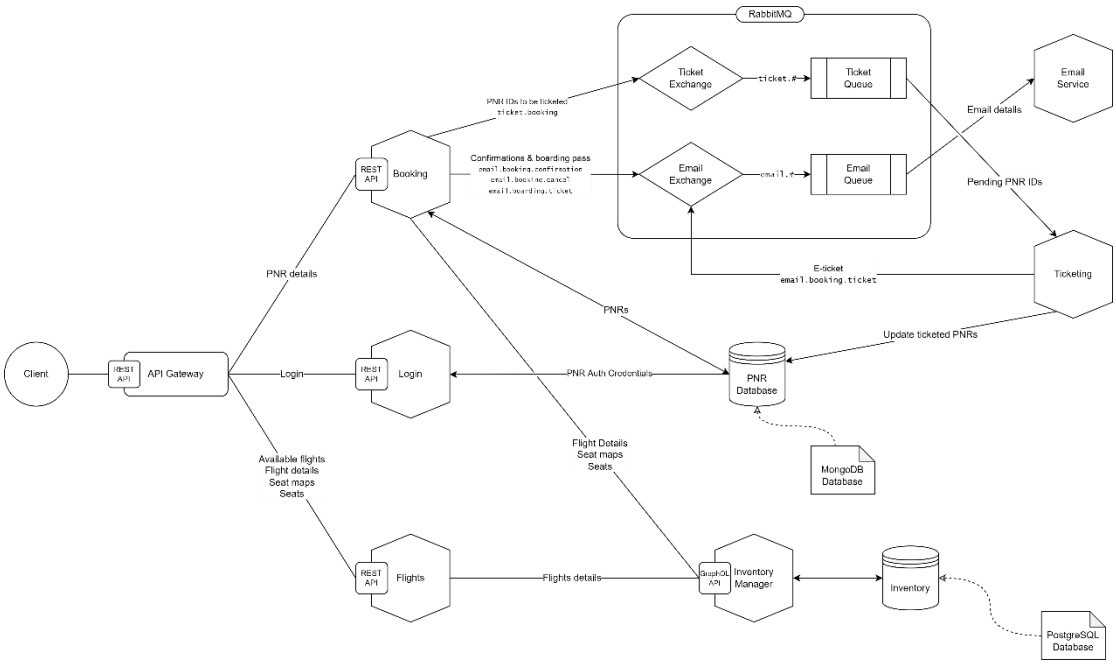
10.5. אילוצים

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 cluster Kubernetes/OpenShift קיים.

11. תיאור הארכיטקטורה

11.1. ארכיטקטורת המערכת



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.

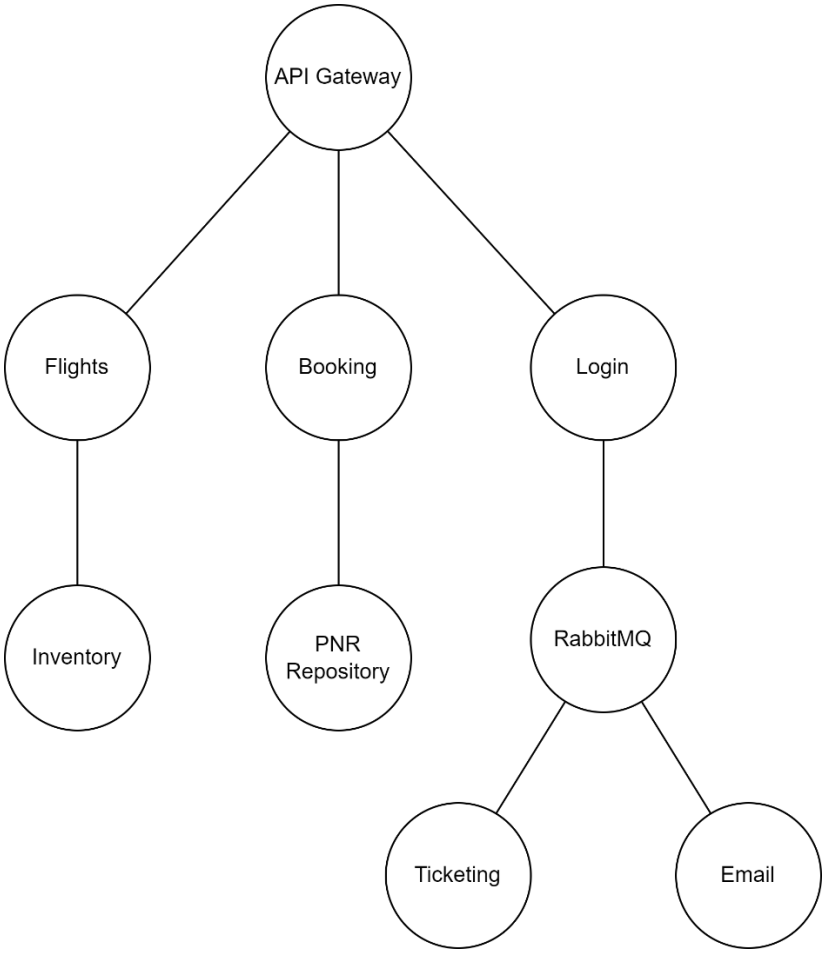
- **תור (Queue)** – מבנה נתונים מסוג FIFO (First In First Out), בו נתונים נשמרים בתור לוגי, כפי שאומר שמו. יש שימוש במבנה נתונים זה אצל RabbitMQ, והוא מושמש בפרויקט לניהול עבודות עתידיות.

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.

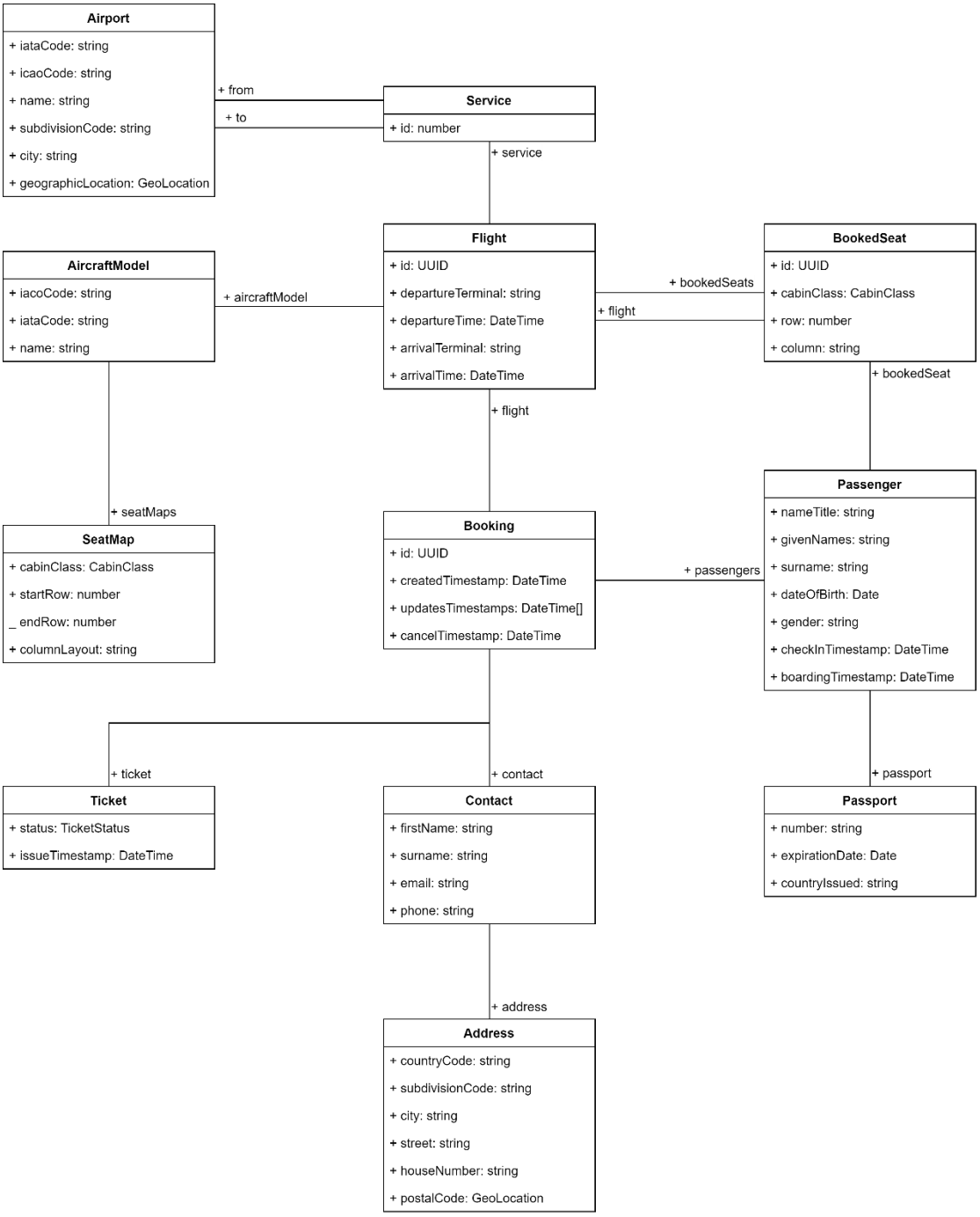
12.5. עץ מודולים



12.6. דיאגרמת UML Use Case

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.



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

תרשים ה"מחלקות" שבסעיף 12.8 הוא למעשה ה-domain model של המערכת, כלומר מודל של הנתונים בהם משתמשת המערכת.

להלן הסבר לגבי כל אחד מהישויות בתרשים:

- **Airport** – פרטים של שדה תעופה אמיתי. מכיל את קודי ה-IATA וה-ICAO (מזהים סטנדרטיים) ופרטים נוספים, כמו שם השדה והמיקום שלו.
- **Service** – שירות של חברת התעופה. שירות הוא למעשה מסלול קבוע בין שדה תעופה מקור לשדה תעופה יעד שמספקת חברת התעופה טיסות עבורו. לכל שירות קיים מספר מזהה, שלרוב יצורף למזהה של חברת התעופה. למשל, השירות LY1 של חברת התעופה אל על הוא בין תל אביב (TLV) לניו יורק (JFK).
- **Flight** – טיסה יחידה במערכת. לכל טיסה קיים מזהה ציבורי מסוג UUID, וכוללת פרטים כמו זמני ההמראה וההגעה, טרמינלי ההמראה וההגעה, מודל המטוס של הטיסה, ורשימה של הכיסאות שהוזמנו בטיסה.
- **AircraftModel** – מודל מטוס של חברת התעופה. מכיל פרטים כמו מזהים סטנדרטיים של המודל ואת סדר הישיבה במטוס (רשימה של SeatMap).
- **SeatMap** – מייצג אזור של מושבים במטוס, והאופן בו הם מסודרים. כל סדר ישיבה במטוס מורכב ממספר של SeatMaps. מכיל את המחלקה בו הוא נמצא (מחלקה ראשונה, עסקים וכו'), את השורה ממנה הוא מתחיל ואת השורה בה הוא נגמר, ואת מתווה המושבים. מתווה מושבים מיוצג כמחרוזת, כאשר כל אות במחרוזת מייצגת את שם העמודה של המושבים, התו "-" מייצג מעבר בין מושבים, והתו "#" מייצג שקיימת שם עמודה, אבל אין שם מושבים. למשל, ABC-###-HJK מייצג סידור ישיבה בו קיימות 9 עמודות, כאשר בין כל 3 עמודות יש מעבר, ובעמודות האמצעיות אין מושבים.
- **BookedSeat** – מושב תפוס בטיסה. לכל מושב יש מזהה UUID ציבורי, ומכיל פרטים לגבי המושב, כמו המחלקה בו הוא נמצא והמיקום שלו במטוס (שורה ועמודה). לכל מושב יש גם ייחוס לטיסה בה הוא נמצא.
- **Booking** – הזמנה (PNR) במערכת. לכל הזמנה מזהה UUID ציבורי ומספר שדות למעקב אחר זמנים בהם נעשו שינויים בהזמנה. הזמנות מכילות גם רשימת נוסעים של ההזמנה, פרטי התקשרות, וכרטיס הטיסה.

- **Passenger** – נוסע השייך להזמנה. מכיל את פרטי הנוסע, כמו שם, תאריך לידה, מגדר וכדומה. נוסף לכך, לכל נוסע יש גם שדה של דרכון, שפרטיו ימולאו בזמן תהליך הצ'ק-אין.
- **Passport** – דרכון של נוסע. מכיל פרטים הקשורים לדרכון, בהם מספר הדרכון, תאריך התפוגה והמדינה בה הונפק.
- **Ticket** – מכיל פרטים לגבי כרטיס הטיסה של נוסע, בהם סטטוס הכרטיס והזמן בו הוא תוקף.
- **Contact** – פרטי התקשרות להזמנה. מכיל פרטים כמו שם, פרטי התקשרות (אימייל, מספר טלפון), וכתובת.
- **Address** – כתובת לפרטי התקשרות. מכיל פרטים כמו מזהה המדינה, העיר, הרחוב, המיקוד וכדומה.

13. רכיבי ממשק

המערכת שפיתחתי חושפת ממשק REST API. אפרט בפרק זה על הממשק של המערכת. מאחר שכבר כתבתי דוקומנטציה באנגלית לממשק הציבורי, אדביק אותו כאן. כל הדוקומנטציה שכאן זמינה גם בכתובת: https://idos2002.github.io/skyline-crs.

13.1. ניהול הזמנות

13.1.1. הזמנת טיסה – Create Booking

Creates a new booking for the requested booking details. May fail if not enough seats are available or if the given PNR details are invalid.

Request

```
POST /booking
```

Body

```
{
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "seat": {
        "cabinClass": "<Cabin class of the seat to book: F / B / E>",
        "row": "<The row of the seat to book>",
        "column": "<The row of the seat to book>"
      }
    }
  ],
  "flightId": "<Flight ID>",
```

```
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "countryCode": "<Contact address ISO 3166-1 alpha-2 country code>",
      "subdivisionCode": "<Contact address ISO 3166-2 subdivision code
(optional)>",
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  }
}
```

Example:

```
{
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Dan",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "seat": {
        "cabinClass": "F",
        "row": 4,
        "column": "A"
      }
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
      "gender": "female",
      "seat": {
        "cabinClass": "F",
        "row": 4,
        "column": "D"
      }
    }
  ],
  "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
  "contact": {
    "firstName": "John",
    "surname": "Doe",
    "email": "john.doe@example.com",
    "phone": "+972541234567",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
```

```
    }
  }
}
```

Success Response - 201 Created

```
{
  "id": "<PNR ID of the newly created booking>",
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "bookedSeatId": "<Booked seat ID of the passenger's seat in the
flight in standard UUID format>"
    }
  ],
  "flightId": "<Flight ID>",
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "country": "<Contact address country>",
      "administrativeDivision": "<Contact address administrative division,
e.g. state, province, region, etc. (optional)>",
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  },
  "ticket": {
    "status": "pending"
  },
  "createdTimestamp": "<PNR creation timestamp>"
}
```

Example:

```
{
  "id": "f362846f-679d-4ef7-857d-e321c622cb41",
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Albert",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "bookedSeatId": "e3bfa7ae-a03b-11ec-a75d-0242ac120002"
    },
    {
      "nameTitle": "Mrs",
```

```
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
      "gender": "female",
      "bookedSeatId": "0509d3a3-5ce1-437d-b4b4-b971aa2c0657"
    }
  ],
  "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
  "contact": {
    "firstName": "John",
    "surname": "Doe",
    "email": "john.doe@example.com",
    "phone": "+972541234567",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
    }
  },
  "ticket": {
    "status": "pending"
  },
  "createdTimestamp": "2020-10-10T14:23:05.659711Z"
}
```

Flight Not Found Response - 404 Not Found

```
{
  "error": "Flight not found",
  "message": "Could not find flight with the requested flight ID."
}
```

Seats Not Available Response - 409 Conflict

```
{
  "error": "Seats not available",
  "message": "Could not book the requested seats, as they are already
booked."
}
```

Validation Error - 422 Unprocessable Entity

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "<Explanation about the error>"
    }
  ]
}
```

Example:

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "body/passengers/0/gender",
      "message": "gender must be a valid enum value"
    }
  ]
}
```

13.1.2. מציאת הזמנה – Find Booking

Gets the booking with the requested PNR ID.

Important: Login is required to access this endpoint.

Request

```
GET /booking/{pnrId}
```

Parameter	Description	Format
{pnrId}	The PNR ID (booking number) of the requested booking.	UUID string

Example:

```
GET /bookings/f362846f-679d-4ef7-857d-e321c622cb41
```

Success Response - 200 OK

```
{
  "id": "<PNR ID of the booking>",
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "bookedSeatId": "<Booked seat ID of the passenger's seat in the
flight in standard UUID format>",
      "passport": {
        "number": "<The passport number (optional)>",
        "expirationDate": "<The expiration date of the passport
(optional)>",
```

```
        "countryIssued": "<The ISO 3166-1 alpha-2 country code of the
country that issued this passport (optional)>"
      },
      "checkInTimestamp": "<Check-in timestamp for this passenger
(optional)>",
      "boardingTimestamp": "<Plane boarding timestamp for this passenger
(optional)>"
    }
  ],
  "flightId": "<Flight ID>",
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "countryCode": "<Contact address ISO 3166-1 alpha-2 country code>",
      "subdivisionCode": "<Contact address ISO 3166-2 subdivision code
(optional)>",
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  },
  "ticket": {
    "status": "<Ticketing status: pending / issued / canceled>",
    "issueTimestamp": "<Ticket issue timestamp (optional)>"
  },
  "createdTimestamp": "<PNR creation timestamp>",
  "updatesTimestamps": ["<PNR updates timestamps (optional)>"],
  "cancelTimestamp": "<PNR cancellation timestamp (optional)>"
}
```

Example:

```
{
  "id": "f362846f-679d-4ef7-857d-e321c622cb41",
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Albert",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "bookedSeatId": "2dc2ad2b-23ea-429a-bcf9-a462d0e42806",
      "passport": {
        "number": "12345678",
        "expirationDate": "2022-01-01T00:00:00.000Z",
        "countryIssued": "IL"
      },
      "checkInTimestamp": "2020-10-20T02:15:54.659720Z"
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
```

```
      "gender": "female",
      "passport": {
        "number": "87654321",
        "expirationDate": "2024-06-01T00:00:00.000Z",
        "countryIssued": "IL"
      },
      "bookedSeatId": "2edb1071-f3ab-4754-b40f-38616e2b8060",
      "checkInTimestamp": "2020-10-20T02:15:54.659720Z"
    }
  ],
  "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
  "contact": {
    "firstName": "John",
    "surname": "Doe",
    "email": "john.doe@example.com",
    "phone": "+972541234567",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
    }
  },
  "ticket": {
    "status": "issued",
    "issueTimestamp": "2020-10-11T22:58:43.236672Z"
  },
  "createdTimestamp": "2020-10-10T14:23:05.659711Z",
  "updatesTimestamps": ["2020-10-17T07:31:01.678945Z"]
}
```


Unauthorized Access Response - 401 Unauthorized

```
{
  "error": "Unauthorized access",
  "message": "The Authorization header is missing or invalid."
}
```

Booking Not Found Response - 404 Not Found

```
{
  "error": "Booking not found",
  "message": "Could not find a booking with the given PNR ID."
}
```

Validation Error - 422 Unprocessable Entity

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "<Explanation about the error>"
    }
```

```
  ]
}
```

Example:

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "path/id",
      "message": "id must be a UUID"
    }
  ]
}
```

13.1.3. עדכון הזמנה – Update Booking

Update the booking with the requested PNR ID. Optional fields omitted previously may be added to the booking through this operation. May fail if the given PNR details are invalid or if all or some of the passengers have already checked in for the flight.

Important: Login is required to access this endpoint.

Note: An updated ticket will be issued for this update.

Restrictions and Warnings

- Passenger's list must have the same length as in the existing PNR - the client may not add passengers for an existing booking!
- The passengers will be updated (replaced) according to their order in the request.

Request

`PUT /booking/{pnrId}`

Parameter	Description	Format
`{pnrId}`	The PNR ID (booking number) of the booking to update.	UUID string

Example:

`PUT /bookings/f362846f-679d-4ef7-857d-e321c622cb41`

Body

```
{
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>"
    }
  ],
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "countryCode": "<Contact address ISO 3166-1 alpha-2 country code>",
      "subdivisionCode": "<Contact address ISO 3166-2 subdivision code
(optional)>",
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  }
}
```

Example:

```
{
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "Josh Daniel",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male"
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-06-01T00:00:00.000Z",
      "gender": "female"
    }
  ],
  "contact": {
    "firstName": "Josh",
    "surname": "Doe",
    "email": "john.doe.updated@example.com",
    "phone": "+972547654321",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
```

```
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
    }
  }
}
```

Success Response - 200 OK

```
{
  "id": "<PNR ID of the updated booking>",
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "bookedSeatId": "<Booked seat ID of the passenger's seat in the
flight in standard UUID format>"
    }
  ],
  "flightId": "<Flight ID>",
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "countryCode": "<Contact address ISO 3166-1 alpha-2 country code>",
      "subdivisionCode": "<Contact address ISO 3166-2 subdivision code
(optional)>",
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  },
  "ticket": {
    "status": "pending",
    "issueTimestamp": "<Ticket issue timestamp (optional)>"
  },
  "createdTimestamp": "<PNR creation timestamp>",
  "updatesTimestamps": ["<PNR updates timestamps (optional)>"]
}
```

Example:

```
{
  "id": "f362846f-679d-4ef7-857d-e321c622cb41",
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Albert",
```

```
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "bookedSeatId": "e3bfa7ae-a03b-11ec-a75d-0242ac120002"
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
      "gender": "female",
      "bookedSeatId": "0509d3a3-5ce1-437d-b4b4-b971aa2c0657"
    }
  ],
  "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
  "contact": {
    "firstName": "John",
    "surname": "Doe",
    "email": "john.doe@example.com",
    "phone": "+972541234567",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
    }
  },
  "ticket": {
    "status": "pending"
  },
  "createdTimestamp": "2020-10-10T14:23:05.659711Z",
  "updatesTimestamps": ["2020-10-16T20:41:07.364729Z"]
}
```


Unauthorized Access Response - `401 Unauthorized`


```
{
  "error": "Unauthorized access",
  "message": "The Authorization header is missing or invalid."
}
```


Booking Not Found Response - `404 Not Found`


```
{
  "error": "Booking not found",
  "message": "Could not find a booking with the given PNR ID."
}
```


Passenger Count Change - `409 Conflict`

In the case the number of passengers in the request body does not match the number of passengers in the booking.

```
{
  "error": "Passenger count change",
  "message": "Passenger additions or removals are not allowed."
}
```

Already Checked In - 409 Conflict

```
{
  "error": "Already checked in",
  "message": "Could not update or cancel a booking which all or some of its
passengers have already checked in."
}
```

Booking Already Canceled - 409 Conflict

```
{
  "error": "Booking already canceled",
  "message": "Could not update or cancel a booking which is already
canceled."
}
```

Validation Error - 422 Unprocessable Entity

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "<Explanation about the error>"
    }
  ]
}
```

Example:

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "path/id",
      "message": "id must be a UUID"
    }
  ]
}
```

13.1.4. ביטול הזמנה – Cancel Booking

Cancels the booking with the requested PNR ID. A booking **cannot be canceled** if all or some of the passengers have already checked in for the flight!

Warning! This operation is irreversible and can only be done once!
Important: Login is required to access this endpoint.
Note: Canceled PNRs will be archived and will not be accessible through this API thereafter.

Request

`POST /booking/{pnrId}/cancel`

Parameter	Description	Format
`{pnrId}`	The PNR ID (booking number) of the booking to cancel.	UUID string

Example:

`POST /bookings/f362846f-679d-4ef7-857d-e321c622cb41/cancel`

Success Response - 200 OK

```
{
  "id": "<PNR ID of the booking>",
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "bookedSeatId": "<Booked seat ID of the passenger's seat in the
flight in standard UUID format>"
    }
  ],
  "flightId": "<Flight ID>",
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "countryCode": "<Contact address ISO 3166-1 alpha-2 country code>",
      "subdivisionCode": "<Contact address ISO 3166-2 subdivision code
(optional)>",
```

```
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  },
  "ticket": {
    "status": "canceled",
    "issueTimestamp": "<Ticket issue timestamp (optional)>"
  },
  "createdTimestamp": "<PNR creation timestamp>",
  "updatesTimestamps": ["<PNR updates timestamps (optional)>"],
  "cancelTimestamp": "<PNR cancellation timestamp (optional)>"
}
```

Example:

```
{
  "id": "f362846f-679d-4ef7-857d-e321c622cb41",
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Albert",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "bookedSeatId": "e3bfa7ae-a03b-11ec-a75d-0242ac120002"
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
      "gender": "female",
      "bookedSeatId": "0509d3a3-5ce1-437d-b4b4-b971aa2c0657"
    }
  ],
  "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
  "contact": {
    "firstName": "John",
    "surname": "Doe",
    "email": "john.doe@example.com",
    "phone": "+972541234567",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
    }
  },
  "ticket": {
    "status": "canceled",
    "issueTimestamp": "2020-10-11T22:58:43.236672Z"
  },
  "createdTimestamp": "2020-10-10T14:23:05.659711Z",
  "updatesTimestamps": ["2020-10-17T07:31:01.678945Z"],
  "cancelTimestamp": "2020-10-20T02:15:54.659720Z"
```

```
}
```

Unauthorized Access Response - 401 Unauthorized

```
{
  "error": "Unauthorized access",
  "message": "The Authorization header is missing or invalid."
}
```

Booking Not Found Response - 404 Not Found

```
{
  "error": "Booking not found",
  "message": "Could not find a booking with the given PNR ID."
}
```

Booking Already Canceled Response - 409 Conflict

```
{
  "error": "Booking already canceled",
  "message": "Could not update or cancel a booking which is already
canceled."
}
```

Already Checked In Response - 409 Conflict

```
{
  "error": "Already checked in",
  "message": "Could not update or cancel a booking which all or some of its
passengers have already checked in."
}
```

Validation Error - 422 Unprocessable Entity

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "<Explanation about the error>"
    }
  ]
}
```

Example:

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "path/id",
```

```
      "message": "id must be a UUID"
    }
  ]
}
```

13.1.5. צ'ק-אין – Check In

Check in all or some passengers of a booking (PNR) to their flight. The check-in process may fail if the given details do not match those in the existing PNR (e.g. name, date of birth, etc.).

When checking in a passenger, a boarding pass will be issued for the passenger, and its information will, be accessible through the find booking endpoint.

Warning! This operation is irreversible and can only be done once!

Important: Login is required to access this endpoint.

Request

`POST /booking/{pnrId}/checkIn`

Parameter	Description	Format
`{pnrId}`	The PNR ID (booking number) of the booking to check in.	UUID string

Example:

`POST /booking/f362846f-679d-4ef7-857d-e321c622cb41/checkIn`

Body

```
{
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "bookedSeatId": "<Booked seat ID of the passenger's seat in the
flight in standard UUID format>",
      "passport": {
        "number": "<The passport number>",
        "expirationDate": "<The expiration date of the passport>",
```

```
        "countryIssued": "<The ISO 3166-1 alpha-2 country code of the
country that issued this passport>"
      }
    }
  ]
}
```

Example:

```
{
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Albert",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "bookedSeatId": "2dc2ad2b-23ea-429a-bcf9-a462d0e42806",
      "passport": {
        "number": "12345678",
        "expirationDate": "2022-01-01T00:00:00.000Z",
        "countryIssued": "IL"
      }
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
      "gender": "female",
      "bookedSeatId": "2edb1071-f3ab-4754-b40f-38616e2b8060",
      "passport": {
        "number": "87654321",
        "expirationDate": "2024-06-01T00:00:00.000Z",
        "countryIssued": "IL"
      }
    }
  ]
}
```

Success Response - `200 OK`

```
{
  "id": "<PNR ID of the booking>",
  "passengers": [
    {
      "nameTitle": "<Passenger name title, e.g. Mr, Mrs, etc. (optional)>",
      "givenNames": "<Passenger given names (first name and middle
names)>",
      "surname": "<Passenger surname (last name)>",
      "dateOfBirth": "<Passenger date of birth (in UTC ISO 8601 format)>",
      "gender": "<Passenger gender: male / female / other / unspecified>",
      "bookedSeatId": "<Booked seat ID of the passenger's seat in the
flight in standard UUID format>",
      "passport": {
```

```
        "number": "<The passport number>",
        "expirationDate": "<The expiration date of the passport>",
        "countryIssued": "<The ISO 3166-1 alpha-2 country code of the
country that issued this passport>"
      },
      "checkInTimestamp": "<Check-in timestamp for this passenger>"
    }
  ],
  "flightId": "<Flight ID>",
  "contact": {
    "firstName": "<First name of the person who made the booking>",
    "surname": "<Surname of the person who made the booking>",
    "email": "<Contact email address>",
    "phone": "<Contact international phone number>",
    "address": {
      "countryCode": "<Contact address ISO 3166-1 alpha-2 country code>",
      "subdivisionCode": "<Contact address ISO 3166-2 subdivision code
(optional)>",
      "city": "<Contact address city>",
      "street": "<Contact address street name>",
      "houseNumber": "<Contact address house number>",
      "postalCode": "<Contact address postal code>"
    }
  },
  "ticket": {
    "status": "<Ticketing status: issued>",
    "issueTimestamp": "<Ticket issue timestamp>"
  },
  "createdTimestamp": "<PNR creation timestamp>",
  "updatesTimestamps": ["<PNR updates timestamps (optional)>"]
}
```

Example:

```
{
  "id": "f362846f-679d-4ef7-857d-e321c622cb41",
  "passengers": [
    {
      "nameTitle": "Mr",
      "givenNames": "John Albert",
      "surname": "Doe",
      "dateOfBirth": "2000-01-01T00:00:00.000Z",
      "gender": "male",
      "bookedSeatId": "2dc2ad2b-23ea-429a-bcf9-a462d0e42806",
      "passport": {
        "number": "12345678",
        "expirationDate": "2022-01-01T00:00:00.000Z",
        "countryIssued": "IL"
      },
      "checkInTimestamp": "2020-10-20T02:15:54.659720Z"
    },
    {
      "nameTitle": "Mrs",
      "givenNames": "Jane",
      "surname": "Doe",
      "dateOfBirth": "2002-01-01T00:00:00.000Z",
      "gender": "female",
```

```
      "bookedSeatId": "2edb1071-f3ab-4754-b40f-38616e2b8060",
      "passport": {
        "number": "87654321",
        "expirationDate": "2024-06-01T00:00:00.000Z",
        "countryIssued": "IL"
      },
      "checkInTimestamp": "2020-10-20T02:15:54.659720Z"
    }
  ],
  "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
  "contact": {
    "firstName": "John",
    "surname": "Doe",
    "email": "john.doe@example.com",
    "phone": "+972541234567",
    "address": {
      "countryCode": "IL",
      "city": "Tel Aviv-Yafo",
      "street": "Shlomo Rd.",
      "houseNumber": "136",
      "postalCode": "6603248"
    }
  },
  "ticket": {
    "status": "issued",
    "issueTimestamp": "2020-10-11T22:58:43.236672Z"
  },
  "createdTimestamp": "2020-10-10T14:23:05.659711Z",
  "updatesTimestamps": ["2020-10-17T07:31:01.678945Z"]
}
```


Unauthorized Access Response - `401 Unauthorized`


```
{
  "error": "Unauthorized access",
  "message": "The Authorization header is missing or invalid."
}
```


Booking Not Found Response - `404 Not Found`


```
{
  "error": "Booking not found",
  "message": "Could not find a booking with the given PNR ID."
}
```


Booking Not Ticketed Response - `409 Conflict`


```
{
  "error": "Booking not ticketed",
  "message": "Could not check in for a booking that has not been ticketed."
}
```

Check-in Validation Error Response - `409 Conflict`

```
{
  "error": "Check-in validation error",
  "message": "Check-in passenger details do not match those of the
booking.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "Passenger details do not match the booking"
    }
  ]
}
```

Example:

```
{
  "error": "Check-in validation error",
  "message": "Check-in passenger details do not match those of the
booking.",
  "details": [
    {
      "cause": "body/passengers/0",
      "message": "Passenger details do not match the booking"
    }
  ]
}
```

Passenger Already Checked-in Response - `409 Conflict`

```
{
  "error": "Passenger already checked-in",
  "message": "One of the passengers has already checked in.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "Passenger is already checked-in"
    }
  ]
}
```

Example:

```
{
  "error": "Passenger already checked-in",
  "message": "One of the passengers has already checked in.",
  "details": [
    {
      "cause": "body/passengers/0",
      "message": "Passenger is already checked-in"
    }
  ]
}
```

Validation Error - 422 Unprocessable Entity

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "<The part of the request that caused the error>",
      "message": "<Explanation about the error>"
    }
  ]
}
```

Example:

```
{
  "error": "Validation error",
  "message": "Request has an invalid format.",
  "details": [
    {
      "cause": "path/id",
      "message": "id must be a UUID"
    }
  ]
}
```

13.2. מידע על טיסות

13.2.1. חיפוש טיסות – Find Flights

Retrieve a list of flights from the requested origin to the requested destination at the provided date.

Request

```
GET /flights/{origin}/{destination}/{departureDate}
```

Parameter	Description	Format
`{origin}`	The IATA airport code of the airport to depart from.	3-letter IATA airport code, e.g. TLV
`{destination}`	The IATA airport code of the destination airport.	3-letter IATA airport code

`{departureTime}`	The departure time to find flights for. [1]	ISO 1806 datetime
`passengers`	The number of passengers to find a flight for. **Default:** 1	Positive integer, e.g. 2
`cabin`	The cabin class of the flight. **Default:** all cabin classes	One of the following cabin class codes: E, B, F

Notes:

1. All flights from the given departure time to 24 hours after will be included. For example: Searching for a flight with a departure time of `2021-01-01T06:00:00Z` will result in a list of all flights from `2021-01-01T06:00:00Z` to `2021-01-02T06:00:00Z`.

Examples:

```
GET /flights/TLV/BER/2021-01-01
GET /flights/AMS/FRA/2021-12-07?passengers=4
GET /flights/TLV/JFK/2021-11-12?class=E
GET /flights/LAX/TLV/2021-03-22?passengers=2&cabin=F
```

Cabin Class Codes

There are three available cabin classes, each associated with a single letter code:

- **E** - Economy class
- **B** - Business class
- **F** - First class

Success Response - `200 OK`

```
{
  "name": "<Service name for this itinerary>",
  "origin": {
    "iataCode": "<IATA airport code of the origin>",
    "icaoCode": "<ICAO airport code of the origin>",
    "name": "<Origin airport name>",
    "location": {
```

```
      "subdivisionCode": "<The ISO 3166-2 subdivision code the airport is
located in>",
      "city": "<The city the airport is located in>",
      "coordinates": {
        "crs": "<The coordinates reference system (CRS) URN used for the
coordinates data, e.g. urn:ogc:def:crs:EPSG::4326>",
        "data": ["<The coordinates data>"]
      }
    }
  },
  "destination": {
    "iataCode": "<IATA airport code of the destination>",
    "icaoCode": "<ICAO airport code of the destination>",
    "name": "<Destination airport name>",
    "location": {
      "subdivisionCode": "<The ISO 3166-2 subdivision code the airport is
located in>",
      "city": "<The city the airport is located in>",
      "coordinates": {
        "crs": "<The coordinates reference system (CRS) URN used for the
coordinates data, e.g. urn:ogc:def:crs:EPSG::4326>",
        "data": ["<The coordinates data>"]
      }
    }
  },
  "flights": [
    {
      "id": "<ID of the flight>",
      "departureTerminal": "<Departure terminal name>",
      "departureTime": "<Departure time in ISO 1806 format>",
      "arrivalTerminal": "<Arrival terminal name>",
      "arrivalTime": "<Arrival time in ISO 1806 format>",
      "aircraftModel": {
        "icaoCode": "<ICAO aircraft type designator code>",
        "iataCode": "<IATA aircraft type designator code>",
        "name": "<Model name of the aircraft>"
      },
      "cabins": [
        {
          "cabinClass": "<Cabin class: E / B / F>",
          "seatsCount": "<Total number of seats>",
          "availableSeatsCount": "<Number of available seats>"
        }
      ]
    }
  ]
}
```

Example:

```
{
  "name": "SKL1",
  "origin": {
    "iataCode": "TLV",
    "icaoCode": "LLBG",
    "name": "Ben Gurion Airport",
    "location": {
      "subdivisionCode": "IL-M",
```

```
      "city": "Tel Aviv-Yafo",
      "coordinates": {
        "crs": "urn:ogc:def:crs:EPSG::4326",
        "data": [32.009444, 34.882778]
      }
    }
  },
  "destination": {
    "iataCode": "JFK",
    "icaoCode": "KJFK",
    "name": "John F. Kennedy International Airport",
    "location": {
      "subdivisionCode": "US-NY",
      "city": "New York City",
      "coordinates": {
        "crs": "urn:ogc:def:crs:EPSG::4326",
        "data": [40.639722, -73.778889]
      }
    }
  },
  "flights": [
    {
      "id": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
      "departureTerminal": "3",
      "departureTime": "2021-10-11T22:45:00Z",
      "arrivalTerminal": "4",
      "arrivalTime": "2021-10-12T10:45:00Z",
      "aircraftModel": {
        "icaoCode": "B789",
        "iataCode": "789",
        "name": "Boeing 787-9 Dreamliner"
      },
      "cabins": [
        {
          "cabinClass": "E",
          "seatsCount": 204,
          "availableSeatsCount": 201
        },
        {
          "cabinClass": "B",
          "seatsCount": 35,
          "availableSeatsCount": 14
        },
        {
          "cabinClass": "F",
          "seatsCount": 32,
          "availableSeatsCount": 21
        }
      ]
    }
  ]
}
```

Flights Not Found Response - `404 Not Found`

```
{
  "error": "Flights not found",
  "message": "The flights for the requested origin and destination
airports."
}
```

13.2.2. קבלת פרטי טיסה – Get Flight Details

Gets the flight with the requested flight ID.

Request

```
GET /flight/{flightId}
```

Parameter	Description	Format
`{flightId}`	The flight ID of the requested flight.	UUID string

Examples:

```
GET /flight/17564e2f-7d32-4d4a-9d99-27ccd768fb7d
```

Cabin Class Codes
There are three available cabin classes, each associated with a single letter code:

- **E** - Economy class
- **B** - Business class
- **F** - First class

Success Response - `200 OK`

```
{
    "id": "<ID of the flight>",
    "name": "<Service name for this itinerary>",
    "origin": {
        "iataCode": "<IATA airport code of the origin>",
        "icaoCode": "<ICAO airport code of the origin>",
        "name": "<Origin airport name>",
        "location": {
            "subdivisionCode": "<The ISO 3166-2 subdivision code the
airport is located in>",
            "city": "<The city the airport is located in>",
            "coordinates": {
```

```
                "crs": "<The coordinates reference system (CRS) URN used
for the coordinates data, e.g. urn:ogc:def:crs:EPSG::4326>",
                "data": [ "<The coordinates data>" ]
            }
        }
    },
    "destination": {
        "iataCode": "<IATA airport code of the destination>",
        "icaoCode": "<ICAO airport code of the destination>",
        "name": "<Destination airport name>",
        "location": {
            "subdivisionCode": "<The ISO 3166-2 subdivision code the
airport is located in>",
            "city": "<The city the airport is located in>",
            "coordinates": {
                "crs": "<The coordinates reference system (CRS) URN used
for the coordinates data, e.g. urn:ogc:def:crs:EPSG::4326>",
                "data": [ "<The coordinates data>" ]
            }
        }
    },
    "departureTerminal": "<Departure terminal name>",
    "departureTime": "<Departure time in ISO 1806 format>",
    "arrivalTerminal": "<Arrival terminal name>",
    "arrivalTime": "<Arrival time in ISO 1806 format>",
    "aircraftModel": {
        "icaoCode": "<ICAO aircraft type designator code>",
        "iataCode": "<IATA aircraft type designator code>",
        "name": "<Model name of the aircraft>"
    },
    "cabins": [
        {
            "cabinClass": "<Cabin class: E / B / F>",
            "seatsCount": "<Total number of seats>",
            "availableSeatsCount": "<Number of available seats>"
        },
    ]
}
```

Example:

```
{
    "id": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
    "name": "SKL1",
    "origin": {
        "iataCode": "TLV",
        "icaoCode": "LLBG",
        "name": "Ben Gurion Airport",
        "location": {
            "subdivisionCode": "IL-M",
            "city": "Tel Aviv-Yafo",
            "coordinates": {
                "crs": "urn:ogc:def:crs:EPSG::4326",
                "data": [ 32.009444, 34.882778 ]
            }
        }
    },
```

```
    "destination": {
        "iataCode": "JFK",
        "icaoCode": "KJFK",
        "name": "John F. Kennedy International Airport",
        "location": {
            "subdivisionCode": "US-NY",
            "city": "New York City",
            "coordinates": {
                "crs": "urn:ogc:def:crs:EPSG::4326",
                "data": [ 40.639722, -73.778889 ]
            }
        }
    },
    "departureTerminal": "3",
    "departureTime": "2021-10-11T22:45:00Z",
    "arrivalTerminal": "4",
    "arrivalTime": "2021-10-12T10:45:00Z",
    "aircraftModel": {
        "icaoCode": "B789",
        "iataCode": "789",
        "name": "Boeing 787-9 Dreamliner"
    },
    "cabins": [
        {
            "cabinClass": "E",
            "seatsCount": 204,
            "availableSeatsCount": 201
        },
        {
            "cabinClass": "B",
            "seatsCount": 35,
            "availableSeatsCount": 14
        },
        {
            "cabinClass": "F",
            "seatsCount": 32,
            "availableSeatsCount": 21
        }
    ]
}
```


Flight Not Found Response - 404 Not Found


```
{
    "error": "Flight not found",
    "message": "Could not find flight with the given flight ID."
}
```

13.2.3. קבלת סדר הישיבה – Get Flight Seats

Gets details about the seats in the requested flight. The result contains the seat map for this flight as well as a list of the booked seats on this flight.

Request

```
GET /flight/{flightId}/seats
```

Parameter	Description	Format
`{flightId}`	The flight ID of the requested flight.	UUID string

Examples:

```
GET /flight/17564e2f-7d32-4d4a-9d99-27ccd768fb7d/seats
```

Cabin Class Codes

There are three available cabin classes, each associated with a single letter code:

- **E** - Economy class
- **B** - Business class
- **F** - First class

Seat Map Column Layouts

The column layout of a seat map represents the way columns are spread in a seat map section using a string of characters.

- Characters in the range `A-Z` represent a column.
- The character `-` represents an aisle between the seats.
- The character `#` represents an empty column, to denote there is are no seats in that column.

For example, the column layout `ABC-DE-F#H` represents a layout where there are three columns to the left (`A`, `B`, `C`), an aisle, then two columns in the center (`D`, `E`), an aisle, and three columns to the right (`F`, `#`, `H`), with an empty column between columns `F` and `H`.

Success Response - `200 OK`

```
{
    "flightId": "<ID of the flight>",
    "aircraftModel": {
```

```
        "icaoCode": "<ICAO aircraft type designator code>",
        "iataCode": "<IATA aircraft type designator code>",
        "name": "<Model name of the aircraft>"
    },
    "seatMap": [
        {
            "cabinClass": "<Cabin class: E / B / F>",
            "startRow": "<Start row of the section>",
            "endRow": "<End row of the section>",
            "columnLayout": "<Column layout for this section, e.g. ABC-DE-
F#H>"
        },
    ],
    "bookedSeats": [
        {
            "row": "<Row number of the booked seat>",
            "column": "<Column name of the booked seat>"
        },
    ]
}
```

Example:

```
{
    "flightId": "17564e2f-7d32-4d4a-9d99-27ccd768fb7d",
    "aircraftModel": {
        "icaoCode": "B789",
        "iataCode": "789",
        "name": "Boeing 787-9 Dreamliner"
    },
    "seatMap": [
        {
            "cabinClass": "F",
            "startRow": 1,
            "endRow": 8,
            "columnLayout": "A-DG-K"
        },
        {
            "cabinClass": "B",
            "startRow": 10,
            "endRow": 14,
            "columnLayout": "AC-DFG-HK"
        },
        {
            "cabinClass": "E",
            "startRow": 21,
            "endRow": 28,
            "columnLayout": "ABC-DFG-HJK"
        },
        {
            "cabinClass": "E",
            "startRow": 29,
            "endRow": 30,
            "columnLayout": "###-###-HJK"
        },
        {
            "cabinClass": "E",
            "startRow": 35,
```

```
            "endRow": 36,
            "columnLayout": "ABC-###-HJK"
        },
        {
            "cabinClass": "E",
            "startRow": 37,
            "endRow": 48,
            "columnLayout": "ABC-DFG-HJK"
        },
        {
            "cabinClass": "E",
            "startRow": 49,
            "endRow": 50,
            "columnLayout": "###-DFG-###"
        },
    ],
    "bookedSeats": [
        {
            "row": 5,
            "column": "A"
        },
        {
            "row": 12,
            "column": "G"
        },
        {
            "row": 42,
            "column": "J"
        }
    ]
}
```


Flight Not Found Response - `404 Not Found`


```
{
    "error": "Flight not found",
    "message": "Could not find flight with the given flight ID."
}
```


13.3. התחברות

13.3.1. התחברות – Log In

Log in to Skyline CRS using the booking credentials to access protected endpoints.

Authentication Process

1. The client sends an authentication request to the `/login` endpoint with the required credentials: PNR ID, first name and last name of the person who made the booking.

2. The login service will verify the credentials with the existing data.

3. If the credentials are correct, a JWT access token will be generated and sent in the response body, and will have a **30 minutes** expiration time. The JWT token will be signed using the HS256 (HMAC-SHA256) algorithm, and will have the following payload format:

```
{
  "sub": "<The PNR ID of this access token>",
  "iat": "<The unix time this token has been issued>",
  "exp": "<The unix time for this token to expire (30 minutes
since issued)>",
}
```

Example payload (decoded):

```
{
  "sub": "f362846f-679d-4ef7-857d-e321c622cb41",
  "iat": "1639176300",
  "exp": "1639178100"
}
```

Using the secret `secret`, the following JWT token would be generated:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmMzYyODQ2Zi02NzlkLTRlZjctODU3ZC1lMzIxYzYyMmNiNDEiLCJpYXQiOiIxNjM5MTc2MzAwIiwiZXhwIjoiMTYzOTE3ODEwMCJ9.l1smQvKIIWZG6dLLopUrXsWs7cff8_SJQ0JYwB_sd9g

4. Then, for each request involving the authenticated booking, the access token will be supplied using the `Authorization` header for authorization. Using an invalid or expired token would result in an error response of course, and would require the client to re-authenticate (starting from step 1 as seen above).

Request

```
POST /login
```

Body

```
{
    "pnrId": "<The PNR ID (booking number) of the requested booking>",
    "firstName": "<First name of the person who made the booking>",
    "surname": "<surname of the person who made the booking>"
}
```

Example:

```
{
    "pnrId": "f362846f-679d-4ef7-857d-e321c622cb41",
    "firstName": "John",
    "surname": "Doe"
}
```

Success Response - 200 OK

```
{
    "token": "<The JWT access token>"
}
```

Example:


```
{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmMzYyODQ2Zi02NzlkLTRlZjctO
DU3ZC1lMzIxYzYyMmNiNDEiLCJpYXQiOiIxNjM5MTc2MzAwIiwiZXhwIjoiMTYzOTE3ODEwMCJ9
.l1smQvKIIWZG6dLLopUrXsWs7cff8_SJQ0JYwB_sd9g"
}
```


Log In Failed Response - 400 Bad Request

```
{
    "error": "Log in failed",
    "message": "Could not authenticate booking for the given PNR ID with
the given first name and surname."
}
```

Validation Error Response - 422 Unprocessable Entity

```
{
    "error": "Validation error",
    "message": "Request has an invalid format.",
    "details": [
        {
            "cause": "<Path to the missing credential>",
            "message": "<Specific error message>"
        },
    ]
}
```

Example:

```
{
    "error": "Validation error",
    "message": "Request has an invalid format.",
    "details": [
        {
            "cause": "body/surname",
            "message": "field is required"
```

```
        },
    ]
}
```

14. תיכון המערכת

14.1. ארכיטקטורת המערכת

תרשימים והסברים על ארכיטקטורת המערכת ניתן לראות בפרקים 10 ו-11.

14.2. תיכון מפורט

הסברים לגבי תפקידי המודלים במערכת וכיצד הם מתקשרים ניתן למצוא בפרקים 10, 11 ו-12. פירוט לגבי תפקידי ומטרות המערכת ניתן למצוא בפרק 3 ובסעיף 10.1.

14.3. חלופות לתיכון המערכת

במהלך תכנון ופתוח הפרויקט לא עלו חלופות לתיכון המערכת. המערכת פותחה באופן מודולרי בשימוש ארכיטקטורת microservice לפי דרישת הצבא וכפי שנהוג כיום בתעשייה לפתח מערכות גדולות.

15. תיאור התוכנה

15.1. סביבת העבודה

במהלך פיתוח הפרויקט השתמשתי במספר סביבות עבודה שונות:

- **Visual Studio Code** – עורך טקסט פופולרי שפיתחה מייקרוסופט, שתומך במגוון רחב של תוספים. לאור זאת, ניתן להפוך את העורך לסביבת פיתוח משולבת (IDE) באמצעות התקנת תוספים. עורך טקסט זה טוב במיוחד לפיתוח web, עם שימוש בטכנולוגיות מתאימות, בהן JavaScript, TypeScript, HTML, CSS, וכדומה. אפשר כמובן גם להשתמש בו לסוגי קבצים נוספים בהתאם לתוספים שמתקינים. למשל, השתמשתי בו בתוספי YAML, Docker, Kubernetes, GraphQL, וכו'. השימוש בעורך היה ל-microservices מבוסס Node.js ולקבצי קונפיגורציה ומניפסט שונים (עבור Docker, Kubernetes, וכדומה).
- **PyCharm** – סביבת פיתוח משולבת (IDE) לשפת התכנות Python שפיתחה החברה IntelliJ. יש לסביבה תמיכה בהתקנת תוספים, שכן היא מבוססת על העורך המפורסם IntelliJ IDEA. השתמשתי בסביבה זו לפיתוח ה-microservices מבוסס Python.
- **Docker** – פלטפורמה ליצירת תמונות וקונטיינרים. השתמשתי בה להרצת המערכת ולאריזת ה-microservices לתמונות.

- **Docker Compose** – תוסף ל-Docker המאפשר שליטה על הרצת קונטיינרים מקומית על ידי קובצי קונפיגורציה (ב-YAML) במקום שימוש ב-CLI. למעשה כלי זה נועד לפיתוח בעיקר, על מנת להקל על המשתמש בהרצת קונטיינרים.
- **NPM** – מנהל החבילות (package manager) של Node.js. השתמשתי בו ב-microservices מבוססי Node.js בפרויקט לניהול ה-dependencies וכדומה.
- **Poetry** – מנהל חבילות (package manager) לשפת התכנות Python. השתמשתי בו ב-microservices מבוססי Python בפרויקט לניהול ה-dependencies וכדומה.

15.2. שפות תכנות

בשביל פיתוח ה-microservices השתמשתי בשתי שפות תכנות עיקריות:

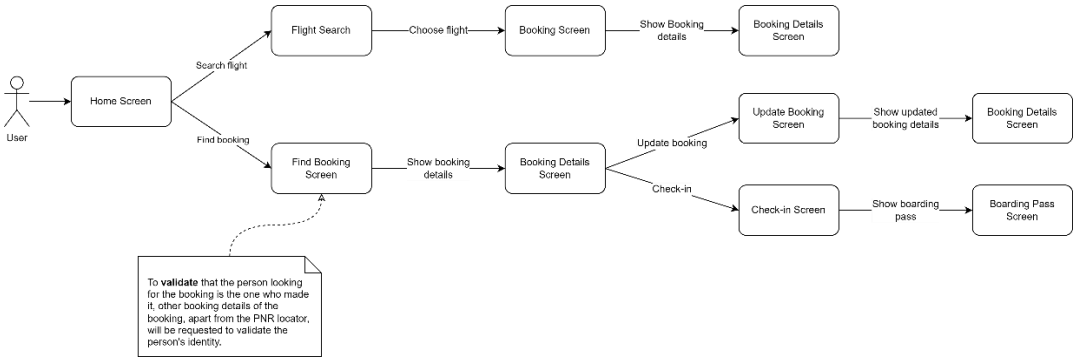
- **TypeScript** – שפת superset לשפת התכנות JavaScript המוסיפה תמיכה ב-Compilation time types שפותחה על ידי מייקרוסופט. פרויקט השתמשתי בה לכתיבת ה-Booking Service, שמשתמש ב-Express בשביל שרת ה-HTTP ורץ בסביבת Node.js.
- **Python** – שפה לשימוש כללי נפוצה ומבוססת. זוהי שפה מפורשת, כלומר לא נדרשת קומפילציה, אלא הקוד רץ כפי שהוא באמצעות שימוש במפרש. השתמשתי בה לפיתוח ה-Login Service וה-Flights Service עם framework בשם FastAPI על גבי שרת ה-ASGI (סטנדרט למימוש שרתי HTTP ב-Python) Uvicorn.

16. תיאור מסכים

מאחר שהפרויקט לא כולל ממשק משתמש אלא מתמקד בצד השרת, אין מסכים שניתן לתאר.

17. תרשים זרימת מסכים

למרות שאין לי בפרויקט ממשק משתמש, להלן תרשים זרימת מסכים לממשק משתמש אפשרי:



18. תפקידי המסכים

מאחר שהפרויקט לא כולל ממשק משתמש אלא מתמקד בצד השרת, אין מסכים שניתן לתאר.

19. תיאור מסך הפתיחה

מאחר שהפרויקט לא כולל ממשק משתמש אלא מתמקד בצד השרת, אין מסך פתיחה שניתן לתאר.

20. מסכים במערכת

מאחר שהפרויקט לא כולל ממשק משתמש אלא מתמקד בצד השרת, אין מסכים שניתן לתאר.

21. הסבר אלמנטי תצוגה

מאחר שהפרויקט לא כולל ממשק משתמש אלא מתמקד בצד השרת, אין אלמנטי תצוגה שניתן לתאר.

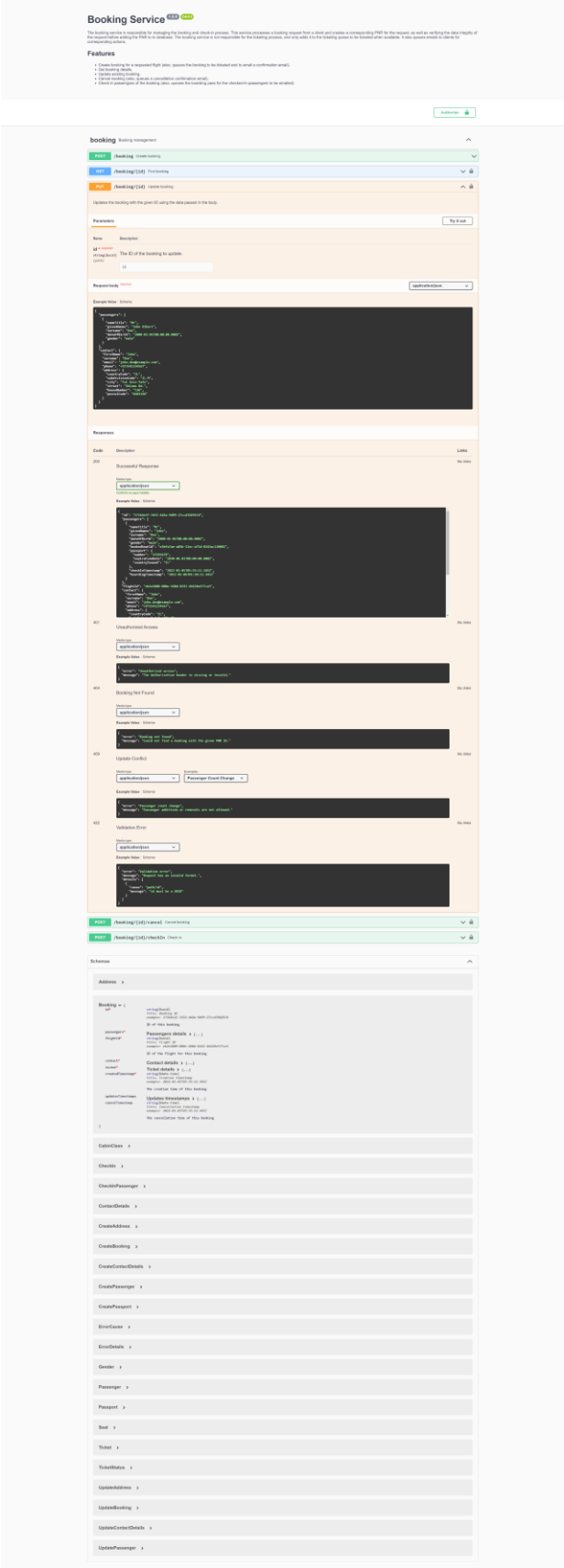
22. הודעות למשתמש

מאחר שהפרויקט לא כולל ממשק משתמש אלא מתמקד בצד השרת, אין הודעות למשתמש שניתן לתאר.

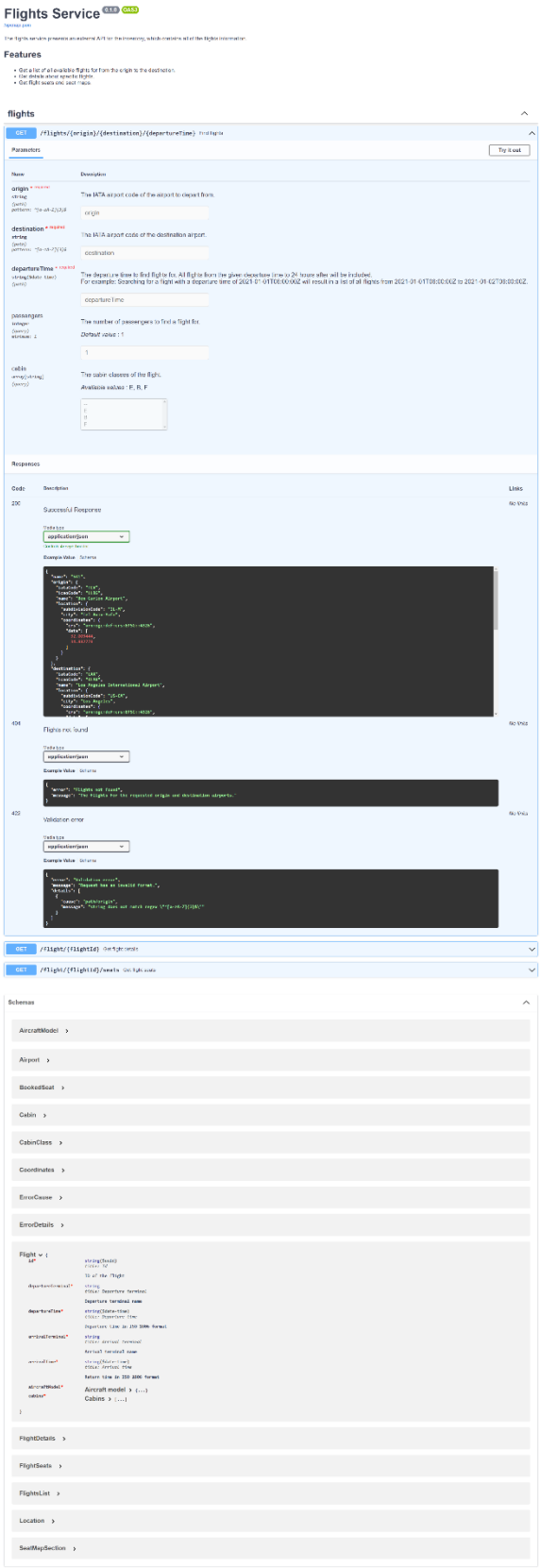
23. ממשק משתמש

כחלק מה-REST API שחושפת המערכת, יש גם Swagger UI עבור הממשק הציבורי. Swagger UI הוא ממשק משתמש גרפי סטנדרטי עבור ממשקי REST API שמשתמש במפרט/סטנדרט של OpenAPI (בעבר Swagger) לתיעוד ממשקי REST API. את הממשק הגרפי עצמו לא כתבתי, אלא כתבתי רק את הקונפיגורציה (תיעוד של OpenAPI) להצגה על ידי Swagger UI. בהמשך יופיעו צילומי מסך לדוגמא של הממשק.

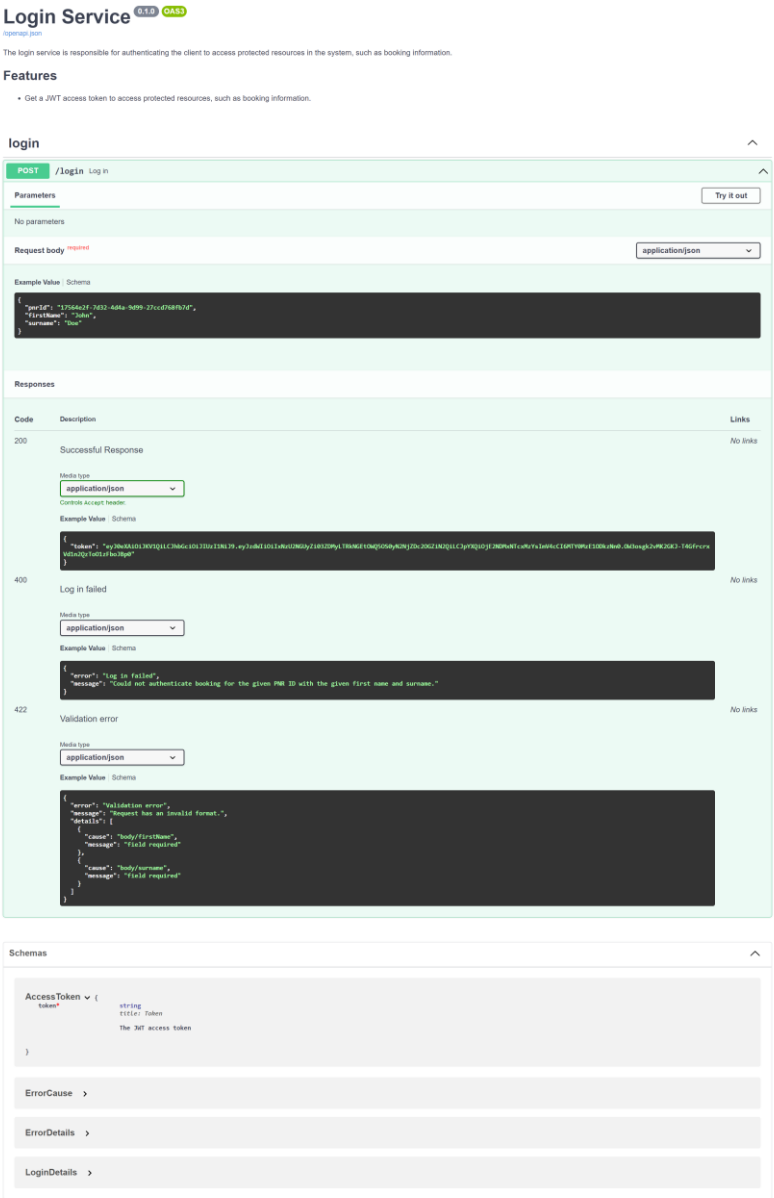
23.1. ניהול הזמנות



23.2. מידע על טיסות



23.3. התחברות



24. קוד התוכנית

לאור סדר הגודל של הפרויקט, לא אצרך בספר את הקוד של המערכת. הקוד יצורף בקובץ מכווץ בנפרד לספר הפרויקט, וזמין גם ב-GitHub repository שלי בכתובת:

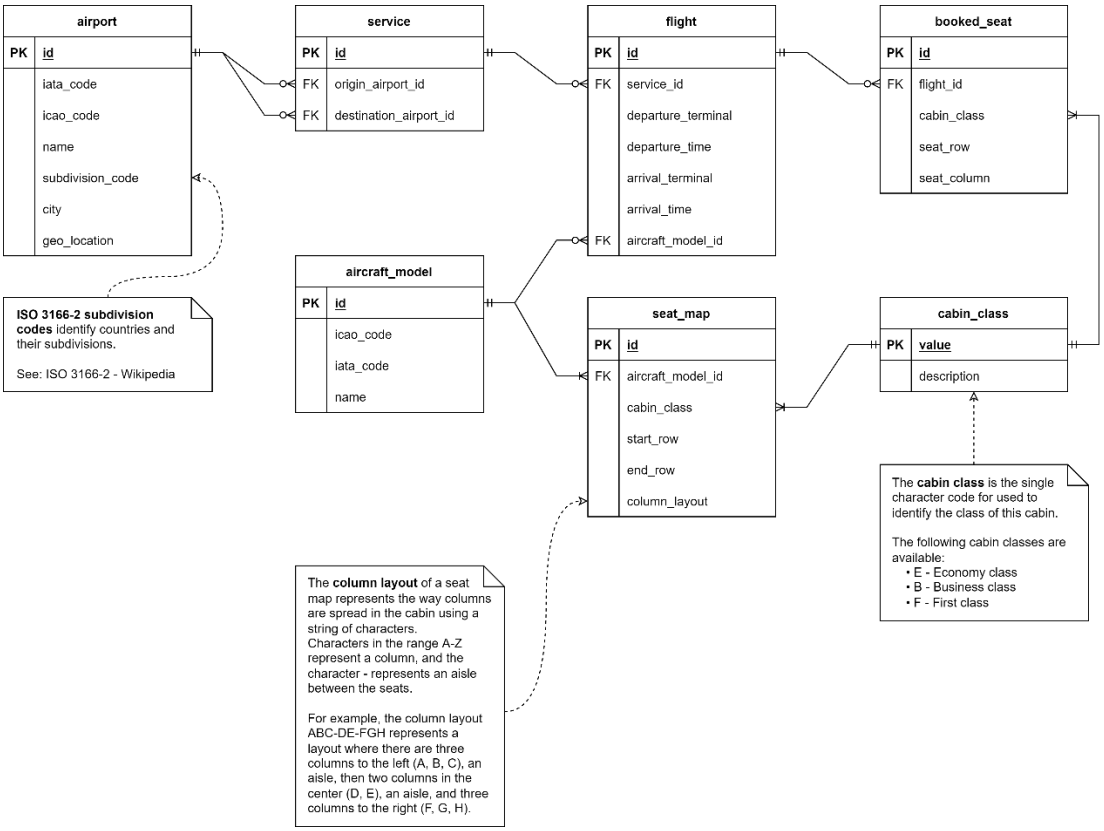
https://github.com/idos2002/skyline-crs

25. תיאור מסדי הנתונים

המערכת עושה שימוש בשני בסיסי נתונים: PostgreSQL עבור נתוני הטיסות (inventory), ו-MongoDB עבור ה-PNR-ים (PNR database).

25.1. מסד נתונים: Inventory

להלן דיאגרמת Entity Relationship (ERD) טבלאית:



להלן פירוט על הישויות:

- **airport** – פרטים של שדה תעופה אמיתי. מכיל את קודי ה-IATA וה-ICAO (מזהים סטנדרטיים) ופרטים נוספים, כמו שם השדה והמיקום שלו.
- **service** – שירות של חברת התעופה. שירות הוא למעשה מסלול קבוע בין שדה תעופה מקור לשדה תעופה יעד שמספקת חברת התעופה טיסות עבורו. לכל שירות קיים מספר מזהה, שלרוב יצורף למזהה של חברת התעופה. למשל, השירות LY1 של חברת התעופה אל על הוא בין תל אביב (TLV) לניו יורק (JFK).
- **flight** – טיסה יחידה במערכת. לכל טיסה קיים מזהה ציבורי מסוג UUID, וכוללת פרטים כמו זמני ההמראה וההגעה, טרמינלי ההמראה וההגעה, מודל המטוס של הטיסה, ורשימה של הכיסאות שהוזמנו בטיסה.
- **aircraft_model** – מודל מטוס של חברת התעופה. מכיל פרטים כמו מזהים סטנדרטיים של המודל ואת סדר הישיבה במטוס.

- **seat_map** – מייצג אזור של מושבים במטוס, והאופן בו הם מסודרים. כל סדר ישיבה במטוס מורכב ממספר seat_map. מכיל את המחלקה בו הוא נמצא (מחלקה ראשונה, עסקים וכו'), את השורה ממנה הוא מתחיל ואת השורה בה הוא נגמר, ואת מתווה המושבים. מתווה מושבים מיוצג כמחרוזת, כאשר כל אות במחרוזת מייצגת את שם העמודה של המושבים, התו "-" מייצג מעבר בין מושבים, והתו "#" מייצג שקיימת שם עמודה, אבל אין שם מושבים. למשל, ABC-###-HJK מייצג סידור ישיבה בו קיימות 9 עמודות, כאשר בין כל 3 עמודות יש מעבר, ובעמודות האמצעיות אין מושבים.
- **booked_seat** – מושב תפוס בטיסה. לכל מושב יש מזהה UUID ציבורי, ומכיל פרטים לגבי המושב, כמו המחלקה בו הוא נמצא והמיקום שלו במטוס (שורה ועמודה). לכל מושב יש גם ייחוס לטיסה בה הוא נמצא.
- **cabin_class** – סוג המחלקה במטוס (הסוג של ה-cabin). למעשה זוהי טבלה המשמשת כ-enumeration.

להלן הגדרת הטבלאות ב-SQL:

```
CREATE TABLE airport (
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    iata_code text NOT NULL,
    -- Check if it is a valid airport iata code, e.g. TLV, LAX, etc.
    CHECK (iata_code ~ '\A[A-Z]{3}\Z'),
    icao_code text NOT NULL,
    -- Check if it is a valid airport icao code, e.g. LLBG, KLAX, etc.
    CHECK (icao_code ~ '\A[A-Z]{4}\Z'),
    name text NOT NULL,
    subdivision_code text NOT NULL,
    -- Check if it is a valid ISO 3166-2 subdivision code, e.g. IL-M, US-
CA, etc.
    CHECK (subdivision_code ~ '\A[A-Z]{2}-[A-Z0-9]{1,3}\Z'),
    city text NOT NULL,
    geo_location geography(point) NOT NULL,
    UNIQUE (iata_code, icao_code)
);

CREATE TABLE service (
    id integer PRIMARY KEY,
    origin_airport_id integer NOT NULL REFERENCES airport (id),
    destination_airport_id integer NOT NULL REFERENCES airport (id),
    UNIQUE (origin_airport_id, destination_airport_id)
);

CREATE TABLE aircraft_model (
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    icao_code text NOT NULL,
    -- Check if it is a valid aircraft icao code, e.g. A5, B487, etc.
    CHECK (icao_code ~ '\A[A-Z0-9]{2,4}\Z'),
    iata_code text NOT NULL,
    -- Check if it is a valid aircraft iata code, e.g. A4F, 313, etc.
```

```
    CHECK (iata_code ~ '\A[A-Z0-9]{3}\Z'),
    name text NOT NULL,
    UNIQUE (icao_code, iata_code)
);

CREATE TABLE seat_map (
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    aircraft_model_id integer REFERENCES aircraft_model (id),
    cabin_class text REFERENCES cabin_class (value) NOT NULL,
    start_row integer NOT NULL CHECK (start_row > 0),
    end_row integer NOT NULL CHECK (end_row > 0),
    CHECK (start_row <= end_row),
    column_layout text NOT NULL,
    -- Check if column layout is in the correct form, e.g. ABC-EF-GHI, ABC,
ABC-DEF, etc.)
    CHECK (column_layout ~ '\A[A-Z#]+(?:-[A-Z#]+)*\Z'),
    UNIQUE (aircraft_model_id, start_row, end_row)
);

CREATE TABLE flight (
    id uuid PRIMARY KEY DEFAULT uuid_generate_v1(),
    service_id integer NOT NULL REFERENCES service (id),
    departure_terminal text NOT NULL,
    departure_time timestamptz NOT NULL,
    arrival_terminal text NOT NULL,
    arrival_time timestamptz NOT NULL,
    CHECK (departure_time < arrival_time),
    aircraft_model_id integer NOT NULL REFERENCES aircraft_model (id)
);

CREATE TABLE booked_seat (
    id uuid PRIMARY KEY DEFAULT uuid_generate_v1(),
    flight_id uuid NOT NULL REFERENCES flight (id),
    cabin_class text REFERENCES cabin_class (value) NOT NULL,
    seat_row integer NOT NULL CHECK (seat_row > 0),
    seat_column text NOT NULL,
    -- Check if the seat's column is a single uppercase letter (A-Z).
    CHECK (seat_column ~ '\A[A-Z]\Z'),
    UNIQUE (flight_id, seat_row, seat_column)
);

CREATE TABLE cabin_class (
    value text PRIMARY KEY,
    description text NOT NULL
);
INSERT INTO cabin_class VALUES
    ('E', 'Economy class'),
    ('B', 'Business class'),
    ('F', 'First class');
```

קיימת גם פונקציה המוגדרת במסד הנתונים:

```
CREATE FUNCTION column_layout_seats_count(column_layout text) RETURNS
integer
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT
RETURN char_length(translate(column_layout, '-#', ''));
```

הפונקציה מקבלת מחרוזת column layout ומחזירה את מספר הכיסאות שבשורה עבור אותו column layout.

כמו כן, קיימים מספר views במסד הנתונים:

```
CREATE VIEW cabin_seats_count AS
    SELECT aircraft_model_id,
           cabin_class,
           sum((end_row - start_row + 1) *
column_layout_seats_count(column_layout)) AS seat_count
    FROM seat_map
    GROUP BY aircraft_model_id, cabin_class;
```

ה-view שלעיל מציג כמה מושבים קיימים לכל מחלקה בדגם מטוס. להלן תיאור של הסכמה שלו:

עמודה	טיפוס	תיאור
aircraft_model_id	integer	מזהה של מודל המטוס.
cabin_class	text	המחלקה עליה מדובר.
seat_count	integer	מספר המושבים במחלקה.

```
CREATE VIEW available_flight_seats_count AS
    WITH flight_cabin_class AS (
        SELECT flight.id AS flight_id,
               flight.aircraft_model_id AS aircraft_model_id,
               cabin_class.value AS cabin_class
        FROM flight CROSS JOIN cabin_class
    ), booked_seats_count AS (
        SELECT flight.id AS flight_id,
               booked_seat.cabin_class AS cabin_class,
               count(*) AS booked_seats_count
        FROM flight INNER JOIN booked_seat ON flight.id =
booked_seat.flight_id
        GROUP BY flight.id, booked_seat.cabin_class
    )
    SELECT flight_cabin_class.flight_id AS flight_id,
           flight_cabin_class.cabin_class AS cabin_class,
```

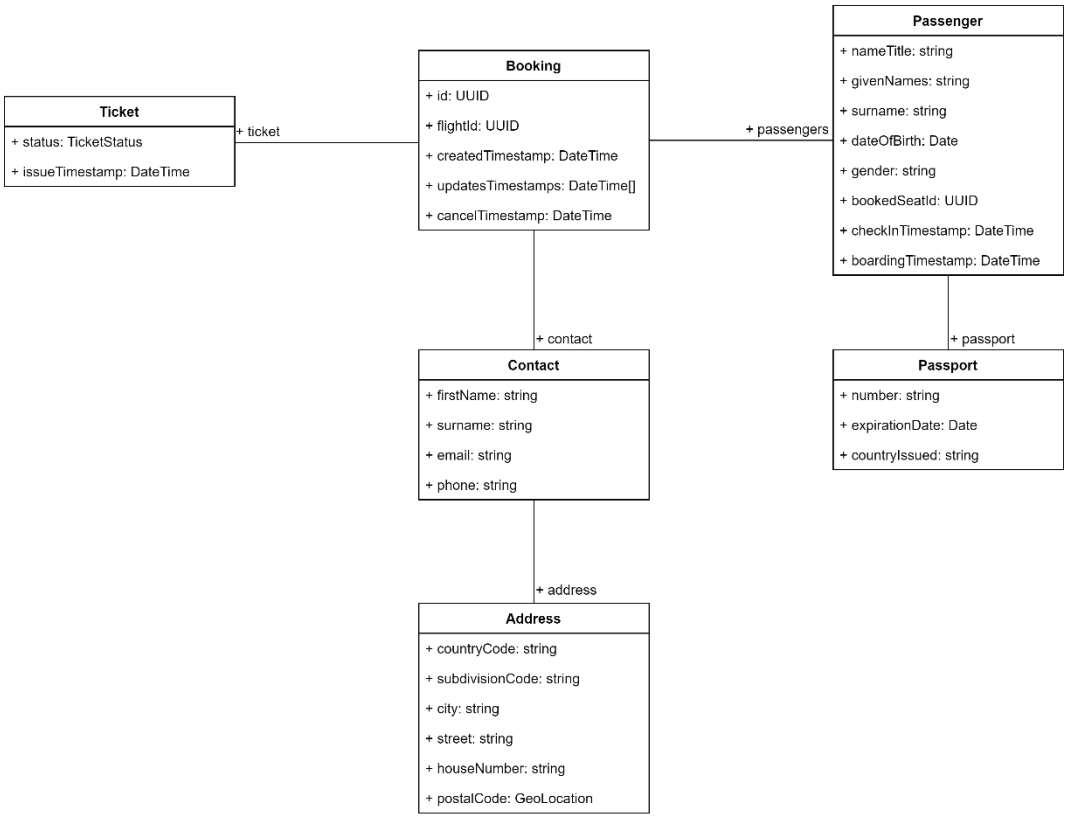
```
            cabin_seats_count.seat_count -
coalesce(booked_seats_count.booked_seats_count, 0) AS
available_seats_count,
            cabin_seats_count.seat_count AS total_seats_count
    FROM flight_cabin_class
         LEFT JOIN booked_seats_count
                   ON flight_cabin_class.flight_id =
booked_seats_count.flight_id
                   AND flight_cabin_class.cabin_class =
booked_seats_count.cabin_class
         LEFT JOIN cabin_seats_count
                   ON flight_cabin_class.aircraft_model_id =
cabin_seats_count.aircraft_model_id
                   AND flight_cabin_class.cabin_class =
cabin_seats_count.cabin_class
    WHERE cabin_seats_count.seat_count > 0;
```

ה-view שלעיל מציג כמה מושבים פנויים יש לכל מחלקה בטיסות שבמסד הנתונים. להלן תיאור של הסכמה שלו:

עמודה	טיפוס	תיאור
flight_id	integer	מזהה של הטיסה.
cabin_class	text	המחלקה עליה מדובר.
available_seats_count	integer	מספר המושבים הפנויים במחלקה.
total_seats_count	integer	מספר המושבים הכולל במחלקה.

25.2. מסד נתונים: PNR Database

מאחר שמסד נתונים זה (MongoDB) הוא מסד נתונים מסוג NoSQL, כלומר אינו טבלאי, ושומר את המידע ב-documents, לא ניתן לתאר את הסכמה שלו בתרשים ERD. למרות זאת, להלן תרשים domain model של המסד:



להלן פירוט על הישויות:

- **Booking** – הזמנה (PNR) במערכת. לכל הזמנה מזהה UUID ציבורי ומספר שדות למעקב אחר זמנים בהם נעשו שינויים בהזמנה. הזמנות מכילות גם רשימת נוסעים של ההזמנה, פרטי התקשרות, וכרטיס הטיסה.
- **Passenger** – נוסע השייך להזמנה. מכיל את פרטי הנוסע, כמו שם, תאריך לידה, מגדר וכדומה. נוסף לכך, לכל נוסע יש גם שדה של דרכון, שפרטיו ימולאו בזמן תהליך הצ'ק-אין.
- **Passport** – דרכון של נוסע. מכיל פרטים הקשורים לדרכון, בהם מספר הדרכון, תאריך התפוגה והמדינה בה הונפק.
- **Ticket** – מכיל פרטים לגבי כרטיס הטיסה של נוסע, בהם סטטוס הכרטיס והזמן בו הוא תוקף.
- **Contact** – פרטי התקשרות להזמנה. מכיל פרטים כמו שם, פרטי התקשרות (אימייל, מספר טלפון), וכתובת.
- **Address** – כתובת לפרטי התקשרות. מכיל פרטים כמו מזהה המדינה, העיר, הרחוב, המיקוד וכדומה.

להלן הגדרת הסכמה, שכוללת גם תיאורים לגבי השדות השונים:

```
{
  "bsonType":"object",
  "required":[
    "passengers",
    "flightId",
    "contact",
    "ticket",
    "createdTimestamp"
  ],
  "properties":{
    "_id":{
      "description":"Unique ID or record locator for this PNR represented
by a standard UUIDv1 representation. (Required)",
      "bsonType":"binData"
    },
    "passengers":{
      "description":"List of passengers details for this itinerary.
(Required)",
      "bsonType":"array",
      "minItems":1,
      "items":{
        "description":"Passenger details.",
        "bsonType":"object",
        "required":[
          "givenNames",
          "surname",
          "dateOfBirth",
          "gender",
          "bookedSeatId"
        ],
        "properties":{
          "nameTitle":{
            "description":"Name title of the passenger, such as Mr, Mrs,
etc.",
            "bsonType":"string"
          },
          "givenNames":{
            "description":"Given names of the passenger (first name and
middle names) as written in the passport. (Required)",
            "bsonType":"string"
          },
          "surname":{
            "description":"Surname (or last name) of the passenger as
written in passport. (Required)",
            "bsonType":"string"
          },
          "dateOfBirth":{
            "description":"Date of birth of the passenger as written in the
passport. (Required)",
            "bsonType":"date"
```

```
          },
          "gender":{
            "description":"Gender of the passenger as written in the
passport. (Required)",
            "enum":[
              "male",
              "female",
              "other",
              "unspecified"
            ]
          },
          "bookedSeatId":{
            "description":"Booked seat ID of the passenger's seat in the
flight in standard UUID format.",
            "bsonType":"binData"
          },
          "passport":{
            "description":"Passport details for this passenger.",
            "bsonType":"object",
            "required":[
              "number",
              "expirationDate",
              "countryIssued"
            ],
            "properties":{
              "number":{
                "description":"The passport number.",
                "bsonType":"string"
              },
              "expirationDate":{
                "description":"The expiration date of the passport.",
                "bsonType":"date"
              },
              "countryIssued":{
                "description":"The ISO 3166-1 alpha-2 country code of the
country that issued this passport.",
                "bsonType":"string",
                "pattern":"^[A-Z]{2}$"
              }
            }
          },
          "checkInTimestamp":{
            "description":"Check-in timestamp for this passenger.",
            "bsonType":"date"
          },
          "boardingTimestamp":{
            "description":"Plane boarding timestamp for this passenger.",
            "bsonType":"date"
          }
        }
      }
```

```
    },
    "flightId":{
      "description":"Flight ID of the flight the booking was made for in
standard UUID format. (Required)",
      "bsonType":"binData"
    },
    "contact":{
      "description":"Contact details of the person who made the booking.
(Required)",
      "bsonType":"object",
      "required":[
        "firstName",
        "surname",
        "email",
        "phone",
        "address"
      ],
      "properties":{
        "firstName":{
          "description":"First name of the person who made the booking.
(Required)",
          "bsonType":"string"
        },
        "surname":{
          "description":"Surname (or last name) of the person who made the
booking. (Required)",
          "bsonType":"string"
        },
        "email":{
          "description":"Email address. (Required)",
          "bsonType":"string"
        },
        "phone":{
          "description":"International phone number. (Required)",
          "bsonType":"string"
        },
        "address":{
          "description":"Address of the person who made the booking.
(Required)",
          "bsonType":"object",
          "required":[
            "countryCode",
            "city",
            "street",
            "houseNumber",
            "postalCode"
          ],
          "properties":{
            "countryCode":{
              "description":"ISO 3166-1 alpha-2 country code. (Required)",
              "bsonType":"string",
```

```
              "pattern":"^[A-Z]{2}$"
            },
            "subdivisionCode":{
              "description":"ISO 3166-2 subdivision code.",
              "bsonType":"string",
              "pattern":"^[A-Z]{2}-[A-Z0-9]{1,3}$"
            },
            "city":{
              "description":"City name. (Required)",
              "bsonType":"string"
            },
            "street":{
              "description":"Street name. (Required)",
              "bsonType":"string"
            },
            "houseNumber":{
              "description":"House number. (Required)",
              "bsonType":"string"
            },
            "postalCode":{
              "description":"Postal code. (Required)",
              "bsonType":"string"
            }
          }
        }
      }
    },
    "ticket":{
      "description":"Information about the ticketing status of this PNR.
(Required)",
      "bsonType":"object",
      "required":[
        "status"
      ],
      "properties":{
        "status":{
          "description":"Status of the ticket to be issued for this PNR.
(Required)",
          "enum":[
            "pending",
            "issued",
            "canceled"
          ]
        },
        "issueTimestamp":{
          "description":"Timestamp of the time a ticket has been issued for
this PNR.",
          "bsonType":"date"
        }
      }
    },
```

```
    "createdTimestamp":{
      "description":"The timestamp on which this PNR has been created.
(Required)",
      "bsonType":"date"
    },
    "updatesTimestamps":{
      "description":"List of the timestamps on which this PNR has been
updated.",
      "bsonType":"array",
      "items":{
        "description":"Updates timestamp",
        "bsonType":"date"
      }
    },
    "cancelTimestamp":{
      "description":"Timestamp on which this booking (PNR) has been
canceled.",
      "bsonType":"date"
    }
  }
}
```

הגדרת הסכמה כתובה בפורמט JSON על פי הספציפיקציה של MongoDB, כפי שמתואר בקישור הבא:

https://www.mongodb.com/docs/manual/core/schema-validation

26. מדריך למשתמש

הפרויקט מכיל מספר סוגי מדריכים למשתמש. עבור מדריך לשימוש ב-REST API הציבורי, ראו פרק 13.

כמו כן, כתבתי מספר קבצי README.md באנגלית עבור החלקים השונים בפרויקט, המסבירים כיצד להשתמש ולפתח עבור המערכת. לכן, אעתיק את תוכנם לכאן.

26.1. הרצת המערכת – README.md גלובלי

Skyline CRS is a scalable computer reservation system (CRS) for a fictional airline called Skyline, built for the cloud. The project is deployed using Docker containers and also provides a basic Kubernetes configuration.

The system exposes a public REST API and uses JWT token-based authentication, hence allowing integration with other travel agencies. There is also a web application for the airline, from which the clients may make all actions.

Documentation

You can find the all of the services' documentation here: Skyline CRS Documentation.

Usage

This is a monorepo for the entirety of the Skyline CRS project. Since this project centers around containerization, it is recommended to use Docker Compose for running the project with the provided `docker-compose.yml` file.

Before continuing, clone this repository by running:

```
git clone https://github.com/idos2002/skyline-crs.git
```

Using Docker Compose

To run this project using Docker Compose, first make sure that Docker and Docker Compose are installed on the system.

Then, create a file named `email.env.local` containing the email service provider details and account credentials for sending emails.

Example file contents using a Zoho Mail account:

```
SKYLINE_EMAIL_ADDRESS=example@zohomail.com
SKYLINE_SMTP_HOST=smtp.zoho.com
SKYLINE_SMTP_PORT=465
SKYLINE_SMTP_USERNAME=example@zohomail.com
SKYLINE_SMTP_PASSWORD=password
```

Check the the email service's README file for more information about the required environment variables.

Finally, to start the project locally, run in the root project directory:

```
docker-compose up -d --build
```

Using Docker

Since the project is containerized, you may choose to build the Docker images of the services individually, or pull them from Docker Hub where they are hosted under the namespace idos2002. All hosted images relating to this project start with the prefix skyline-.

For using the images, consult their respective README file. Note that most of the services depend on other services to be running, so make sure to have the needed containers running as well.

Development

For information regarding local development of Skyline CRS services, check the services' README files.

26.2. הרצה ופיתוח מקומי של Booking Service

The **booking service** is responsible for managing the booking and check-in process. This service processes a booking request from a client and creates a corresponding PNR for the request, as well as verifying the data integrity of the request before adding the PNR to to database.

The booking service is not responsible for the ticketing process, and only adds it to the ticketing queue to be ticketed when available. It also queues emails to clients for corresponding actions.

Features

- Create booking for a requested flight (also, queues the booking to be ticketed and to email a confirmation email).
- Get booking details.
- Update existing booking.
- Cancel booking (also, queues a cancellation confirmation email).
- Check in passengers of the booking (also, queues the boarding pass for the checked-in passengers to be emailed)

Usage

This is a Docker based Node.js project, therefore it should be run in a container.

Using Docker Compose

To run this service locally, along with its dependencies, Docker Compose may be used.

Navigate to the root Skyline CRS directory and run:

```
docker-compose up -d --build
```

This will run the entire Skyline CRS system locally using Docker Compose. You may modify `docker-compose.yml` to your liking, along with its respective .env file.

Using Docker

To build the image directly, navigate to the project's directory and run:

```
docker build -t skyline-booking .
```

And to run the image in a new container run:

```
docker run -dp 80:80 \
  -e SKYLINE_ACCESS_TOKEN_SECRET=<Access token secret> \
  -e SKYLINE_INVENTORY_MANAGER_URL=<Inventory manager URL> \
  -e SKYLINE_PNR_DB_URI=<PNR database URI> \
  -e SKYLINE_RABBITMQ_URI=<RabbitMQ URI> \
  -e SKYLINE_TICKET_EXCHANGE_NAME=<Ticket exchange name> \
  -e SKYLINE_EMAIL_EXCHANGE_NAME=<Email exchange name> \
  skyline-booking
```

Note that this service depends on three other services:

- Inventory manager - Used to retrieve flights data and manage booked seats.
- PNR database - Used to store and manage bookings (PNRs).
- RabbitMQ - Used by Skyline CRS to manage queues. It is used for queueing bookings for ticketing and queueing emails.

API Documentation

The service's API is documented at the Skyline CRS documentation.

If the service is up and running, there is an interactive API documentation UI, which uses Swagger UI, and is accessible at the `/docs` endpoint of the service.

Environment Variables

The service is configured using environment variables. Note that some environment variables are required, and the image will not run without them.

`SKYLINE_ACCESS_TOKEN_SECRET` (Required)

The secret to use for signing the generated JWT access tokens.

`SKYLINE_INVENTORY_MANAGER_URL` (Required)

Inventory manager URL to query the flights information from and manage booked seats.
Example values:

- http://localhost:8080/v1/graphql

- http://inventory-manager/v1/graphql

`SKYLINE_RABBITMQ_URI` (Required)

URI to the RabbitMQ service/cluster to use for queueing bookings for ticketing and emails. The URI *should* contain the vhost to connect to.
Example values:

- `amqp://username:password@localhost:5672/skyline`
- `amqp://username:password@pnr:5672/skyline`

`SKYLINE_TICKET_EXCHANGE_NAME` (Required)

The name of the RabbitMQ ticket exchange. This exchange should be a topic exchange.
Example value: `ticket`.

`SKYLINE_EMAIL_EXCHANGE_NAME` (Required)

The name of the RabbitMQ email exchange. This exchange should be a topic exchange.
Example value: `email`.

`SKYLINE_PNR_DB_URI` (Required)

The PNR database MongoDB URI for managing the bookings' PNRs.
Example values:

- `mongodb://username:password@localhost:27017`
- `mongodb://username:password@pnr:27017`

`SKYLINE_PNR_DB_NAME`
The name of the PNR database for the given database URI.
Default: `pnr`.

`SKYLINE_PNR_DB_COLLECTION_NAME`
The name of the PNR database's PNR collection for the given database URI.
Default: `pnrs`.

`SKYLINE_TICKET_BOOKING_ROUTING_KEY`
The routing key to use for queueing bookings to be ticketed using the RabbitMQ ticket exchange.
Default: `ticket.booking`

`SKYLINE_EMAIL_BOOKING_CONFIRMATION_ROUTING_KEY`
The routing key to use for queueing confirmation emails using the RabbitMQ email exchange.
Default: `email.booking.confirmation`

`SKYLINE_EMAIL_BOOKING_CANCELLATION_ROUTING_KEY`
The routing key to use for queueing cancellation confirmation emails using the RabbitMQ email exchange.
Default: `email.booking.cancel`

`SKYLINE_EMAIL_BOARDING_PASS_ROUTING_KEY`
The routing key to use for queueing boarding pass emails using the RabbitMQ email exchange.
Default: `email.boarding.ticket`

`SKYLINE_PORT`
The TCP port for the service to listen on for incoming requests.
Default: 80.

`SKYLINE_LOG_LEVEL`
Log level of the service. Available values are: `TRACE, DEBUG, INFO, WARNING, ERROR, FATAL.`
Default: INFO.

Development

This project uses TypeScript and is based on the Node.js framework Express. Since the project is Docker based, it is not required to install Node.js on the system, but it is **highly recommended** for local development.

Requirements

- Node.js version 16.14.x (LTS).
- A running instance of inventory manager along with its dependencies.
- A running instance of the PNR database along with its dependencies.
- An available instance or cluster of RabbitMQ.

Installation

To install all dependencies, navigate to the project's directory and run:

```
npm install
```

If you would like to validate and not modify the contents of `package-lock.json`, run:

```
npm ci
```

Running Locally

In order to run the project locally, make sure to first set all required environment variables for the current shell session.

To start the development server locally using ts-node, make sure you are inside the project's directory and run:

```
npm start
```

To start the development server in watch mode, run instead:

```
npm run start:dev
```

This will watch for file changes and restart the server accordingly.

Since the project uses bunyan as the logging library, to print the JSON logs in a human-readable manner, use the following command:

```
npm start | npx bunyan
```

This also works with `start:dev`, etc. For more information about bunyan's CLI, see the documentation for the package.

Tools

The project uses multiple tools to assure code quality and correctness. It is required to use these tools to ensure code quality before any commit!

- Prettier - Prettier is an opinionated code formatter with support for: JavaScript (including experimental features), TypeScript, JSON etc. It removes all original styling and ensures that all outputted code conforms to a consistent style.
  Usage: `npm run format`
- ESLint - ESLint is an open source JavaScript linting utility, which also supports TypeScript.
  Usage: `npm run lint`
- Jest - The testing framework used for the project.
  Usage:
  - `npm test` - Unit tests or specs.
  - `npm run test:e2e` - End-to-end tests.

26.3. הרצה ופיתוח מקומי של Login Service

The **login service** is responsible for authenticating the client to access protected resources in the system, such as booking information.

Features

- Get a JWT access token to access protected resources, such as booking information.

Usage

This is a Docker based Python project, therefore it should be run in a container.

Using Docker Compose

To run this service locally, along with its dependencies, Docker Compose may be used.

Navigate to the root Skyline CRS directory and run:

```
docker-compose up -d --build
```

This will run the entire Skyline CRS system locally using Docker Compose. You may modify `docker-compose.yml` to your liking, along with its respective .env file.

Using Docker

To build the image directly, navigate to the project's directory and run:

```
docker build -t skyline-login .
```

And to run the image in a new container run:

```
docker run -dp 80:80 \
  -e SKYLINE_PNR_DB_URL=<PNR database URL> \
  -e SKYLINE_ACCESS_TOKEN_SECRET=<Access token secret> \
  skyline-flights
```

Note that this service depends on the PNR database to retrieve data. It is still usable to a degree by providing some URL to the `SKYLINE_PNR_DB_URL` and the `SKYLINE_ACCESS_TOKEN_SECRET` environment variables, which will allow access to the `/docs` and `/redoc` endpoints, but all other requests will result in an error from the service.

API Documentation

The service's API is documented at the Skyline CRS documentation.

If the service is up and running, there are also two interactive API documentation UIs available for the service:

- Swagger UI - accessible at /docs.
- Redoc - accessible at `/redoc`.

Environment Variables

The service is configured using environment variables. Note that some environment variables are required, and the image will not run without them.

`SKYLINE_ACCESS_TOKEN_SECRET` (Required)

The secret to use for signing the generated JWT access tokens.

`SKYLINE_PNR_DB_URL` (Required)

The PNR database URL for validating login data.
Example values:

- `mongodb://username:password@localhost:27017`
- `mongodb://username:password@pnr:27017`

`SKYLINE_PNR_DB_NAME`
The name of the PNR database for the given database URL. **Default:** pnr.

`SKYLINE_PNR_DB_COLLECTION_NAME`
The name of the PNR database's PNR collection for the given database URL. **Default:** `pnrs`.

`SKYLINE_PORT`
The TCP port for the service to listen on for incoming requests.
Default: `80`.

`SKYLINE_LOG_LEVEL`
Sets the log level for the service. Available values are: `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`.
Default: `INFO`.

Development

This project is based on the Python framework FastAPI and uses the Python package manager Poetry. Since the project is Docker based, it is not required to install Python and Poetry on the system, but it is **highly recommended** for local development.

Requirements

- Python version 3.10 and up.
- Poetry version 1.1 and up.
- A running instance of inventory manager along with its dependencies.

Installation

To create a virtual environment with poetry and install all dependencies, navigate to the project's directory and run:

```
poetry install --no-root
```

This will create a new virtual environment and install all dependencies to it, except the project itself (hence the --no-root option).

To activate the virtual environment for the current shell run:

```
poetry shell
```

or use poetry run to execute commands inside the virtual environments without activating it. Check the Poetry documentation for more information.

Running Locally

In order to run the project locally, make sure to first set all required environment variables for the current shell session. The project uses the ASGI server Uvicorn for running the application.

To start the Uvicorn server locally, make sure you are at the projects directory and run:

```
uvicorn flights.main:app --port 80 --log-config logging.yaml --no-access-log
```

This will start the server on localhost:80 and will set up the logging for the application using the `logging.yaml` file.

Tools

The project uses multiple tools to assure code quality and correctness. Most of the tools are configured in the project's `pyproject.toml` file, except those who do not support this kind of configuration yet, and have their own configuration files, e.g. Flake8 with the configuration file `.flake8`. It is required to use these tools to ensure code quality before any commit!

- Black - An opinionated Python code formatter, used to keep the code formatting consistent.
  Example Usage: `black .`
- isort - Sorts, organizes and formats import statements in the project.
  Example usage: `isort .`
- Flake8 - A Python style guide enforcer and linter which wraps around the tools: PyFlakes, pycodestyle and Ned Batchelder's McCabe complexity checker.
  Example usage: `flake8`
- MyPy - A static type checker for Python which uses Python's type hints for type checking.
  Example usage: `mypy .`
- PyTest - The testing framework used for the project.
  Example usage: `pytest`
- Coverage.py - Coverage.py is a tool for measuring code coverage of Python programs. It is used together with PyTest to measure the tests' code coverage.
  Example usage: `coverage run -m pytest && coverage report`

26.4. הרצה ופיתוח מקומי של Flights Service

The **flights service** provides an external API for the inventory, which contains all the flights' information.

Features

- Get a list of all available flights for from the origin to the destination.
- Get details about specific flights.
- Get flight seats and seat maps.

Usage

This is a Docker based Python project, therefore it should be run in a container.

Using Docker Compose

To run this service locally, along with its dependencies, Docker Compose may be used.

Navigate to the root Skyline CRS directory and run:

```
docker-compose up -d --build
```

This will run the entire Skyline CRS system locally using Docker Compose. You may modify `docker-compose.yml` to your liking, along with its respective .env file.

Using Docker

To build the image directly, navigate to the project's directory and run:

```
docker build -t skyline-flights .
```

And to run the image in a new container run:

```
docker run -dp 80:80 -e SKYLINE_INVENTORY_MANAGER_URL=<Inventory manager URL> skyline-flights
```

Note that this service depends on the inventory manager to retrieve data. It is still usable to a degree by providing some URL to the `SKYLINE_INVENTORY_MANAGER_URL` environment variable, which will allow access to the `/docs` and `/redoc` endpoints, but all other requests will result in an error from the service.

API Documentation

The service's API is documented at the Skyline CRS documentation.

If the service is up and running, there are also two interactive API documentation UIs available for the service:

- Swagger UI - accessible at `/docs`.
- Redoc - accessible at `/redoc`.

Environment Variables

The service is configured using environment variables. Note that some environment variables are required, and the image will not run without them.

`SKYLINE_INVENTORY_MANAGER_URL` (Required)
Sets the inventory manager URL to query the inventory information from. Example values:

- `http://localhost:8080/v1/graphql`
- `http://inventory-manager/v1/graphql`

`SKYLINE_PORT`
Sets the TCP port for the service to listen on for incoming requests.
Default: `80`.

`SKYLINE_LOG_LEVEL`
Sets the log level for the service. Available values are: DEBUG, INFO, WARNING, ERROR, CRITICAL.
Default: `INFO`.

`SKYLINE_IATA_AIRLINE_CODE`
Sets the Skyline IATA airline's code. Note that it must conform to the IATA airline designator standard.
Regex: `/^[A-Z0-9]{2,3}$/`
Default: `SK`

`SKYLINE_ICAO_AIRLINE_CODE`
Sets the Skyline ICAO airline's code. Note that it must conform to the ICAO airline designator standard.
Regex: `/^[A-Z]{3}$/`
Default: `SKL`

Development

This project is based on the Python framework FastAPI and uses the Python package manager Poetry. Since the project is Docker based, it is not required to install Python and Poetry on the system, but it is **highly recommended** for local development.

Requirements

- Python version 3.10 and up.
- Poetry version 1.1 and up.
- A running instance of inventory manager along with its dependencies.

Installation

To create a virtual environment with poetry and install all dependencies, navigate to the project's directory and run:

```
poetry install --no-root
```

This will create a new virtual environment and install all dependencies to it, except the project itself (hence the --no-root option).

To activate the virtual environment for the current shell run:

```
poetry shell
```

or use poetry run to execute commands inside the virtual environments without activating it. Check the Poetry documentation for more information.

Running Locally

In order to run the project locally, make sure to first set all required environment variables for the current shell session. The project uses the ASGI server Uvicorn for running the application.

To start the Uvicorn server locally, make sure you are at the projects directory and run:

```
uvicorn flights.main:app --port 80 --log-config logging.yaml --no-access-log
```

This will start the server on localhost:80 and will set up the logging for the application using the `logging.yaml` file.

Tools

The project uses multiple tools to assure code quality and correctness. Most of the tools are configured in the project's pyproject.toml file, except those who do not support this kind of configuration yet, and have their own configuration files, e.g. Flake8 with the configuration file .flake8. It is required to use these tools to ensure code quality before any commit!

- Black - An opinionated Python code formatter, used to keep the code formatting consistent.
  Example Usage: `black .`
- isort - Sorts, organizes and formats import statements in the project.
  Example usage: `isort .`
- Flake8 - A Python style guide enforcer and linter which wraps around the tools: PyFlakes, pycodestyle and Ned Batchelder's McCabe complexity checker.
  Example usage: `flake8`
- MyPy - A static type checker for Python which uses Python's type hints for type checking.
  Example usage: `mypy .`
- PyTest - The testing framework used for the project.
  Example usage: `pytest`

- Coverage.py - Coverage.py is a tool for measuring code coverage of Python programs. It is used together with PyTest to measure the tests' code coverage.
  Example usage: `coverage run -m pytest && coverage report`

26.5. הרצה ופיתוח מקומי של Ticketing Service

The **ticketing service** is responsible for managing the ticketing process of new bookings. This service consumes ticketing messages from RabbitMQ of bookings to ticket, and in return updates the corresponding booking in the PNR database accordingly. The service also queues the flight tickets for the booking to be emailed.

Features

- Consumes ticketing messages from RabbitMQ.
- Updates ticketed bookings in the PNR database.
- Queues the flight tickets to be emailed.

Usage

This is a Docker based Node.js project, therefore it should be run in a container.

Using Docker Compose

To run this service locally, along with its dependencies, Docker Compose may be used.

Navigate to the root Skyline CRS directory and run:

```
docker-compose up -d --build
```

This will run the entire Skyline CRS system locally using Docker Compose. You may modify `docker-compose.yml` to your liking, along with its respective `.env` file.

Using Docker

To build the image directly, navigate to the project's directory and run:

```
docker build -t skyline-ticketing .
```

And to run the image in a new container run:

```
docker run -dp 80:80 \
  -e SKYLINE_RABBITMQ_URI=<RabbitMQ URI> \
  -e SKYLINE_TICKET_EXCHANGE_NAME=<Ticket exchange name> \
  -e SKYLINE_TICKET_QUEUE_NAME=<Ticket queue name> \
  -e SKYLINE_TICKET_BOOKING_BINDING_KEY=<Ticket booking binding key> \
  -e SKYLINE_EMAIL_EXCHANGE_NAME=<Email exchange name> \
  -e SKYLINE_DEAD_LETTER_EXCHANGE_NAME=<Dead letter exchange name> \
  -e SKYLINE_PNR_DB_URI=<PNR database URI> \
  skyline-ticketing
```

Note that this service depends on two other services:

- RabbitMQ - Used by Skyline CRS to manage queues. It is used to consume ticketing messages and queue tickets to be emailed.
- PNR database - Used to read and update booking (PNR) details when ticketing.

Environment Variables

The service is configured using environment variables. Note that some environment variables are required, and the image will not run without them.

SKYLINE_RABBITMQ_URI (Required)

URI to the RabbitMQ service/cluster to use for consuming the ticket queue and queuing flight ticket to be emailed. The URI *should* contain the vhost to connect to.
Example values:

- `amqp://username:password@localhost:5672/skyline`
- `amqp://username:password@pnr:5672/skyline`

`SKYLINE_TICKET_EXCHANGE_NAME` (Required)

The name of the RabbitMQ ticket exchange. This exchange should be a topic exchange.
Example value: `ticket`.

`SKYLINE_TICKET_QUEUE_NAME` (Required)

The name of the RabbitMQ ticket queue. This queue should be a durable queue with a dead letter exchange and routing key, as given by `SKYLINE_DEAD_LETTER_EXCHANGE_NAME` and `SKYLINE_DEAD_LETTER_ROUTING_KEY`.
Example value: `ticket`.

`SKYLINE_TICKET_BOOKING_BINDING_KEY` (Required)

The binding key of the RabbitMQ ticket exchange and the ticket queue.
Example value: ticket.#.

`SKYLINE_EMAIL_EXCHANGE_NAME` (Required)

The name of the RabbitMQ email exchange. This exchange should be a topic exchange.
Example value: `email`.

`SKYLINE_DEAD_LETTER_EXCHANGE_NAME` (Required)

The name of the RabbitMQ dead letter exchange for the ticket queue. This exchange should be a topic exchange.
Example value: `dlx`.

`SKYLINE_PNR_DB_URI` (Required)

The PNR database MongoDB URI for updating the bookings' PNRs.
Example values:

- `mongodb://username:password@localhost:27017`
- `mongodb://username:password@pnr:27017`

`SKYLINE_PNR_DB_NAME`
The name of the PNR database for the given database URL.
Default: `pnr`.

`SKYLINE_PNR_DB_COLLECTION_NAME`
The name of the PNR database's PNR collection for the given database URL.
Default: `pnrs`.

`SKYLINE_PREFETCH_COUNT`
The RabbitMQ consumer prefetch count to use for this service, when consuming from the given ticket queue. For more information, see: How to Optimize the RabbitMQ Prefetch Count.
Default: `1`.

`SKYLINE_TICKET_BOOKING_MESSAGE_TYPE`
The message type for ticketing messages in the given ticket queue.
Default: `ticket.booking`.

`SKYLINE_EMAIL_TICKET_ROUTING_KEY`
The routing key to use for queueing the flight tickets to be emailed through the RabbitMQ email exchange.
Default: `email.booking.ticket`

`SKYLINE_DEAD_LETTER_ROUTING_KEY`
The routing key to use for dead-lettered messages of the ticket queue. **Default:** `ticket.booking`

`SKYLINE_LOG_LEVEL`
Log level of the service. Available values are: `TRACE, DEBUG, INFO, WARNING, ERROR, FATAL`.
Default: `INFO`.

Development

This project uses TypeScript and uses amqplib for consuming messages from the ticket queue. Since the project is Docker based, it is not required to install Node.js on the system, but it is **highly recommended** for local development.

Requirements

- Node.js version 16.14.x (LTS).
- An available instance or cluster of RabbitMQ.
- A running instance of the PNR database along with its dependencies.

Installation

To install all dependencies, navigate to the project's directory and run:

```
npm install
```

If you would like to validate and not modify the contents of package-lock.json, run:

```
npm ci
```

Running Locally

In order to run the project locally, make sure to first set all required environment variables for the current shell session.

To start the development server locally using ts-node, make sure you are inside the project's directory and run:

```
npm start
```

To start the development server in watch mode, run instead:

```
npm run start:dev
```

This will watch for file changes and restart the server accordingly.

Since the project uses bunyan as the logging library, to print the JSON logs in a human-readable manner, use the following command:

```
npm start | npx bunyan
```

This also works with `start:dev`, etc. For more information about bunyan's CLI, see the documentation for the package.

Tools

The project uses multiple tools to assure code quality and correctness. It is required to use these tools to ensure code quality before any commit!

- Prettier - Prettier is an opinionated code formatter with support for: JavaScript (including experimental features), TypeScript, JSON etc. It removes all original styling and ensures that all outputted code conforms to a consistent style.
  Usage: `npm run format`
- ESLint - ESLint is an open source JavaScript linting utility, which also supports TypeScript.
  Usage: `npm run lint`
- Jest - The testing framework used for the project.
  Usage:
  - `npm test` - Unit tests or specs.
  - `npm run test:e2e` - End-to-end tests.

26.6. הרצה ופיתוח מקומי של Email Service

The **email service** is responsible for sending emails to clients for related actions they took with Skyline CRS. This service consumes email messages from RabbitMQ of different emails to send and sends they corresponding email to the client according to the contact information of the client's booking.

Features

- Consumes email messages from RabbitMQ.
- Sends booking confirmation emails.
- Sends flight ticket emails.
- Sends booking cancellation confirmation emails.
- Sends boarding pass emails.

Usage

This is a Docker based Node.js project, therefore it should be run in a container.

Using Docker Compose

To run this service locally, along with its dependencies, Docker Compose may be used.

Navigate to the root Skyline CRS directory and run:

```
docker-compose up -d --build
```

This will run the entire Skyline CRS system locally using Docker Compose. You may modify `docker-compose.yml` to your liking, along with its respective .env file. Make sure to also create an `email.env.local` file as described in the Skyline CRS repository's README file.

Using Docker

To build the image directly, navigate to the project's directory and run:

```
docker build -t skyline-email .
```

And to run the image in a new container run:

```
docker run -dp 80:80 \
  -e SKYLINE_RABBITMQ_URI=<RabbitMQ URI> \
  -e SKYLINE_EMAIL_EXCHANGE_NAME=<Email exchange name> \
  -e SKYLINE_EMAIL_QUEUE_NAME=<Email queue name> \
  -e SKYLINE_EMAIL_BINDING_KEY=<Email binding key> \
  -e SKYLINE_DEAD_LETTER_EXCHANGE_NAME=<Dead letter exchange name> \
  -e SKYLINE_PNR_DB_URI=<PNR database URI> \
  -e SKYLINE_INVENTORY_MANAGER_URL=<Inventory manager URL> \
  -e SKYLINE_EMAIL_ADDRESS=<From email address> \
  -e SKYLINE_SMTP_HOST=<SMTP server host name> \
  -e SKYLINE_SMTP_PORT=<SMTP server host name> \
  -e SKYLINE_SMTP_USERNAME=<SMTP server username> \
  -e SKYLINE_SMTP_PASSWORD=<SMTP server password> \
  skyline-ticketing
```

Note that this service depends on two other services:

- RabbitMQ - Used by Skyline CRS to manage queues. It is used to consume emailing messages.
- PNR database - Used to read booking (PNR) details.
- Inventory manager - Used to retrieve flights and booked seats data.

Environment Variables

The service is configured using environment variables. Note that some environment variables are required, and the image will not run without them.

`SKYLINE_RABBITMQ_URI` (Required)

URI to the RabbitMQ service/cluster to use for consuming the email queue. The URI *should* contain the vhost to connect to.
Example values:

- `amqp://username:password@localhost:5672/skyline`
- `amqp://username:password@pnr:5672/skyline`

`SKYLINE_EMAIL_EXCHANGE_NAME` (Required)

The name of the RabbitMQ email exchange. This exchange should be a topic exchange.
Example value: `email`.

`SKYLINE_EMAIL_QUEUE_NAME` (Required)

The name of the RabbitMQ email queue. This queue should be a durable queue with a dead letter exchange and routing key, as given by SKYLINE_DEAD_LETTER_EXCHANGE_NAME and `SKYLINE_DEAD_LETTER_ROUTING_KEY`.
Example value: `email`.

`SKYLINE_EMAIL_BINDING_KEY` (Required)

The binding key of the RabbitMQ email exchange and the email queue.
Example value: `email.#`.

`SKYLINE_DEAD_LETTER_EXCHANGE_NAME` (Required)

The name of the RabbitMQ dead letter exchange for the email queue. This exchange should be a topic exchange.
Example value: `dlx`.

`SKYLINE_INVENTORY_MANAGER_URL` (Required)

Inventory manager URL to query the flights and booked seats information from.
Example values:

- `http://localhost:8080/v1/graphql`
- `http://inventory-manager/v1/graphql`

`SKYLINE_PNR_DB_URI` (Required)

The PNR database MongoDB URI for reading the bookings' details.
Example values:

- `mongodb://username:password@localhost:27017`
- `mongodb://username:password@pnr:27017`

`SKYLINE_EMAIL_ADDRESS` (Required)

The email address to send emails from.
Example value: `example@zohomail.com`.

`SKYLINE_SMTP_HOST` (Required)[1]

The SMTP server host name or IP address to use for sending emails.
Example value: `smtp.zoho.com`.

`SKYLINE_SMTP_PORT` (Required)[1]

The SMTP server's port to use for sending emails.
Example value: `465`.

`SKYLINE_SMTP_USERNAME` (Required)[1]

The username to use for authenticating with the given SMTP server. Example values:

- `example@zohomail.com`
- `username`

`SKYLINE_SMTP_PASSWORD` (Required)[1]

The password to use for authenticating with the given SMTP server using the given username. Example value: `password`.

`SKYLINE_PNR_DB_NAME`

The name of the PNR database for the given database URL.
Default: `pnr`.

`SKYLINE_PNR_DB_COLLECTION_NAME`

The name of the PNR database's PNR collection for the given database URL.
Default: `pnrs`.

`SKYLINE_PREFETCH_COUNT`

The RabbitMQ consumer prefetch count to use for this service, when consuming from the given email queue. For more information, see: How to Optimize the RabbitMQ Prefetch Count.
Default: `1`.

`SKYLINE_EMAIL_BOOKING_CONFIRMATION_MESSAGE_TYPE`
The message type for booking confirmation emails in the given email queue.
Default: `email.booking.confirmation`.

`SKYLINE_EMAIL_FLIGHT_TICKET_MESSAGE_TYPE`
The message type for flights ticket emails in the given email queue.
Default: `email.booking.ticket`.

`SKYLINE_EMAIL_BOOKING_CANCELLATION_MESSAGE_TYPE`
The message type for booking cancellation confirmation emails in the given email queue.
Default: `email.booking.cancel`.

`SKYLINE_EMAIL_BOARDING_PASS_MESSAGE_TYPE`
The message type for boarding pass emails in the given email queue.
Default: `email.boarding.ticket`.

`SKYLINE_DEAD_LETTER_ROUTING_KEY`
The routing key to use for dead-lettered messages of the email queue. **Default:** `email.all`

`SKYLINE_IATA_AIRLINE_CODE`
The Skyline IATA airline's code. Note that it must conform to the IATA airline designator standard.
Regex: `/^[A-Z0-9]{2,3}$/`
Default: `SK`

`SKYLINE_LOG_LEVEL`
Log level of the service. Available values are: `TRACE, DEBUG, INFO, WARNING, ERROR, FATAL`.
Default: `INFO`.

Notes:

1. These environment variables are generally required when running in a docker container (as the NODE_ENV environment variable is set to production in the service's Dockerfile). However, in local development, unless NODE_ENV is set to production, they are not required, as the emails are not sent and are instead generated, saved, and previewed locally.

Development

This project uses TypeScript and uses amqplib for consuming messages from the ticket queue. It also uses email-templates for sending email and Handlebars.js as the templating engine for the emails. Since the project is

Docker based, it is not required to install Node.js on the system, but it is **highly recommended** for local development.

Requirements

- Node.js version 16.14.x (LTS).
- An available instance or cluster of RabbitMQ.
- A running instance of the PNR database along with its dependencies.
- A running instance of the inventory manager along with its dependencies.

Installation

To install all dependencies, navigate to the project's directory and run:

```
npm install
```

If you would like to validate and not modify the contents of package-lock.json, run:

```
npm ci
```

Running Locally

In order to run the project locally, make sure to first set all required environment variables for the current shell session, as well as setting NODE_ENV to development (in order for email previews to open in the browser).

To start the development server locally using ts-node, make sure you are inside the project's directory and run:

```
npm start
```

To start the development server in watch mode, run instead:

```
npm run start:dev
```

This will watch for file changes and restart the server accordingly.

Since the project uses bunyan as the logging library, to print the JSON logs in a human-readable manner, use the following command:

```
npm start | npx bunyan
```

This also works with `start:dev`, etc. For more information about bunyan's CLI, see the documentation for the package.

Tools

The project uses multiple tools to assure code quality and correctness. It is required to use these tools to ensure code quality before any commit!

- Prettier - Prettier is an opinionated code formatter with support for: JavaScript (including experimental features), TypeScript, JSON etc. It removes all original styling and ensures that all outputted code conforms to a consistent style.
  Usage: `npm run format`
- ESLint - ESLint is an open source JavaScript linting utility, which also supports TypeScript.
  Usage: `npm run lint`
- Jest - The testing framework used for the project.
  Usage:
  - `npm test` - Unit tests or specs.
  - `npm run test:e2e` - End-to-end tests.

27. בדיקות והערכה

רוב ה-microservices במערכת מכילים בדיקות יחידה ובדיקות אינטגרציה. כמו כן, בדקתי גם את המערכת בצורה ידנית מספר פעמים על מנת לבדוק את נכונותה. למרות זאת, חסרות בדיקות על מנת שיהיה אחוז כיסוי גבוה מספיק לפרויקט, והדבר בעיקר מפאת חוסר בזמן. לכן, כחלק מתוספות עתידיות לפרויקט, יש להוסיף בדיקות נוספות למערכת, ובמיוחד כאלו שיצליחו לבדוק את כולה (בדיקות end-to-end).

28. ניתוח יעילות

28.1. ביצועי המערכת

לאור אופי המערכת, כפי שהוסבר כבר בסעיפים קודמים, אין מגבלה ממשית כל עוד ניתן להרחיב את המערכת אופקית (להרים קונטיינרים נוספים). מבחינת ביצועים, המערכת עושה שימוש בהרבה טכנולוגיות קיימות ומוכחות, ובמהלך הפיתוח ניסיתי כמה שפחות להמציא את הגלגל. עם זאת, עדיין יש שימוש בזמני ריצה לא מאוד יעילים (Node.js ו-Python) שיכולים להשפיע על הביצועים, אבל תודות להרחבה אופקית זאת לא בהכרח בעיה – אם כי מעלה את מחיר התפעול של המערכת בפועל (שכן היא צורכת יותר משאבים).

28.2. עלות המערכת

המערכת משתמשת בטכנולוגיות קוד פתוח עם רישיונות חינמיים, כך שמבחינת המוצרים עצמם אין בעיה של מחיר. עם זאת, מאחר שהמערכת בנויה לשימוש ארגוני ולריצה בענן מבוסס Kubernetes, שידועה במערכת כבדה שצורכת משאבים רבים, יש עלות לתפעול המערכת, בין אם על ענן מקומי של הארגון (on-premise) או על שירותי ענן חיצוניים (Amazon AWS, Google Cloud, Azure, וכדומה).

28.3. אמינות המערכת

המערכת נכתבה בצורה כמה שיותר אמינה, בין אם ברמת הארכיטקטורה ובין אם ברמת הקוד. מבחינת הארכיטקטורה, קיים הניסיון למנוע מנתונים להיות תלויים באוויר – תמיד יהיה ניסיון לשמור את הנתונים במקום כלשהו על מנת למנוע איבוד נתונים (ב-RabbitMQ, במסדי הנתונים). מבחינת הקוד, עשיתי בדיקות רבות ל-data races, וניסיתי למנוע אותם עד כמה שאפשר, על מנת שמידע תמיד יהיה עקבי.

28.4. שלמות המערכת

המערכת במצבה הנוכחי שלמה מבחינה תפעולית. יש דברים נוספים שניתן לפתח עבודה, עליהם אפרט בפרק 31, אך גם במצבה הנוכחי היא תפעולית לגמרי.

29. אבטחת מידע

המערכת משתמשת באימות באמצעות JWT tokens על מנת למנוע גישה לא מורשית את המידע במערכת. למידע נוסף, ראו סעיף 13.3.

30. מסקנות

הפרויקט Skyline CRS הוא פרויקט פיתוח התוכנה הגדול ביותר שעשיתי. הוא כלל התעסקות בהרבה היבטים שונים של פיתוח ואפילו DevOps, מהפיתוח של התוכנה ועד ה-deployment שלה.

במהלך הדרך למדתי דברים רבים:

- חיזקתי את הידע שלי ב-TypeScript ו-Python ובסביבות הפיתוח והכלים שלהן.
- למדתי לעומק יותר לגבי Docker ועל כיצד ניתן לעשות deployment ל-Kubernetes ו-OpenShift.
- למדתי כיצד לתכנן ולעצב מערכת מבוזרת מההתחלה ולתאר את הארכיטקטורה שלה והאופן שזורם בה המידע.
- למדתי לגבי היתרונות והחסרונות של טכנולוגיות מסוימות, ומתי כדאי להשתמש באיזה כלי ולאיזו עבודה.

אני חושב שהפרויקט תרם לי מאוד להבנה לגבי מה שעומד מאחורי מערכות כאלו, ועד כמה קשה לתחזק ולהפעיל מערכות גדולות הרבה יותר, ואף גרם לי להעריך יותר את העבודה שעומדת מאחוריהן.

הפרויקט גם לימד אותי במה אני אוהב להתעסק, אילו טכנולוגיות אני מעדיף, אילו היבטים של הפיתוח אני אוהב ולא אוהב, ועוד.

נהניתי מאוד מפיתוח הפרויקט, גם אם היו רגעים חסרי מוטיבציה, אך אהבתי את היבט הלמידה שליווה את הפרויקט והאופן בו התאפשר לי לבנות מערכת שכזו מהבסיס ומעלה.

לבסוף, הפרויקט גם לימד אותי כיצד לתכנן ולחלק את העבודה, להעריך זמנים טוב יותר, וגם לדעת איפה אפשר להתפשר לטובת חלקים אחרים חשובים יותר.

31. פיתוחים עתידיים

במהלך הפיתוח עלו מספר פיתוחים עתידיים שהייתי רוצה להוסיף, ויכול להיות שחלקם יתווספו כבר לאחר כתיבת ספר זה:

- אתר עבור חברת התעופה המדומה, שיאפשר לי להמחיש בצורה פחות טכנית כיצד פועלת המערכת (למעשה לשימוש על ידי לקוח רגיל).
- מימוש של אחסון מידע ישן בארכיון, במיוחד עבור ה-PNR database עבורו היה microservice בתכנון המקורי שהיה אחראי להעברת PNR-ים ישנים לארכיון. דבר זה יהיה אפשרי גם עבור ה-inventory. אחסון מידע ישן בארכיון יתרום רבות לביצועים של מסדי הנתונים, שכן תהיה כמות נתונים קטנה יותר בהם יצטרכו לחפש.
- הוספת בדיקות יחידה ואינטגרציה חסרות ל-microservices והוספת בדיקות end-to-end למערכת עצמה.

32. ביבליוגרפיה

1. ויקיפדיה: https://wikipedia.org
2. Stack Overflow: https://stackoverflow.com
3. Node.js: https://nodejs.org/en
4. TypeScript: https://www.typescriptlang.org
5. Express: https://expressjs.com
6. Python: https://www.python.org
7. FastAPI: https://fastapi.tiangolo.com
8. RabbitMQ: https://www.rabbitmq.com
9. PostgreSQL: https://www.postgresql.org
10. MongoDB: https://www.mongodb.com
11. Docker: https://docs.docker.com
12. Kubernetes: https://kubernetes.io
13. OpenShift: https://www.redhat.com/en/technologies/cloud-computing/openshift