האוניברסיטה הפתוחה

המחלקה למתמטיקה ולמדעי המחשב

סדנה בתכנות מונחה עצמים

מספר קורס 20586

SwiftTicket מסמך תכנון ועיצוב

העבודה הוכנה על-ידי: אנטון דכטיאר, ת"ז 321691909

תאריך ההגשה: 13/05/2024

תוכן עניינים

תיאור כללי של המערכת

מערכת SwiftTicket מיועדת לניהול וטיפול בקריאות טכניות ובקשות משתמשים במחלקת ה-IT של החברה. SwiftTicket היא תוכנה ששומרת על מעקב אחר בעיות הלקוחות כך שצוות התמיכה או ה-IT יכולים לפתור אותן במהירות. היא מספקת לצוות את הכלים היומיומיים הדרושים להם כדי לבצע את עבודתם ביעילות.

מערכת "SwiftTicket" בנויה לפי דפוס העיצוב MVC (Model-View-Controller) עם אלמנטים של MVVM (Model-View-ViewModel) שמעוד מתאים עבור "SwiftTicket". MVC, מודל 3-Tier ונכתבה באמצעות ASP.NET Core 8 .

MVC (Model-View-Controller) הוא דוגמה לארכיטקטורה בתוך התכנות המבוססת על אובייקטים, שבה נפרדים שלושה רכיבים עיקריים: שכבת הנתונים (Model), שכבת הלוגיקה העסקית (Business Logic Layer), ושכבת התצוגה (Presentation Layer).

1. שכבת בסיס הנתונים (Database Layer - Model) - בשכבה זו מתרחשת הגישה לבסיס הנתונים, והיא אחראית על אודות הנתונים והקשרים ביניהם. המודל מכיל את הבניית הנתונים, ביצוע שאילתות, וכל הלוגיקה הקשורה למניפולציה של המידע.
2. שכבת הלוגיקה העסקית (Business Logic Layer - Controller) - השכבה זו אחראית על הלוגיקה העסקית של האפליקציה. היא מקבלת קלט מהתצוגה, מעבירה אותו למודל לביצוע פעולות ומחזירה את התוצאות לתצוגה. Controller נותן את ההוראות למודל כיצד לעדכן את המידע ואילו נתונים להציג למשתמש.
3. שכבת התצוגה (Presentation Layer - View) - שכבת התצוגה היא אחראית על התצוגה והקשר עם המשתמש. התצוגה מציגה את המידע למשתמש בצורה מובנית וקלה להבנה. היא מתחשבת בקלט של המשתמש ומעבירה אותו ל-Controller לטיפול.

באמצעות הפרדה זו, אם יהיה צורך לשנות יחידת קוד, השפעה על שאר היחידות נמוכה ואין צורך בשינויים יתר במערכת. זה מאפשר נציגות וגמישות בפיתוח ותחזוקה של המערכת.

תיקיות הפרויקט מאורגנות באופן הבא לכל תיקיה יש תפקיד מסוים במבנה האפליקציה:

תיקיה ראשית מכילה את כל שאר התיקיות ואת מחלקה Program.cs עם קובץ קונפיגורציה appsettings.json. המחלקה Program.cs משמשת כנקודת הכניסה לאפליקציה. היא מכילה את הפונקציה Main, שהיא הפונקציה הראשונה שמופעלת כאשר האפליקציה מתחילה לרוץ. פונקציה זו בדרך כלל מגדירה את ה-WebApplication Builder, מגדירה ומוסיפה שירותים למיכל התלויות, קובעת את תצורת ה-HTTP request pipeline ובסופו של דבר מפעילה את האפליקציה.

קובץ הקונפיגורציה appsettings.json משמש לאחסון תצורה הניתנת לשינוי בלי לשנות את קוד המקור של האפליקציה. זה כולל פרטים כמו מחרוזות חיבור למסדי נתונים, פרמטרים לתצורת האפליקציה, ועוד. זה מאפשר לך לשנות הגדרות ללא צורך לקומפל ולהפצת האפליקציה מחדש. בפרויקט שלי appsettings.json כולל גם הגדרות עבור Mailgun ומחרוזת החיבור למסד הנתונים SQL Server .

Controllers - תיקיה זו מכילה את Controllers של האפליקציה. Controllers אלו משמשים כגשר בין מודלים לתצוגות, והם מטפלים בבקשות המשתמש, מבצעים לוגיקה של אפליקציה ובסופו של דבר מחזירים תגובה למשתמש.

Data - תיקיה זו מכילה את ה-DbContext של Entity Framework Core, אשר משמש כמנגנון לגישה ולניהול של בסיס הנתונים. היא יכולה גם לכלול מחלקות עזר נוספות לניהול מסד הנתונים.

Interfaces - תיקיה זו מכילה את ממשקים שמשמשים להגדרת חוזים לפונקציונליות שיכולה להיות מיושמת על ידי מחלקות אחרות. זה מסייע ביצירת קוד גמיש וניתן לבדיקות.

Migrations - תיקיה זו מכילה קבצי מיגרציה שנוצרים על ידי Entity Framework Core כדי לשמור על סנכרון בין מבנה מסד הנתונים למודלים שבקוד.

Models - תיקיה זו מכילה מחלקות שמגדירות את המודלים (העצמים) שהאפליקציה משתמשת בהם. זה כולל גם מחלקות עם Data Annotations שמגדירות איך המודלים נשמרים במסד הנתונים.

Services - תיקיה זו מכילה מחלקות שמספקות פונקציונליות לוגית לאפליקציה, כמו שליחת מיילים, חיבור ל-APIs חיצוניים ועוד.

Utilities - תיקיה זו מכילה כלים ומחלקות עזר שמשמשות למטרות כלליות ברחבי האפליקציה, כמו פונקציות עזר לתאריכים, מחרוזות ועוד.

ViewModels - תיקיה זו מכילה מחלקות שמשמשות להעברת נתונים בין ה-Controllers ל-Views. כל ViewModels מייצג צורת הנתונים הספציפית שה-View צריך להציג.

Views - תיקיה זו מכילה קבצי Razor (קבצי CSHTML) שמגדירים את המבנה והנראות של הדפים שמשתמש רואה. הקבצים אלו משתמשים ב-HTML ובדרך כלל משלבים גם קוד C# להצגת הנתונים מה-ViewModels.

מסד הנתונים של הפרויקט מתבסס על SQL Server שמאוחסן ב-Azure. הוא משמש לאחסון וניהול של כל הנתונים הרלוונטיים לאפליקציה, כולל פרטי המשתמשים, פרטי הכרטיסים, תגובות וכדומה. הבחירה ב-SQL Server מאפשרת ניהול יעיל של טרנזקציות, הרצת שאילתות מורכבות.

בפרויקט SwiftTicket, מסד הנתונים נבנה באמצעות Entity Framework Core, שהוא Object-Relational Mapping (ORM) framework שמקל על האינטגרציה בין הקוד לבסיס נתונים SQL.

הנה כיצד זה פועל בפרויקט:

- הגדרת מודלים - בתיקייה Models, יצרת מחלקות שמשקפות את הטבלאות במסד הנתונים. לדוגמה, יש לך מחלקה למשתמש (`User`), כרטיס (`Ticket`), תגובות (`Comment`), ועוד. כל מחלקה מגדירה תכונות שמתואמות לעמודות בטבלה במסד הנתונים.
- הגדרות Entity Framework - בתיקייה Data, יש את מחלקת ApplicationDbContext, שהיא נגזרת מ-DbContext. מחלקה זו משמשת כשכבת הנתונים של האפליקציה ומכילה את ההגדרות לכל המודלים שמשתמשים בהם. כאן מוגדרים הקשרים בין הטבלאות, כגון יחסי חד-לרבים, רבים-לרבים וכו".
- מיגרציות - באמצעות כלי המיגרציות של Entity Framework, ניתן להגדיר שינויים במבנה של מסד הנתונים באופן שמשתקף בקוד. כאשר נדרש להוסיף או לשנות מודלים, ניתן ליוצר

מיגרציה חדשה שמכילה את השינויים. פקודת Update-Database של EF Core מחילה את המיגרציה על מסד הנתונים, כך שהשינויים במודלים מתבצעים גם במסד הנתונים.

לדוגמה אם נדרש להוסיף עמודה חדשה לטבלה, ניתן להוסיף תכונה חדשה למודל המתאים, ליצור מיגרציה חדשה, ולאחר מכן לעדכן את מסד הנתונים. השימוש במיגרציות מאפשר לשמור על סנכרון בין הקוד למבנה מסד הנתונים ולהבטיח עקביות ובטיחות בעת עדכונים ושינויים.

הנחות עבודה בכתיבת פרויקט

הנחות עבודה בפרויקט מתייחסות להנחות שנעשו במהלך תכנון ופיתוח הפרויקט. הנה כמה מהן:

הנחות טכנולוגיות

- השימוש ב-#C ו-ASP.NET Core כטכנולוגיות בסיס לבניית האפליקציה מניח שיש לך גישה למערכת הפעלה שתומכת בפיתוח דוטנט. השימוש ב-SQL Server כמערכת מסד נתונים מניח שהפרויקט יפעל בסביבה שבה יש גישה ל-SQL Server או לשירותי ענן התומכים ב-SQL.

הנחות משתמש

- מניח שלמשתמשים יש ידע בסיסי בשימוש בממשקי ווב ובהתמצאות במערכות אינטראקטיביות. מניח שלמשתמשים יש גישה רציפה לאינטרנט כדי להבטיח גישה לאפליקציה בענן.

הנחות פיתוח

- מניח שהפיתוח יתבצע בסביבה עם גישה לכלים ומשאבים הדרושים, כולל עדכונים ותמיכה לטכנולוגיות השונות. מניח שיש גישה לתיעוד ולקהילות תמיכה של הטכנולוגיות המשמשות בפרויקט, למקרה של צורך בעזרה טכנית.

הנחות בטיחות

- מניח שמדיניות הבטיחות וההגנה במסד הנתונים ובאפליקציה מספקת מגן כנגד רוב ההתקפות הסיבר הידועות. מניח שהמשתמשים ישמרו על זהירות בסיסית בנושאי אבטחת מידע וסיסמאות.

הנחות תפעול

- מניח שהאפליקציה תתוחזק על ידי אדם או צוות עם ידע טכני מתאים לניהול ותחזוקה של יישומי ווב ומסדי נתונים.

הבנת ההנחות הללו מאפשרת להבין את המגבלות והדרישות הקיימות כאשר מתכננים ומפתחים אפליקציה, ולהתכונן בהתאם לפתרון בעיות ולתפעול יעיל של המערכת.

מוסכמות רישום

שמות מחלקות, מתודות ושדות - כל שם שמורכב מיותר ממילה אחת ישתמש ב-PascalCase כלומר כל מילה מתחילה באות גדולה.

מחלקות וממשקים - כל מחלקה מתחילה באות גדולה. ממשקים באות 'I'.

מחלקות וממשקים בדפוסי עיצוב - מחלקות וממשקים המשתתפים בדפוס עיצוב אך אינם מייצגים אובייקט קונקרטי ישתמשו בשם המשתתף בדפוס.

שדות פרטיים - שדה פרטי במחלקה יתחיל בתו '_'.

קבועים וערכי enum - ייכתבו באותיות גדולות בלבד.

פרמטרים ומשתנים מקומיים - יתחילו באות קטנה.

שיטות ציבוריות ופרטיות - שיטות ציבוריות יתחילו באות גדולה. שיטות פרטיות או מוגנות יתחילו באות קטנה.

פקדים בטופס - כל פקד יתחיל באות קטנה ולאחר מכן יבוא סוג הפקד עם מילים מופרדות על ידי '_'.

שכבת בסיס הנתונים

שכבת הנתונים בפרויקט מהווה את הקשר בין מערכת האפליקציה לבסיס הנתונים שבו המידע נשמר ונשלף. היא אחראית על הניהול והטיפול בכל מה שקשור לנתונים, כולל קריאה, כתיבה, עדכון ומחיקת נתונים מבסיס הנתונים.

תפקידים של שכבת הנתונים:

- הפרדה בין הלוגיקה של האפליקציה לבין ניהול הנתונים - זה מאפשר להפוך את הקוד ליותר נקי ומסודר, כאשר כל חלק מתמקד במשימות שונות.
- אבטחת הנתונים - דאגה לאבטחת נתונים באמצעות סנטרליזציה של ניהול הנתונים, מה שמקל על הגנה על פרטיות המשתמשים ומניעת גישה לא מורשת.
- ניהול עסקאות - טיפול בעסקאות מסד הנתונים באופן יעיל ובטוח, כולל טיפול בשגיאות ובכישלונות.

בסיס הנתונים:

המערכת משתמשת ב - SQL Server מערכת ניהול מסד נתונים רלציונית (RDBMS) שתומכת בשפת השאילתה SQL . זהו בסיס נתונים חזק ומוביל בתעשייה, המספק תמיכה רחבה ב-transactions, אבטחה, שחזור נתונים ויותר.

מחלקות מרכזיות בשכבת הנתונים:

1. DB Context - מחלקה שמשמשת כממשק לבסיס הנתונים. היא מאפשרת אינטראקציה עם ה - entity EF Core עם תמיכה ב-Identity Framework
2. Migration - מחלקות אשר משמשות לשליטה וניהול של שינויים ועדכונים במבנה בסיס הנתונים.
3. Models - מחלקות שמגדירות את המבנה של הטבלאות בבסיס הנתונים, כולל הגדרות שדות וקשרים בין הטבלאות.

דפוס עבודה:

שכבת הנתונים פועלת לפי דפוס ה - Repository Pattern דפוס זה מאפשר ארגון טוב יותר של קוד הנתונים, פשטות בבדיקות ומפחית תלות במקור הנתונים הקונקרטי.

תיאור שך שוטות הרלוונטיות והגדרות המודלים במחלקות:

ApplicationDbContext.cs בתיקיית Data

DbSets	**Tickets**: נתונים עבור כרטיסים במערכת. מייצג את טבלת הכרטיסים במסד הנתונים. **ServiceHistories**: רשומות היסטוריית שירות עבור כל כרטיס. **Comments**: הערות שנכתבות על כרטיסים. **Sites**, **Categories**, **UrgencyLevels**, **TicketStatuses**: טבלאות עזר שמכילות מידע על אתרים, קטגוריות, רמות דחיפות וסטטוסים של כרטיסים בהתאמה.
OnModelCreating	במתודה זו מתבצעת הגדרה של הקשרים בין טבלאות, הגדרות על מפתחות זרים, הגדרות על מחיקות קסקדיות, ואינדקסים. כמו כן, ניתן להגדיר כאן טבלאות התחלתיות עם נתונים קבועים (seed data). לדוגמה: הגדרת יחסים ומפתחות זרים בין כרטיסים למשתמשים, בין הערות לכרטיסים וכו'. יצירת נתוני התחלה עבור טבלאות כמו אתרים וסטטוסים.
הגדרות התחול למנהל ומשתמשים	כוללת יצירת משתמש מנהל כברירת מחדל והשמתו בתפקיד המתאים, כמו גם הגדרת תפקידים נוספים במערכת. זה מאפשר התחלה מהירה של המערכת עם משתמשי ברירת מחדל.
UseLoggerFactory	בפונקציית OnConfiguring, מוגדר LoggerFactory שמאפשר לראות את פקודות SQL שנשלחות לשרת הנתונים בזמן ריצה, דבר שמאוד שימושי לניפוי באגים ואופטימיזציה של שאילתות.
מחיקות קסקדיות	מוגדרת אסטרטגיה של מחיקה שלא תגרום למחיקות רקורסיביות שיכולות לגרום לבעיות עקביות במסד הנתונים. זה חשוב במיוחד במערכות גדולות עם יחסים מורכבים בין ישויות.

בכללי, ApplicationDbContext מהווה את הליבה של ניהול הנתונים באפליקציה, והיא מאפשרת גישה מאורגנת וניהול של כל הנתונים במערכת.

InitialMigration.cs
20240505110002_InitialMigration.cs בתיקיית Migrations

Up()	מתודה זו מכילה את השינויים או ההוספות שיש לבצע במבנה הבסיס נתונים. זה כולל יצירת טבלאות חדשות, עמודות, אינדקסים, וכדומה. לדוגמה, אם נדרש להוסיף טבלת `Tickets` לבסיס הנתונים , קוד היצירה של הטבלה ייכלל במתודה `Up()` של ה-migration המתאים.
Down()	זוהי המתודה שמשמשת לביטול השינויים שבוצעו במתודת `Up()`. היא חיונית לשימוש כאשר צריך להחזיר את מבנה הבסיס נתונים למצבו הקודם למיגרציה. לדוגמה, אם הוספנו טבלה במתודת `Up()`, מתודת `Down()` תכיל פקודה למחיקת הטבלה.

פרויקט שמשתמש ב Entity Framework Core, הקובץ-
ApplicationDbContextModelSnapshot.cs משמש כתיעוד של מבנה הבסיס נתונים בזמן מסוים. הוא מייצג את המצב הנוכחי של המודל בקוד , כולל כל הטבלאות, העמודות, המגבלות והיחסים שהוגדרו דרך המודלים.

Comment.cs תיאור מחלקה\טבלה בתיקיית Models

מחלקה זו משמשת לייצוג תגובות במערכת, עם מזהים לכרטיס ולמשתמש המתייחסים אליה, וכן תאריך היצירה והתוכן של התגובה עצמה. השדות Ticket ו-User הם מאפייני ניווט שמספקים גישה ישירה לאובייקטים המתאימים בזיכרון, מה שמאפשר גישה נוחה וישירה לנתונים המקושרים.

שדה	טיפוס	הסבר
CommentId	int	מזהה ייחודי
TicketId	int	מזהה של הכרטיס שאליו מתייחסת התגובה
UserId	string	מזהה של המשתמש שהוסיף את התגובה

Content	string	טקסט התגובה
CreatedAt	DateTime	תאריך ושעה בהם נוצרה התגובה
Ticket	Ticket	מאפיין ניווט לכרטיס אליו מתייחסת התגובה
User	User	מאפיין ניווט לכרטיס אליו מתייחסת התגובה

ErrorViewModel.cs תיאור מחלקה\טבלה בתיקיית Models

מחלקת ErrorViewModel משמשת לייצוג מידע על שגיאות באפליקציה. היא מכילה מזהה בקשה (RequestId), שיכול לעזור באיתור בעיות ושגיאות על ידי התייחסות לבקשה ספציפית שגרמה לשגיאה. התכונה ShowRequestId משמשת לקביעה האם להציג את מזהה הבקשה, בהתאם לכך אם הוא אינו ריק, מה ששימושי בתצוגה של מסכי שגיאה בממשק המשתמש.

שדה	טיפוס	הסבר
RequestId	string?	מזהה בקשה אופציונלי, יכול להיות null
ShowRequestId	bool	תכונה המחזירה אמת אם RequestId אינו ריק

ServiceHistory.cs תיאור מחלקה\טבלה בתיקיית Models

מחלקת ServiceHistory משמשת לייצוג היסטוריית שירות שנעשתה עבור כרטיסים במערכת. היא מתעדת פרטים כמו פעולות שנעשו, מי ביצע את הפעולה ומתי זה קרה. השדות Ticket ו-User הם מאפייני ניווט שקשרים את הרשומה לאובייקטים המתאימים בזיכרון ולגשת אליהם בקלות יותר מתוך הקוד.

שדה	טיפוס	הסבר

ServiceHistoryId	int	מזהה ייחודי לרשומת ההיסטוריה של קריאת השירות
TicketId	int	מזהה של הכרטיס שאליו מתייחסת הרשומה
Date	DateTime	תאריך האירוע של השירות
ActionTaken	string	תיאור של הפעולה שבוצעה או השינוי שנעשה
UserId	string	מזהה של המשתמש שביצע את הפעולה
Ticket	Ticket?	מאפיין ניווט אופציונלי לכרטיס שאליו מתייחסת הרשומה
User	User?	מאפיין ניווט אופציונלי למשתמש שביצע את הפעולה

Ticket.cs תיאור מחלקה\טבלה בתיקיית

מחלקת Ticket משמשת לייצוג כרטיסים במערכת, כאשר כל כרטיס מייצג דיווח על בעיה או פנייה שקיבלה הארגון. המחלקה מכילה מגוון נתונים אודות הכרטיס, כולל מידע על מי פתח את הכרטיס, מצבו, דחיפותו, וכן הלאה, ומציעה גם אפשרות לטיפול וניהול תגובות והיסטוריית שירות בקשר לכרטיס. Models

שדה	טיפוס	הסבר
TicketId	int	מזהה ייחודי לכרטיס
Title	string	כותרת הכרטיס
Description	string	תיאור מפורט של הבעיה בכרטיס
CreatedAt	DateTime	תאריך ושעה שבהם נוצר הכרטיס

ClosedAt	DateTime?	תאריך ושעה שבהם נסגר הכרטיס, ניתן להיות null אם פתוח
StatusId	int	מזהה מצב הכרטיס, ברירת מחדל היא "חדש"
TicketStatus	TicketStatus?	ניווט למצב הכרטיס
UserId	string	מזהה של המשתמש שיצר את הכרטיס
User	User?	ניווט למשתמש שיצר את הכרטיס
CurrentSite	int	מזהה אתר נוכחי של הכרטיס
Site	Site?	ניווט לאתר שבו נמצא הכרטיס
Category	string	קטגוריה של הכרטיס
SubCategory	string	תת-קטגוריה של הכרטיס
Urgency	int	מזהה דחיפות הכרטיס, ברירת מחדל היא 1
UrgencyLevel	UrgencyLevel?	ניווט לרמת הדחיפות של הכרטיס
MobileNumber	string	מספר טלפון ליצירת קשר
LabLocation	string	מיקום המעבדה שבה נדרש השירות
TechnicianId	string?	מזהה של הטכנאי שאמור לטפל בכרטיס
Technician	User?	ניווט לטכנאי שמטפל בכרטיס

Comments	ICollection<Comment>	אוסף של תגובות המשויכות לכרטיס
ServiceHistories	ICollection<ServiceHistories>	אוסף של רשומות היסטוריית שירות לכרטיס

TicketReference.cs תיאור מחלקה\טבלה בתיקיית Models

כל אחת מהמחלקות הללו מייצגת רכיב נתונים שמשויך לכרטיס שירות במערכת. כל מחלקה מכילה מזהה ייחודי ושם שמתאר את המחלקה:

- Site משמש לייצוג מיקומים פיזיים שבהם יכולה להתבצע השירות.
- Category משמשת לקביעת סוג הבקשה או הבעיה בכרטיס.
- UrgencyLevel משמשת להגדרת דחיפות הטיפול בבקשה או בכרטיס.

Site (מיקום שירות)

שדה	טיפוס	הסבר
Id	int	מזהה ייחודי למיקום השירות
Name	string	שם המיקום, עד 255 תווים

Category (קטגוריית בקשת שירות)

שדה	טיפוס	הסבר
Id	int	מזהה ייחודי לקטגוריה
Name	string	שם הקטגוריה, עד 255 תווים

UrgencyLevel (רמת דחיפות בקשת שירות)

שדה	טיפוס	הסבר

Id	int	מזהה ייחודי לרמת הדחיפות
Name	string	שם רמת הדחיפות, עד 255 תווים

TicketStatus.cs תיאור מחלקה\טבלה בתיקיית Models

מחלקת TicketStatus משמשת לייצוג מצבי הכרטיסים במערכת. זהו רכיב חשוב במעקב אחר תהליך הטיפול בכל כרטיס, מאפשר לזהות מצב הנוכחי של כל כרטיס. השדה Name מאפשר לתאר את המצב בצורה ברורה ומובנת, כגון "חדש", "בטיפול", "טופל", "נדחה" וכדומה.

שדה	טיפוס	הסבר
Id	int	מזהה ייחודי למצב הכרטיס
Name	string	שם מצב הכרטיס, עד 255 תווים

User.cs תיאור מחלקה\טבלה בתיקיית Models

מחלקת User מורחבת מהמחלקה IdentityUser של ASP.NET Core Identity, שמספקת תכונות בסיסיות לניהול משתמשים כולל אימות ואבטחה. בנוסף לתכונות הסטנדרטיות של מודל משתמש, הוספת שדות ניווט כדי לשייך כרטיסים, תגובות ורשומות היסטוריית שירות שהמשתמש נתן או שהיו קשורות אליו. זה מאפשר ניהול מרכזי ויעיל של כל הפעילויות שהמשתמש מבצע במערכת.

שדה	טיפוס	הסבר
Id	string	מזהה ייחודי למשתמש, מורשת מ-IdentityUser

UserName	string	שם משתמש, מורשת מ-IdentityUser
Email	string	כתובת דוא"ל של המשתמש, מורשת מ-IdentityUser
PhoneNumber	string	מספר טלפון של המשתמש, מורשת מ-IdentityUser
TicketsCreated	ICollection<Ticket>	אוסף של כרטיסים שנוצרו על ידי המשתמש
Comments	ICollection<Comment>	אוסף של תגובות שנעשו על ידי המשתמש
ServiceHistories	ICollection<ServiceHistories>	אוסף של רשומות היסטוריית שירות שמשויכות למשתמש

שכבת הלוגיקה

AccountController.cs המתודות העיקריות במחלקה המנהלת פונקציונליות הקשורה לניהול חשבונות:

שם	סוג	הסבר
Login ()	GET	מציגה טופס להתחברות למערכת. מקבלת אופציונלית מחרוזת returnUrl שאליה יופנה המשתמש לאחר התחברות מוצלחת.
Login ()	POST	מעבדת את נתוני הטופס מהמשתמש להתחברות. אם ההתחברות מוצלחת, מפנה לעמוד הראשי או ל-URL שסופק. במקרה של כשל, מציגה שגיאה בטופס.
Register ()	GET	מציגה טופס להרשמת משתמש חדש למערכת.
Register ()	POST	מעבדת את נתוני ההרשמה מהמשתמש ויוצרת חשבון חדש.

		במידה והרשמה מוצלחת, שולחת אימייל לאימות ומפנה לדף ההתחברות. במקרה של כשל, מציגה שגיאות בטופס.
Logout ()	POST	מבצעת התנתקות למשתמש מהמערכת ומפנה לדף הבית.
ConfirmEmail ()	GET	מעבדת את בקשת אימות הדוא"ל שנשלחה למשתמש לאחר ההרשמה. במידה והאימות מוצלח, מפנה לדף ההתחברות. במקרה של כשל, מציגה דף שגיאה.

AdminController.cs שיטות עיקריות במחלקת המנהלת את פונקציונליות הניהול האדמיניסטרטיבי באפליקציה

שם	סוג	הסבר
UsersList()	GET	מציגה רשימה של כל המשתמשים במערכת.
AddUser()	GET	מציגה טופס להוספת משתמש חדש, כולל רשימת
AddUser()	POST	מעבדת ומוסיפה משתמש חדש למערכת לפי הנתונים שהוזנו. אם התהליך מוצלח, מוסיף את המשתמש לתפקיד שנבחר ומפנה לרשימת המשתמשים. אם יש שגיאות, מציגה אותן בטופס.
DeleteUser()	POST	מאפשרת מחיקת משתמש מהמערכת לפי מזהה המשתמש. אם המחיקה מוצלחת, מפנה לרשימת המשתמשים; אחרת, מחזירה שגיאה.

EditUser()	GET	מציגה טופס לעריכת משתמש קיים, כולל פרטים ותפקידיו הנוכחיים.
EditUser()	POST	מעבדת עדכון פרטי המשתמש ו/או התפקיד שלו במערכת. אם יש שינוי בתפקיד, מעדכנת גם את התפקיד. אם יש בעיות, מציגה שגיאות בטופס.
Reports()	GET	מציגה טופס ליצירת דוחות על פעילות במערכת, לדוגמה ספירת טיקטים שנסגרו בתקופה מסוימת
Reports()	POST	מעבדת את נתוני הדוח על פי תאריכים שהוזנו ומציגה את התוצאות

מחלקת AdminController.cs מוגבלת לשימוש על ידי משתמשים בעלי תפקיד "Admin" על מנת להבטיח שליטה ובקרה על ניהול המערכת, כולל הוספה, מחיקה ועריכת משתמשים, וכן יצירת דוחות על פעילות במערכת.

HomeController.cs המתודות במחלקת המנהלת פונקציונליות בסיסית בדף הבית ודפים קשורים:

שם	סוג	הסבר
Index ()	GET	מציגה את דף הבית של האפליקציה. פונקציה זו מעבירה לטופס מידע האם המשתמש מחובר או לא.
Privacy ()	GET	מציגה דף עם מדיניות הפרטיות של האפליקציה
Error ()	GET	מציגה דף שגיאה כאשר קורה תקלה במערכת. דף זה מציג מזהה בקשה שיכול לעזור בניתוח תקלות ובפתרון בעיות.

מחלקה HomeController נועדה להציג דפים סטטיים או פחות דינמיים כגון דף הבית ומדיניות פרטיות, ולטפל בהצגת דפי שגיאות במערכת. היא מספקת ממשק פשוט ויעיל לניהול תכנים בסיסיים שאינם דורשים אינטראקציה מורכבת מצד המשתמש.

RoleController.cs המתודות העיקריות במחלקת המנהלת פונקציונליות הקשורה לניהול תפקידים באפליקציה.

שם	סוג	הסבר
Index ()	GET	מציגה רשימה של כל התפקידים במערכת.
Create ()	GET	מציגה טופס ליצירת תפקיד חדש.
Create ()	POST	מעבדת ויוצרת תפקיד חדש במערכת לפי הנתונים שהוזנו. אם התהליך מוצלח, מפנה בחזרה לרשימת תפקידים. במקרה של שגיאות, מציגה אותן בטופס.
Edit ()	GET	מציגה טופס לעריכת תפקיד קיים, מציגה את פרטי התפקיד.
Edit ()	POST	מעבדת ומעדכנת תפקיד קיים לפי השינויים שבוצעו. במקרה של הצלחה, מפנה בחזרה לרשימת תפקידים. במקרה של שגיאות, מציגה אותן בטופס.
Delete ()	GET	מציגה טופס למחיקת תפקיד, כולל אזהרה או אישור לפעולה.
DeleteConfirmed ()	POST	מבצעת מחיקה של תפקיד לאחר אישור. אם המחיקה מוצלחת, מפנה בחזרה לרשימת תפקידים. אם יש שגיאות, מציגה אותן.

ManageUserRoles ()	GET	מציגה טופס לניהול התפקידים של משתמשים במערכת, כולל אפשרות לשייך משתמשים לתפקידים שונים.
UpdateUserRoles ()	POST	מעדכנת את תפקידי המשתמשים במערכת לפי הבחירות שנעשו, כולל הוספה והסרה של משתמשים מתפקידים.

מחלקv RoleController.cs מוגבלת לשימוש על ידי מנהלים בלבד על מנת לשמור על בקרה על ניהול תפקידים באפליקציה, והיא מספקת כלים ליצירה, עריכה, מחיקה וניהול של תפקידים והשיוך של משתמשים לתפקידים אלו.

TicketController.cs המתודות העיקריות במחלקה המנהלת פונקציונליות הקשורה לניהול טיקטים:

שם	סוג	הסבר
CreateRequestAsync ()	GET	מציגה טופס ליצירת טיקט חדש, כולל רשימות דינמיות של אתרים, קטגוריות ודחיפויות זמינות.
CreateRequest ()	POST	מעבדת ויוצרת טיקט חדש במערכת לפי הנתונים שהוזנו. אם המשתמש לא מחובר, מציגה שגיאה. אם התהליך מוצלח, מפנה לרשימת הטיקטים של המשתמש.
MyTickets ()	GET	מציגה רשימה של כל הטיקטים של המשתמש המחובר. אם המשתמש לא מחובר, מפנה לדף ההתחברות.
Edit ()	GET/POST	מציגה טופס לעריכת טיקט קיים ומעבדת שינויים שנעשו לטיקט. אם השינוי מוצלח, מפנה בחזרה לרשימת הטיקטים של המשתמש.
CloseTicket ()	POST	מעבדת בקשה לסגירת טיקט. אם הטיקט נסגר בהצלחה, מפנה בחזרה לרשימת הטיקטים של המשתמש. אם לא, מציגה שגיאה.

ShowSearchForm ()	GET	מציגה טופס חיפוש לטיקטים.
PerformSearch ()	GET	מבצעת חיפוש טיקטים לפי מונח חיפוש שסופק ומציגה את התוצאות.
Details ()	GET	מציגה פרטים מלאים על טיקט ספציפי, כולל הערות, מצב נוכחי, טכנאי מוקצה ועוד.
AssignTechnician ()	POST	מקצה טכנאי לטיקט. אם הפעולה מוצלחת, מפנה בחזרה לפרטי הטיקט.
AssignToMe ()	POST	מאפשרת לטכנאי להקצות טיקט לעצמו. אם ההקצאה מוצלחת, מפנה לדף הפרטים של הטיקט.
UpdateStatus ()	POST	מעדכנת את מצב הטיקט. אם העדכון מוצלח, מפנה בחזרה לדף הטכנאים.
ClosedTickets ()	GET	מציגה רשימת כל הטיקטים הסגורים, לפי משתמש או לכל המשתמשים אם המשתמש הוא מנהל או טכנאי.
TechnicianDashboard ()	GET/POST	מציגה ומעדכנת לוח מחוונים לטכנאים עם טיקטים מסוננים לפי מספר פרמטרים. מפנה בחזרה ללוח המחוונים עם הנתונים המעודכנים.
AddComment ()	POST	מאפשרת הוספת הערה לטיקט ושליחת אימייל למשתמש במידה וההערה נוספה בהצלחה. אם התהליך מוצלח, מפנה בחזרה לפרטי הטיקט.

מחלקה TicketController מספקת ממשק מורכב יותר שמאפשר ניהול ותפעול של טיקטים במערכת, כולל יצירת טיקטים, חיפוש, עריכה, סגירה, ומתן גישה לפרטים מפורטים על כל טיקט.

שכבת התצוגה

שכבת התצוגה בפרויקט מורכבת משני רכיבים עיקריים: תצוגות (Views) ומודלי תצוגה (ViewModels). כל אחד מהם משחק תפקיד חשוב כיצד המידע מוצג למשתמש וכיצד קלט מהמשתמש מועבר לשכבות אחרות באפליקציה.

Views

תצוגות ב-SwiftTicketApp מיושמות בעזרת קבצי Razor (קבצים עם סיומת .cshtml). Razor הוא סינטקס שמאפשר שילוב של HTML עם קוד C# באופן חלק ויעיל, מה שמאפשר לכתוב קוד דינמי שמשולב בקלות עם העיצוב. במערכת ישנם קבצי .cshtml לכל אחד מהדפים הבאים:

- Account - טיפול בכל הקשור לחשבונות משתמשים, כולל התחברות, הרשמה, ואימות חשבון.
- Admin - ממשק ניהול למנהלים לטיפול במשתמשים ובתפקידים.
- Home - דף הבית של האפליקציה, המציג מידע בסיסי וקישורים לפעולות נפוצות.
- Roles - ניהול תפקידים במערכת, כולל יצירה, עריכה ומחיקה של תפקידים.
- Tickets - ניהול טיקטים, כולל צפייה בטיקטים, עריכה, וטיפול בפניות.

ViewModels

ViewModels הם קלאסים שמגשרים בין ה-Controllers ל-Views. הם מותאמים לאחסון המידע הדרוש ספציפית לכל תצוגה. ViewModel יכול להכיל נתונים שהם תוצאה של מספר מודלים, והוא מספק את המידע בצורה שהתצוגה יכולה להשתמש בו בצורה יעילה ונוחה.

לדוגמה - ViewModel של טיקטים עשוי לכלול:

- רשימת כל הטיקטים של המשתמש.
- רשימת סטטוס ים לטיקטים.
- רשימת תגובות של כל טיקט.
- פרטים נוספים שמועברים לטופס לעריכת טיקט או ליצירת טיקט חדש.

איך הם עובדים יחד?

1. זרימת נתונים מה-Controller ל-View: ה-Controller טוען או מעבד נתונים מהמודלים (דרך שכבת השירות או ישירות) וממפה אותם ל-ViewModel. לאחר מכן ה-ViewModel מועבר ל-View דרך המתודה שב-Controller.
2. הצגת נתונים ב-View: ה-View משתמשת בנתונים שהגיעו ב-ViewModel להצגת תוכן דינמי בדף האינטרנט. השילוב של Razor מאפשר לפתח לשלב קוד C# בתוך דפי HTML, כך שהנתונים מה-ViewModel יכולים להיות מוצגים ולהשפיע על המבנה או הלוגיקה של הדף.
3. שליחת נתונים מה-View ל-Controller: כאשר משתמש מבצע פעולה שדורשת עיבוד נתונים, כמו שליחת טופס, הנתונים מה-View נשלחים חזרה ל-Controller. ה-Controller יכול לעבד את הנתונים, לעדכן מודלים במערכת או לבצע שאילתות, ואז להחזיר תגובה או לנתב את המשתמש לדף אחר.

4. מעגל חיים של דף אינטרנט: מתחיל בבקשה מהמשתמש, עובר ל-Controller שטוען נתונים ומעביר אותם ל-ViewModel, ה-ViewModel מועבר ל-View שמציגה את הנתונים, ותגובות מהמשתמש נשלחות חזרה ל-Controller לעיבוד נוסף.

המבנה הזה מאפשר גמישות גבוהה בניהול תכנים ולוגיקת משתמש תוך שמירה על קוד נקי ומודולרי, שכן כל שכבה עוסקת בתפקידים שהיא מומחית להם והנתונים מועברים בין השכבות בצורה מובנית וברורה.

ממשקים ומחלקות האבסטרקטיות

בפרויקט SwiftTicketApp ממשקים ומחלקות אבסטרקטיות משמשים להפרדה בין הלוגיקה של האפליקציה לבין פרטי המימוש, ולהקלת האינטגרציה והבדיקות של רכיבים בודדים במערכת.

ITicketService- ממשק זה מגדיר את כלל הפעולות הנדרשות לניהול טיקטים באפליקציה, והוא מאפשר למחלקת TicketService לממש את הפונקציונליות הנדרשת תוך כדי שמירה על גמישות בשינוי הלוגיקה או המסד נתונים מאחורי הקלעים ללא השפעה על שאר הקוד המשתמש בממשק זה.

TicketService - מחלקה זו מממשת את ממשק ITicketService ומספקת את הלוגיקה לביצוע הפעולות שמוגדרות בממשק. היא כוללת פעולות כמו יצירת טיקט, עדכון טיקט, חיפוש טיקטים, ועוד. מחלקה זו משתמשת במסד הנתונים (דרך מחלקת ApplicationDbContext) ובמנהל המשתמשים (UserManager<User>) לניהול המידע הנדרש.

MailgunEmailService - מחלקה זו מספקת שירותים לשליחת דואר אלקטרוני באמצעות שירות Mailgun. היא משתמשת ב-IConfiguration לקבלת הגדרות התצורה וב-IHttpClientFactory ליצירת לקוח HTTP שנועד לשלוח בקשות HTTP. מחלקה זו אינה מממשת ממשק מפני שהיא ספציפית לשירות של Mailgun ואינה דורשת אפשרות להחלפה דינמית של ההטמעה.

דוגמא לשימוש - כאשר מתבצעת פעולה שדורשת יצירת טיקט במערכת, ה-Controller הרלוונטי (לדוגמה, TicketController) יקרא למתודה CreateTicketAsync ממחלקת TicketService. מחלקת השירות תבצע את כל הפעולות הנדרשות לבדיקת תקינות, יצירה במסד נתונים, וכו'. לאחר הפעולה, תוחזר תגובה (ServiceResponse) שמציינת אם הפעולה הצליחה ומה הייתה השגיאה אם נכשלה.

השימוש בממשקים ובמחלקות אבסטרקטיות מקל על בדיקת הקוד ועל התחזוקה שלו, מאחר והם מאפשרים להחליף רכיבים ולבדוק רכיבים בנפרד ללא תלות בקוד הספציפי שמממש את הפונקציונליות.

דפוסי עיצוב

MVC (Model-View-Controller) - הפרויקט נבנה על פי הדפוס MVC, שהוא אחד מדפוסי העיצוב הנפוצים ביותר לפיתוח אפליקציות אינטרנט. דפוס זה מפריד בין הלוגיקה של היישום (Model), ממשק המשתמש (View) והבקרה (Controller)

- Model - מייצג את הדאטה והלוגיקה של האפליקציה.
- View - מציג את הממשק למשתמש.
- Controller - מתווך בין ה-View וה-Model, מעבד קלטים ומחזיר תגובות.

Repository - דפוס זה משמש להפרדה בין לוגיקת העסקים לבין גישה לנתונים. בפרויקט קיימ שימוש בממשק כמו ITicketService שמפריד בין האופן שבו הנתונים נשמרים ונטענים מהמסד לבין הלוגיקה שמשתמשת בנתונים אלו.

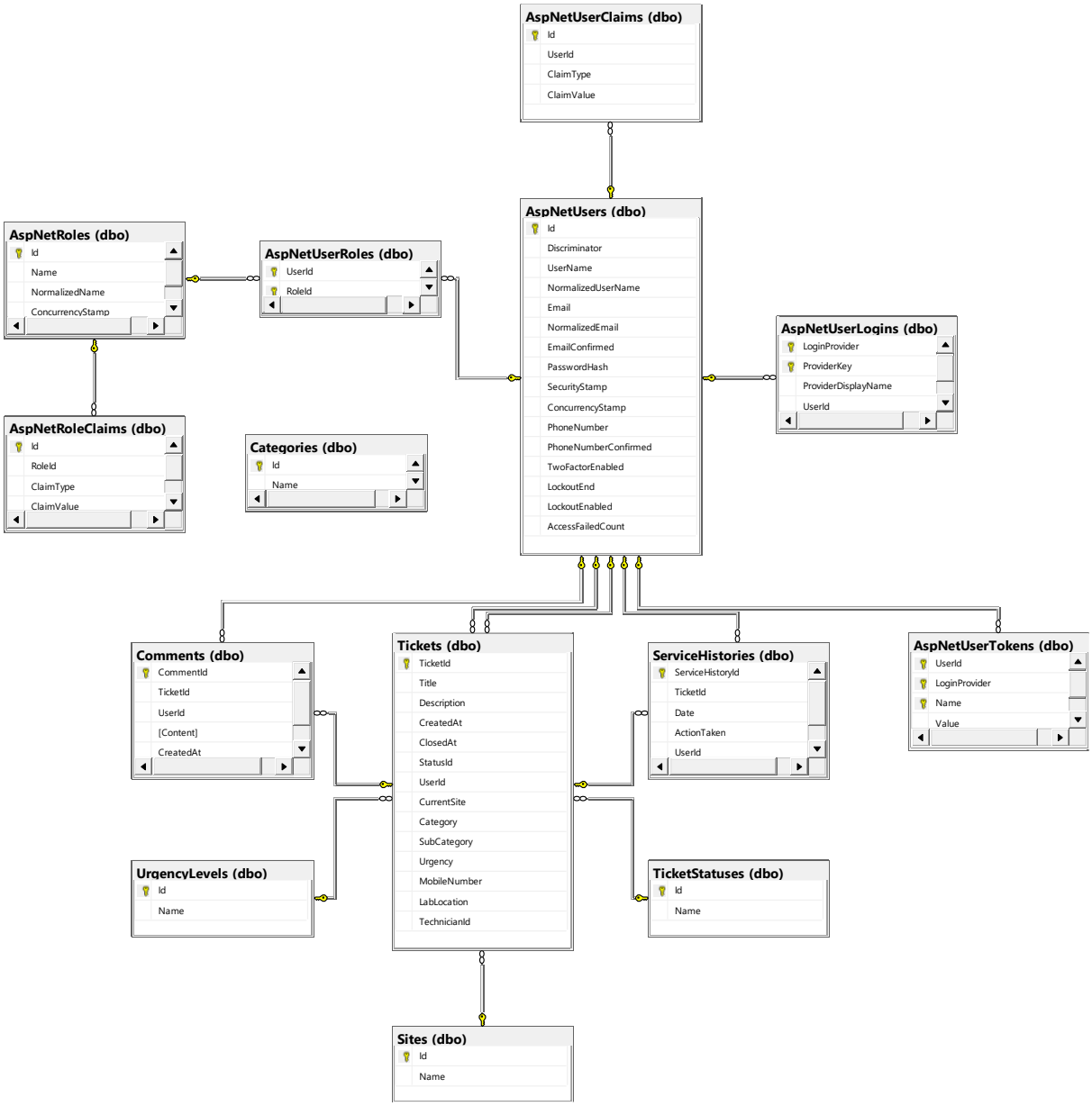
Dependency Injection (DI) - הוא דפוס שמאפשר הזרקת תלויות (כמו מסד נתונים או שירותים אחרים) לקומפוננטה שזקוקה להם במקום שהקומפוננטה תייצר אותם בעצמה. זה מאפשר גמישות רבה יותר וקוד נקי יותר. בפרויקט שימוש ב-DI נראה בקריאה לשירותים כמו MailgunEmailService וTicketService דרך הבנאי של הקונטרולרים.

Singleton - הוא דפוס עיצוב שבו קיימת אינסטנס יחיד של מחלקה לכל האפליקציה. בפרויקטים מבוססי .NET, מסגרות כמו ASP.NET Core טופלת DI באופן שכולל תמיכה ב-Singleton כאשר רכיב כגון IConfiguration וIHttpClientFactory יכולים להינתן כ-Singleton דרך ה- DI Container.

Factory - משמש ליצירת אובייקטים ללא הצורך לציין את המחלקה הספציפית בקוד שיוצר את האובייקט. זה מאפשר גמישות ביצירת אובייקטים שהמימוש שלהם יכול להשתנות על פי הגדרות או תנאי סביבה. לדוגמה: בשירות דוא"ל ניתן לשנות בין ספקים שונים על ידי שינוי המחלקה המממשת את ה-Factory.

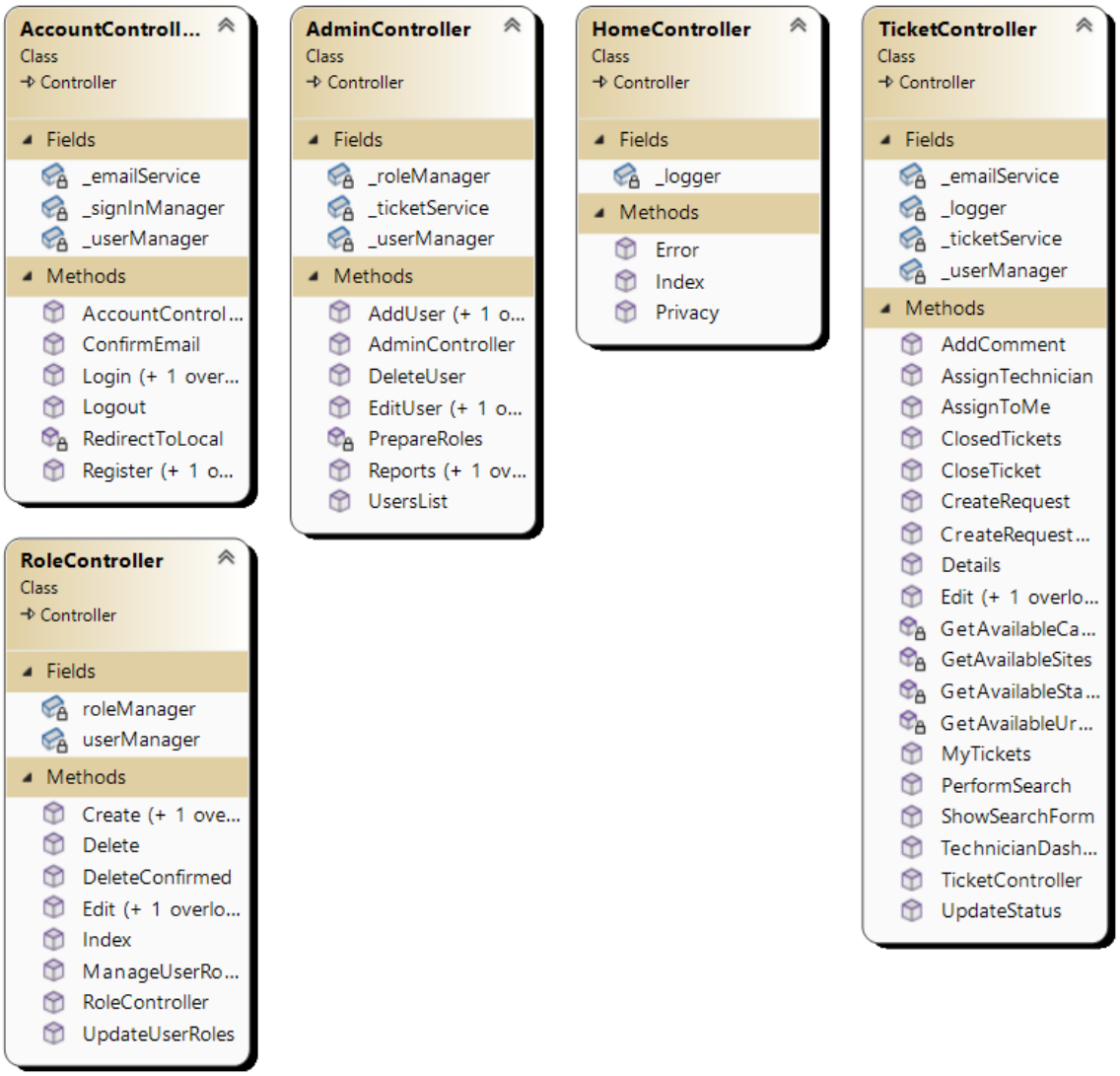
נספח: דיאגראמות

דיאגראמת מסד נתונים (Data Base Diagram)

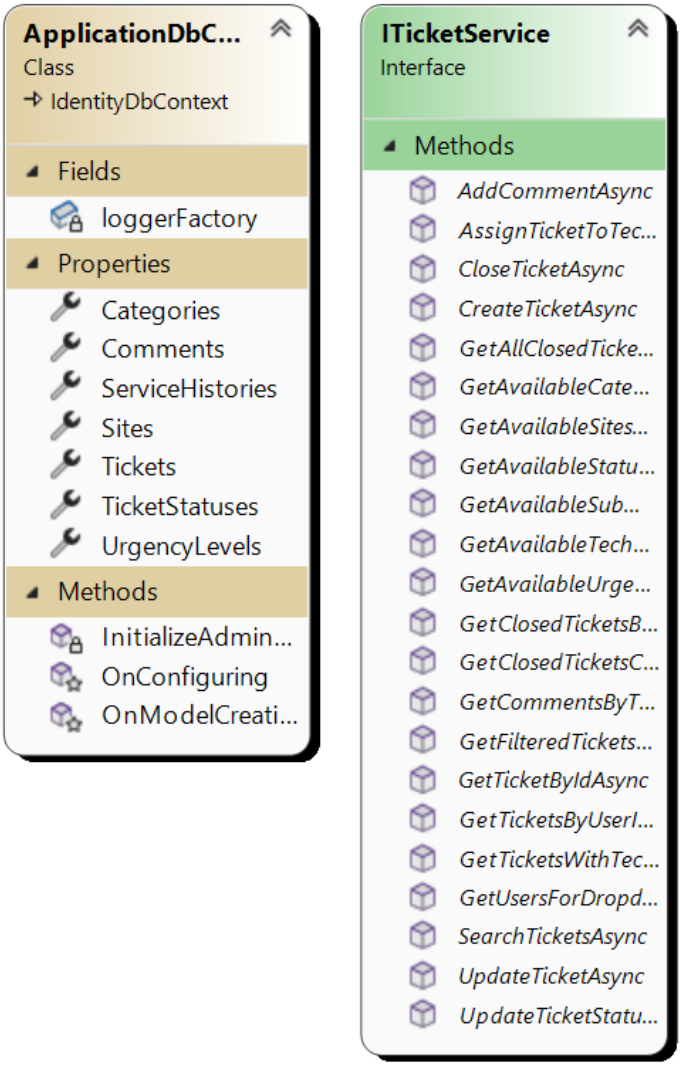


דיאגראמת מחלקות (Class Diagram)

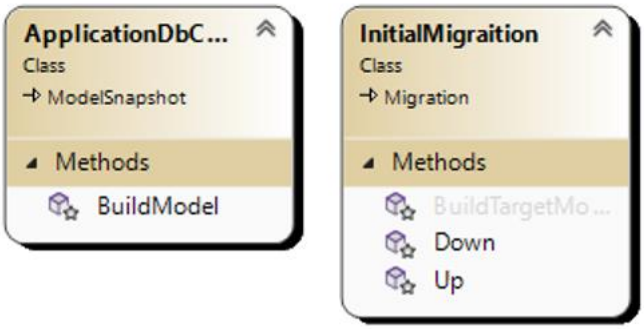
Controllers

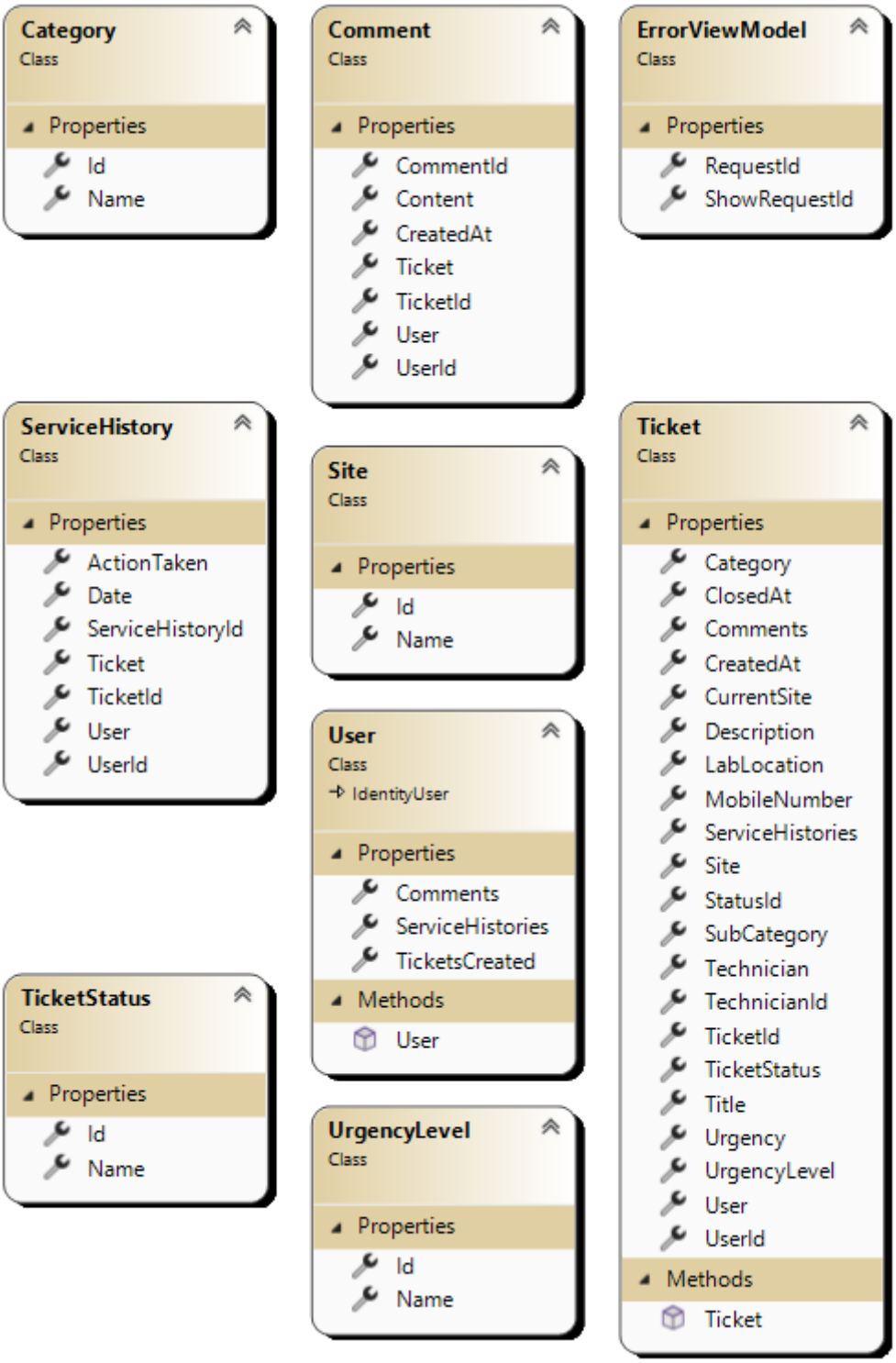


Data and Interfaces

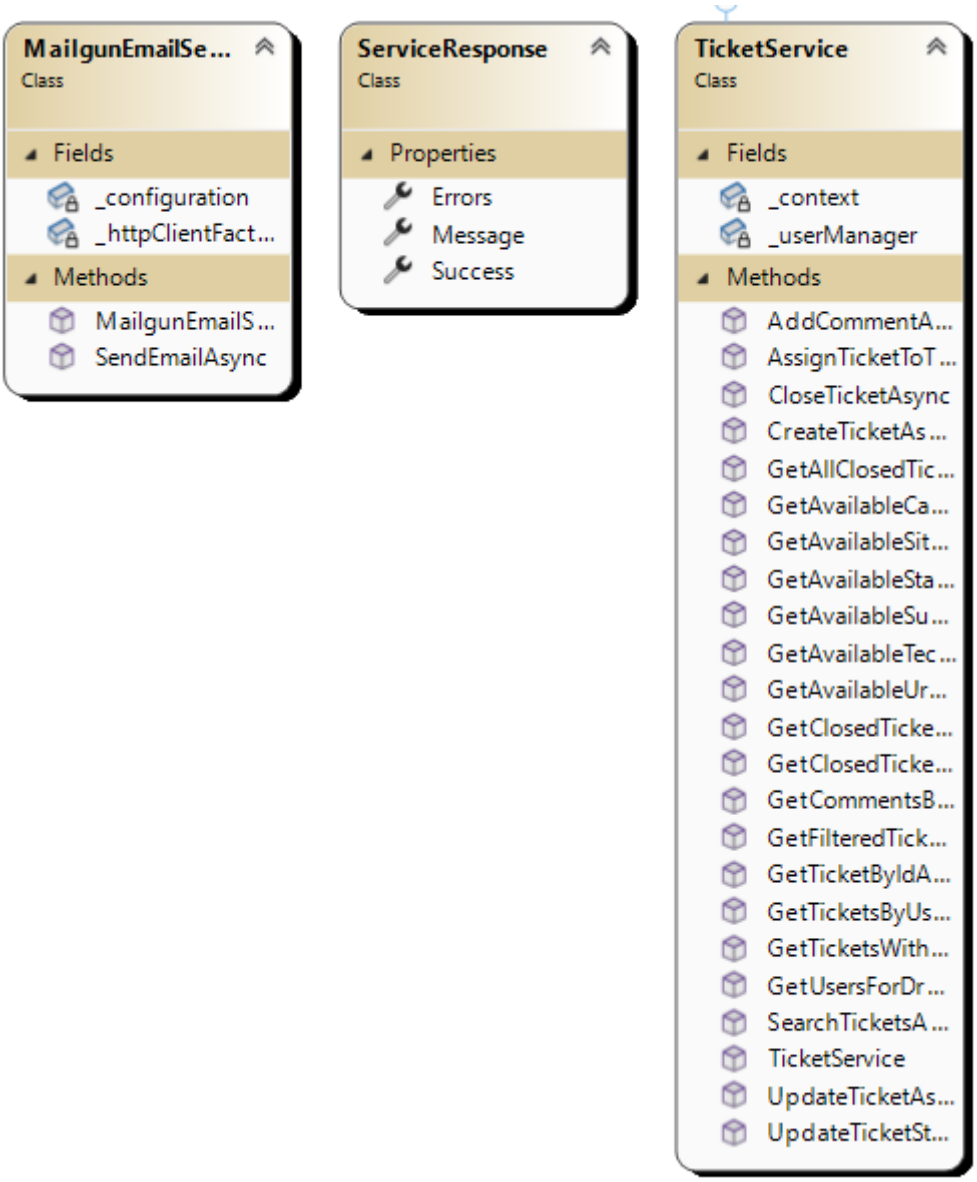


Migrations

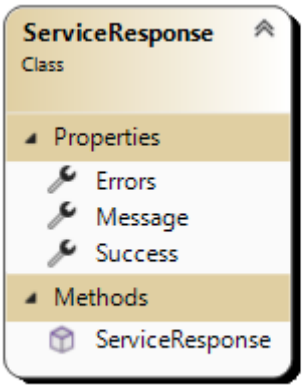



Category
Class
Properties
Id
Name
Comment
Class
Properties
CommentId
Content
CreatedAt
Ticket
TicketId
User
UserId
ErrorViewModel
Class
Properties
RequestId
ShowRequestId
ServiceHistory
Class
Properties
ActionTaken
Date
ServiceHistoryId
Ticket
TicketId
User
UserId
Site
Class
Properties
Id
Name
Ticket
Class
Properties
Category
ClosedAt
Comments
CreatedAt
CurrentSite
Description
LabLocation
MobileNumber
ServiceHistories
Site
StatusId
SubCategory
Technician
TechnicianId
TicketId
TicketStatus
Title
Urgency
UrgencyLevel
User
UserId
Methods
Ticket
User
Class
IdentityUser
Properties
Comments
ServiceHistories
TicketsCreated
Methods
User
TicketStatus
Class
Properties
Id
Name
UrgencyLevel
Class
Properties
Id
Name

Services



Utilities




LoginViewModel
Class
Properties
Email
Password
RememberMe
RegisterViewMo...
Class
Properties
ConfirmPasswo...
Email
Password
HomeViewModel
Class
Properties
IsUserAuthentic...
RoleViewModel
Class
Properties
Id
RoleName
ServiceCatalogV...
Class
AddUserViewM...
Class
Properties
Email
Password
Roles
SelectedRole
EditUserViewM...
Class
Properties
CurrentRole
Email
Id
NewPassword
Roles
SelectedRole
UserName
ReportViewModel
Class
Properties
ClosedTicketsC...
EndDate
StartDate
UserRoleViewM...
Class
Properties
AvailableRoles
CurrentRoleId
CurrentRoleNa...
UserId
UserName
EditRoleViewM...
Class
Properties
Id
RoleName
ClosedTicketsVi...
Class
Properties
ClosedTickets
SelectedUserId
Users
CommentView...
Class
Properties
Content
CreatedAt
User
UserName
CreateTicketVie...
Class
Fields
attachments
Properties
Attachments
AvailableCateg...
AvailableSites
AvailableUrgen...
Category
CC
CurrentSite
Description
LabLocation
MobileNumber
SubCategory
Urgency
TicketSearchVie...
Class
Properties
SearchResults
SearchTerm
EditTicketView...
Class
Properties
Category
CurrentSite
Description
LabLocation
MobileNumber
Statuses
StatusId
SubCategory
TicketId
Urgency
TechnicianDash...
Class
Properties
SelectedSite
SelectedStatus
SelectedSubCat...
SelectedSubmit...
SelectedTechni...
SelectedUrgenc...
Sites
Statuses
SubCategories
Submitters
Technicians
Tickets
UrgencyLevels
TicketDetailsVie...
Class
Properties
AssignedTechni...
AvailableStatuses
AvailableTechni...
Comments
CreatedAt
CurrentStatus
Description
SelectedTechni...
Status
TicketId

דיאגראמת פעילות (Activity Diagrams)

התהליך שבחרתי להדגים הוא מרכזי וקריטי לפעילות המערכת. לדוגמה:

- תהליך פתיחת בקשה הוא הלב והנשמה של מערכת לניהול קריאות כיוון שהוא מהווה את נקודת הפתיחה לכל תהליך השירות והתמיכה.
- ייצוגיות - תהליך שנבחר מייצג מגוון רחב של תכונות ויכולות של המערכת ובכך סופק מבט כולל על האופן בו פועלת המערכת.
- אופי המערכת והמשתמשים - בחירה זו משקפת את צרכי המשתמשים העיקריים של המערכת, כאשר רוב הפעולות במערכת נעשות דרך תהליך זה או תהליכים דומים לו.
- קלות ההבנה וההפקה - הבחירה בתהליך זה גם מנקודת מבט של קלות ההבנה וההפקה של הדיאגרמה. תהליך שהוא ברור וממוקד יאפשר להבין את המערכת בצורה טובה יותר.