טכנולוגיות אינטרנט מתקדמות (**WEB**) - 61776

- הגשת פרויקט -

**FurEver Friends - Pet Clinic | B8 | Group 4**

|  |  |
| --- | --- |
| **שם חבר/ת הצוות** | **ת"ז** |
| עומר לב | 209403427 |
| מתן ראובן טל | 208017772 |
| ספיר גרסטמן | 323116186 |
| אריאל לניאדו | 318930393 |
| נועה שיטרית | 206943219 |
| טל יגודין | 324256536 |

**תקציר הפרויקט**

הפרויקט עוסק בפיתוח אתר אינטרנט מודרני למרפאת החיות **FurEver Friends**, המציע חוויית שימוש מתקדמת ונגישה לבעלי חיות מחמד ולצוות הקליניקה. האתר מיועד לשני סוגי משתמשים עיקריים: **לקוחות (בעלי חיות) וצוות מזכירות המרפאה**, וכל אחד מהם נהנה ממערכת פונקציונלית מותאמת.

עבור הלקוחות, האתר מאפשר קביעת פגישות עבור חיותיהם לפי סוג טיפול, זמינות תורים ולפי הווטרינר המועדף. בנוסף, ניתן לערוך או לבטל פגישות קיימות, לעדכן פרטי משתמש, לנהל פרופיל עבור כל חיה כולל תמונה, לצפות במרשמים שניתנו ולסקור את היסטוריית הטיפולים. קיימת גם אפשרות לקבוע תור חירום בעת הצורך.

עבור צוות המזכירות, המערכת כוללת דשבורד עשיר המציג נתונים כגון מספר לקוחות, רווחים, לו"ז פגישות יומי ושבועי וטיפולים נפוצים. המזכירה יכולה לנהל את התורים (כולל תורים דחופים), לעדכן סטטוס טיפולים, להוסיף הערות וטרינר, לייצא פגישות לאקסל, לנהל לקוחות וחיותיהם וכן לנהל אנשי צוות וזמינותם. בנוסף, ניתן להנפיק ולנהל מרשמים לפי הנחיות הרופאים.

האתר מעוצב בקפידה, רספונסיבי ומותאם לכל סוגי המכשירים, עם תמיכה במצב כהה (Dark Mode) ונגישות גבוהה. כחלק מהחדשנות, משולב בו צ'אטבוט אינטראקטיבי המאפשר למשתמשים לבצע מגוון פעולות ולקבל מענה לשאלות נפוצות בנוגע למרפאה.

**קישור לעמוד GitHub:**

<https://github.com/Method-for-Software-System-Development/Advanced_Web>

**קישור לאתר (Vercel):**

<https://furever-friend-client.vercel.app/>

**קישור לסרטון השיווקי (YouTube):**

<https://www.youtube.com/watch?v=pYU9rKEr7R0>

**טכנולוגיות בפרויקט**

**Frontend (Client)**

**Core Framework & Libraries:**

* **React 19.0.0** – Main UI library
* **TypeScript 5.7.2** – Primary programming language
* **Vite 6.3.1** – Build tool and dev server
* **React Router DOM 7.6.0** – Routing and navigation
* **React DOM 19.0.0** – Rendering React to the DOM

**UI Components & Styling:**

* **Tailwind CSS 4.1.4** – Primary CSS framework
* **Lucide React 0.509.0** – Icon library
* **React Calendar 5.1.0** – Calendar component

**HTTP Client:**

* **Axios 1.9.0** – For making HTTP requests

**Build & Development Tools:**

* **@vitejs/plugin-react-swc 3.8.0** – Vite plugin with SWC
* **ESLint 9.22.0** – Linting
* **TypeScript ESLint 8.26.1** – TypeScript-specific linting

**Deployment:**

* **Vercel** – Hosting platform

**Backend (Server)**

**Runtime & Core Framework:**

* **Node.js** – Runtime environment
* **Express 4.21.2** – Web framework
* **TypeScript 5.8.3** – Programming language
* **ts-node 10.9.2** – Run TypeScript directly

**Database & ODM:**

* **MongoDB** – NoSQL database
* **Mongoose 8.15.1** – MongoDB ODM (Object Data Modeling)

**Authentication & Security:**

* **JSON Web Tokens (jsonwebtoken 9.0.2)** – User authentication
* **bcryptjs 3.0.2** – Password encryption

**Email Services:**

* **Nodemailer 7.0.3** – Sending emails
* **Gmail SMTP** – Email service provider

**File Handling:**

* **Multer 2.0.0** – File uploads
* **ExcelJS 4.4.0** – Excel file generation
* **XLSX 0.18.5** – Working with Excel files

**Middleware & Utils:**

* **CORS 2.8.5** – Cross-Origin Resource Sharing
* **Body-parser 2.2.0** – Parsing request bodies
* **dotenv 16.5.0** – Environment variable management

**AI & External APIs:**

* **Google Gemini API** – AI chatbot
* **Cloudinary API** – A cloud service for image storage and processing
* **Axios 1.9.0** – HTTP client

**Deployment:**

* **Vercel** – Hosting platform

**תהליך העבודה בצוות וחלוקת משימות**

**מהנדס המערכת:** מתן ראובן טל

**תיאור תהליך העבודה בצוות:**

בתחילת העבודה על הפרויקט הוסכם כי מתן יהיה מהנדס המערכת. תפקידו כלל תיאום ואחריות על כלל תהליך הפיתוח, הגדרת תכולת העבודה, ניתוח הדרישות, הקצאת משימות לחברי הצוות בהתאם ליכולותיהם, ומעקב אחר ההתקדמות עד להשלמת כל שלב בפרויקט.

העבודה בוצעה בשיתוף פעולה מלא, כאשר כבר בשלב ההתחלתי קיימנו ישיבת תכנון משותפת במהלכה פורקו הדרישות לתתי-משימות ברורות והוקצו משימות לפי תחומי עניין, ידע וניסיון. מתן דאג לעדכן את כל חברי הצוות בהתפתחויות ולוודא עמידה בלוחות הזמנים שהוגדרו.

**חלוקת העבודה:**

|  |  |  |
| --- | --- | --- |
| שם חבר הצוות | משימות שהוקצו | משימות שהושלמו |
| מתן ראובן טל | * הובלת תהליך העבודה לפי עקרונות Agile-Scrum * תיאום בין חברי הצוות והנחיית פגישות סטטוס קבועות * ווידוא עמידה בזמנים ויעילות התהליך * ממשק מול המרצה במידת הצורך * מימוש את מערכת הצ'אטבוט * פיתוח מנגנון ההתחברות לאתר | כל המשימות בוצעו במלואן |
| עומר לב | * בחירת קו עיצובי כולל פלטת צבעים, גופנים וכדומה * מימוש עמוד הבית וקומפוננטות גלובליות * כתיבת תיק משתמש הכולל הסברים וצילומי מסך * הפקת סרטון שיווקי להצגת האתר | כל המשימות בוצעו במלואן |
| טל יגודין | * ניסוח רשימת הדרישות הפונקציונליות והלא פונקציונליות * מימוש בצד המזכירה: דשבורד, ניהול פגישות, סטטוס טיפולים, ייצוא לאקסל * ביצוע Deployment מלא של האתר באמצעות Vercel | כל המשימות בוצעו במלואן |
| נועה שיטרית | * מימוש בצד המזכירה: ניהול מטופלים, צוות המרפאה והוספת מרשמים * הכנת דיאגרמת Use Case * הובלה וביצוע של בדיקות QA לאורך שלבי הפיתוח | כל המשימות בוצעו במלואן |
| אריאל לניאדו | * מימוש בצד המשתמש: עריכת פרטי משתמש, צפייה ועריכת פרטי חיות המחמד * הכנת דיאגרמת ארכיטקטורה מלאה של צד השרת והלקוח * הכנסת נתונים למסד הנתונים (DB) לצורכי ההגשה והדגמה | כל המשימות בוצעו במלואן |
| ספיר גרסטמן | * מימוש בצד המשתמש: קביעת פגישות, צפייה במרשמים והיסטוריית טיפולים * מימוש לוגיקת תורי חירום * כתיבת תיק למתכנת | כל המשימות בוצעו במלואן |

**בנוסף למשימות הייעודיות, כל חברי הצוות לקחו חלק פעיל בתיעוד הקוד ובזיהוי ותיקון באגים לאורך תהליך הפיתוח.**

**שיתוף פעולה בצוות:**

לצורך תיאום רציף, שמרנו על ממשק עבודה בין חברי הצוות באמצעות קבוצת WhatsApp ייעודית. כמו כן, קיימנו פגישה שבועית בשרת Discord בה דנו על אתגרים שהתגלו, עדכוני סטטוס ולקיחת החלטות משותפות להמשך הדרך.

ניהול הבאגים והמשימות בוצע באופן מסודר באמצעות טבלה משותפת, שאפשרה מעקב שוטף, שיתוף פעולה יעיל ופתרון תקלות בזמן אמת.

**רשימת דרישות**

**דרישות פונקציונליות:**

1. המערכת מאפשרת חיבור משתמשים
2. המערכת מאפשרת לבצע איפוס סיסמה למשתמש
3. המערכת מאפשרת להתכתב עם עוזרת ווירטואלית
4. המערכת מאפשרת לראות מידע על ווטרינרים
5. המערכת מאפשרת למשתמש לראות היסטורית טיפולים של החיות
6. המערכת מאפשרת לראות תורים עתידיים של חיות
7. המערכת מאפשרת לבטל תור פעיל של לקוח
8. המערכת מאפשרת לערוך תור פעיל של לקוח
9. המערכת מאפשרת לקבוע תור חדש לחיה של לקוח
10. המערכת מאפשרת לערוך פרופיל משתמש
11. המערכת מאפשר למשתמש לשנות סיסמה
12. המערכת מאפשרת למשתמש לראות פרטי חיות בבעלותו
13. המערכת מאפשר למשתמש להעלות תמונה של חיה
14. המערכת מאפשרת לקבוע תור חירום
15. המערכת מאפשרת צפייה במרשמים
16. המערכת מאפשרת לשלוח מייל
17. המערכת מאפשרת לצפות בסטטיסטיקות
18. המערכת מאפשרת להתנתק מהמערכת
19. המערכת מאפשרת ניהול משתמשי לקוחות המרפאה
20. המערכת מאפשרת הרשמה של משתמשים
21. המערכת מאפשרת הוספת חיה
22. המערכת מאפשרת עריכת פרטי חיה
23. המערכת מאפשרת ניהול צוות עובדים
24. המערכת מאפשרת הוספת עובד
25. המערכת מאפשרת עריכת פרטי עובד
26. המערכת מאפשרת ניהול מרשמים
27. המערכת מאפשרת הוספת מרשם חדש
28. המערכת מאפשרת מחיקת מרשם
29. המערכת מאפשרת שינוי סטטוס מרשם
30. המערכת מאפשרת יצירת דו"ח תורים יומי
31. המערכת מאפשרת שינוי סטטוס תור
32. המערכת תציג מדריך משתמש אינטראקטיבי בצורת חלון מודאלי, שידריך את המשתמש בשלבים על הפיצ’רים המרכזיים במערכת
33. המדריך יאפשר דילוג בין שלבים, דילוג על כל ההדרכה וסגירה

**דרישות לא פונקציונליות:**

**Execution (מאפייני ביצוע):**

1. משתמש יכול להיות לקוח או מזכירה
2. איפוס סיסמה שולח מייל למשתמש
3. עוזרת ווירטואלית ניתן לשימוש על ידי משתמש לא מחובר, לקוח ומזכירה
4. העוזרת הווירטואלית מאפשרת לבצע את הפעולות הבאות: לקבוע פגישה, לבטל פגישה, לראות היסטוריה, להציג שעות פעילות של המרפאה, להציג מידע קשר, לקבוע תור חירום ולשאול שאלה בטקסט חופשי
5. הפעולות הבאות דורשות חיבור על ידי לקוח בעוזרת הווירטואלית: לקבוע פגישה, לבטל פגישה, לראות היסטוריה, לקבוע תור חירום
6. מידע של ווטרינרים אפשרי לראות על ידי משתמש לא רשום, לקוח ומזכירה
7. מידע על ווטרינרים כולל: שם, תמונה, תפקיד, ותיאור
8. היסטורית טיפולים של חיה תתאפשר לצפייה על ידי לקוח
9. היסטורית טיפולים של חיה כוללת את: שם החיה, ווטרינר מטפל, סוג ביקור, עלות ביקור, תאריך הביקור, תיאור משתמש, וסיכום פגישה על ידי ווטרינר
10. סוגי הביקור האפשריים הם: בדיקת בריאות, חיסון, עיקור, סירוס, ניקוי שיניים, טיפול חירום, ניתוח, הדמיה אבחנתית, בדיקות דם, מעקב, טיפוח, ייעוץ התנהגותי, שבב מיקרוסקופי.
11. תורים עתידיים ניתן לראות על ידי לקוח
12. תורים עתידיים כוללים: ווטרינר מטפל, שם חיה, תאריך ושעה, משך טיפול, תיאור כללי, סוג טיפול
13. ביטול תור אפשרי על ידי לקוח או מזכירה
14. מזכירה יכולה לבטל תור של כל לקוח אחר
15. לאחר אימות המשתמש בתהליך איפוס הסיסמה, המשתמש יבחר סיסמה חדשה
16. לקוח יכול לבטל רק תורים שלו
17. ביטול תור מעביר את התור למצב בוטל מבלי אפשרות לשנות חזרה
18. ערכית תור של חיה אפשרית על ידי לקוח
19. עריכת תור מזיזה את תאריך התור לתאריך חדש
20. קביעת תור אפשרית על ידי לקוח ומזכירה
21. קביעת תור על ידי לקוח יכולה להיות רק מסוג בדיקת בריאות
22. עריכת פרופיל משתמש מתאפשרת על ידי לקוח ומזכירה
23. לקוח יכול לערוך את הפרטים הבאים: מספר טלפון, עיר מגורים, רחוב, מיקוד
24. שינוי סיסמה אפשרי על ידי לקוח או מזכירה
25. סיסמה צריכה להכיל לפחות 6 תווים
26. משתמש לא יכול להחליף לאותה סיסמה שכבר יש לו כרגע
27. צפייה בפרטי חיה בבעלות לקוח מתאפשר על ידי לקוח ומזכירה
28. לקוח יכול לראות רק חיות בבעלותו
29. פרטי חיה מכילים: סוג חיה, גזע, שנת לידה, משקל, מין והאם פעיל
30. העלאת תמונה של חיה אפשרית על ידי לקוח
31. אם אין תמונה לחיה, תוצג תמונה ברירת מחדל
32. תמונות ברירת מחדל שקיימות הן: חתול, כלב, עז, תוכי, ארנב, נחש, חייזר
33. חיה שלא מסוג שיש תקבל תמונה של חייזר
34. לכל תמונה של חיה יש גרסת זכר וגרסת נקבה
35. קביעת תור חירום מתאפשרת על ידי לקוח ומזכירה
36. לקוח יכול לקבוע תור חירום רק לחיות שבעלותו
37. תור חירום מבטל תורים אחרים שהם לא דחופים
38. תור לא דחוף הוא מסוג:
39. צפייה במרשמי לקוח אפשרית על ידי לקוח ומזכירה
40. מרשם כולל: שם תרופה, כמות, תאריך הנפקה, תוקף מרשם, סיבת הנפקה, האם הונפק, חיה
41. שליחת מייל מתבצעת על ידי מערכת שליחת מיילים
42. סוגי מיילים שקיימים הם: מייל איפוס סיסמה, מייל חירום
43. שליחת מייל חירום מתבצעת לאחר קביעת תור חירום
44. צפייה בסטטיסטיקות מתאפשר עבור מזכירה
45. סטטיסטיקות כוללות: סך הכל מטופלים, כמות צוות, מספר תורים היום, רווח שבועי, רווח חודשי, תורים היום (פירוט), תורים בהמשך שבוע (פירוט), סיכום שבועי (כולל מספר תורים סך הכל, תורים שהושלמו, תורים שבוטלו, אחוז ביצוע ורווח), סוג טיפול פופולרי (מספר של כל סוג פגישה)
46. התנתקות מתאפשרת עבור לקוח ומזכירה
47. ניהול משתמשי הלקוחות מתאפשר עבור מזכירה
48. ניהול משתמשי לקוחות כולל: הוספת לקוח חדש, הוספת חיה ללקוח, עריכת פרטי חיה ועריכת פרטי משתמש
49. פרטי משתמש כוללים: שם פרטי, שם משפחה, מספר טלפון, אימייל, רחוב, עיר, מיקוד
50. הרשמה של משתמשים תכלול הרשמה של לקוחות בלבד
51. הרשמה של לקוחות מתאפשרת עבור מזכירה
52. הוספת חיה ללקוח מתאפשרת עבור מזכירה
53. עריכת פרטי חיה מתאפשרת עבור מזכירה
54. סוגי העובדים הם: ראש ווטרינרים וראש מרפאה, ווטרינר, עוזר ווטרינר, מזכירה
55. ניהול צוות העובדים כולל: הוספת עובד חדש ועריכת פרטי עובד קיים.
56. פרטי עובד כוללים: שם פרטי, שם משפחה, אימייל, מספר טלפון, תפקיד, שנות ניסיון, מומחיות, מספר רישיון, תיאור, תמונה, ימי פעילות
57. ניהול צוות העובדים מתאפשר עבור מזכירה
58. ניהול מרשמים מתאפשר עבור מזכירה
59. ניהול מרשמים כולל: הוספת מרשם חדש, מחיקת מרשם קיים ושינוי סטטוס מרשם קיים.
60. סוגי סטטוס מרשם הם: ממתין, ובוצע
61. דו"ח תורים יומי כולל: סיכום של כמה תורים בוצעו באותו יום ומה המספר של תורים בסטטוס בוצע/בוטל/נקבע, סך רווח יומי, טבלה עם פירוט מלא של פרטי התור
62. דו"ח תורים יומי מיוצא לקובץ אקסל
63. שינוי סטטוס של תור מתאפשר עבור מזכירה
64. סוגי סטטוס של תור הם: נקבע, אושר, בתהליך, בוצע, לא הגיע ובוטל
65. המערכת תספק מדריך משתמש אינטראקטיבי שמסביר בשלבים את הפיצ’רים המרכזיים במערכת, עם תמיכה במובייל ודסקטופ

**Evolution (מאפייני תחזוקה והתפתחות):**

1. המערכת תאפשר הוספת שדות נתונים חדשים (למשל מידע נוסף עבור חיות) מבלי לדרוש שינוי מבני במבנה בסיס הנתונים, באמצעות שימוש בסכמת נתונים גמישה (mongo DB)
2. המערכת תאפשר הוספה פשוטה של סוגי טיפולים חדשים ללא צורך בשינויים משמעותיים בקוד.
3. המערכת תאפשר שינוי של דוחות קיימים והוספת דוחות חדשים על פי דרישות המשתמש.
4. המערכת תאפשר הוספת מודולים חדשים (כגון מודול לתזמון מתקדם או מודול להודעות) ללא פגיעה במודולים קיימים.

**דרישות חיצוניות:**

**Mongo DB:**

1. המערכת תתקשר עם Mongo DB Atlas לצורך שמירת נתוני משתמשים, חיות, תורים, מרשמים וצוות
2. הקשר יתבצע באמצעות ספריית Mongoose

**Gmail SMTP:**

1. המערכת תתקשר עם Gmail SMTP server על מנת לשלוח מיילים
2. הקשר יתבצע באמצעות API של Gmail SMTP server

**RESTful API:**

1. המערכת מספקת RESTful API לתקשורת בין צד הלקוח (React) לצד השרת (Express.js).
2. התקשורת מתבצעת בפורמט JSON, עם אימות והרשאות באמצעות JWT.

**ממשק משתמש (React UI):**

1. המערכת כוללת ממשק משתמש אינטראקטיבי מבוסס React, רספונסיבי ומותאם לדסקטופ ולמובייל.
2. הממשק מאפשר ניהול תורים, חיות מחמד, משתמשים וצפייה בדוחות וסטטיסטיקות.

**Google Gemini API:**

1. המערכת מבצעת אינטגרציה עם Google Gemini API לצורך הפעלת צ'אטבוט חכם ללקוחות.
2. החיבור מתבצע באמצעות REST API מאובטח, עם שליחת בקשות וקליטת תשובות דינמיות.

**Cloudinary API:**

1. המערכת תתקשר עם שירות Cloudinary לצורך העלאת ושמירת תמונות של חיות מחמד בענן.
2. ההעלאה מתבצעת באמצעות Cloudinary API עם אימות בעזרת מפתחות גישה (API Key, Secret) המוגדרים במשתני סביבה.

**ארכיטקטורת האתר**

תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**תרשים Use-Case**

תמונה שמכילה טקסט, תרשים, תוכנית, שחור ולבן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**מבנה האתר (תיקיות וקבצים)**

**Client:**

| App.tsx

| main.tsx

| vite-env.d.ts

|

+---assets

| | aboutSecImg.png

| | logo.png

| |

| \---animals

|

+---components

| | FooterSection.tsx

| | Navbar.tsx

| | ScrollToHash.tsx

| | ThemeToggle.tsx

| |

| +---auth

| | FirstTimePasswordChangeModal.tsx

| | Login.tsx

| | LogoutButton.tsx

| | ProtectedRoute.tsx

| | SignUpForm.tsx

| |

| +---chatbot

| | ChatButton.tsx

| | ChatWindow.tsx

| | MessageBubble.tsx

| |

| +---homePage

| | AboutSection.tsx

| | ContactSection.tsx

| | TeamMemberCard.tsx

| | TeamSection.tsx

| |

| +---secretary

| | | AddAppointmentForm.tsx

| | | AddPatientForm.tsx

| | | AddPetForm.tsx

| | | AppointmentNotesInline.tsx

| | | AppointmentNotesModal.tsx

| | | AppointmentView.tsx

| | | ChangePasswordModal.tsx

| | | ConfirmCancelModal.tsx

| | | DashboardButton.tsx

| | | EditPatientForm.tsx

| | | EditPetForm.tsx

| | | EmergencyAppointmentModal.tsx

| | | EnhancedSecretaryWelcome.tsx

| | | ManagePatientsView.tsx

| | | PatientCard.tsx

| | | PatientList.tsx

| | | PrescriptionManagement.tsx

| | | StaffManagement.tsx

| | | TailwindCalendar.tsx

| | |

| | +---appointment

| | | AppointmentFormFields.tsx

| | | ClientSearch.tsx

| | | PetSelection.tsx

| | | TimeSlotSelector.tsx

| | |

| | +---prescription

| | | ClientSearchPrescription.tsx

| | | MedicineSearch.tsx

| | | PetSelectionPrescription.tsx

| | |

| | \---staff

| | AvailabilityScheduler.tsx

| | ImageUpload.tsx

| | StaffCard.tsx

| | StaffForm.tsx

| | StaffGrid.tsx

| |

| \---user

| | AddAppointmentFormToClient.tsx

| | AppointmentViewClient.tsx

| | CancelReasonModal.tsx

| | ClientProfile.tsx

| | EditAppointmentForm.tsx

| | EditUserProfile.tsx

| | PetCard.tsx

| | PetImageUpload.tsx

| | PetLastTreatment.tsx

| | PrescriptionCard.tsx

| | ShowPrescriptions.tsx

| | TreatmentCard.tsx

| | TreatmentHistory.tsx

| | TutorialModal.tsx

| | UnfulfilledPrescriptions.tsx

| | UserInformationCard.tsx

| | UserNavButton.tsx

| |

| \---appointments

| AppointmentFormFieldsClient.tsx

| EmergencyAppointmentModal.tsx

| PetSelectionClient.tsx

| TimeSlotSelectorClient.tsx

|

+---config

| api.ts

|

+---pages

| ClientPage.tsx

| Home.tsx

| SecretaryPage.tsx

|

+---services

| appointmentService.ts

| appointmentUtils.ts

| medicineService.ts

| patientService.ts

| petService.ts

| prescriptionService.ts

| staffService.ts

| statisticsService.ts

| userService.ts

|

+---styles

| index.css

| mobile-utilities.css

|

\---types

index.ts

prescription.ts

**Server:**

| server.ts

|

+---models

| appointmentSchema.ts

| medicineSchema.ts

| petSchema.ts

| prescriptionSchema.ts

| staffSchema.ts

| treatmentSchema.ts

| userSchema.ts

|

+---routes

| appointment.route.ts

| chatbot.route.ts

| medicine.route.ts

| pet.route.ts

| prescription.route.ts

| staff.route.ts

| statistics.route.ts

| treatment.route.ts

| users.route.ts

|

+---services

| appointmentService.ts

| authMiddleware.ts

| emailService.ts

| treatmentService.ts

|

\---utils

dateParse.ts

dbConnection.ts

**מבנה מסד הנתונים (DB)**



**תיק למתכנת**

**פונקציות מרכזיות**

**צד השרת (Server):**

**server.ts**

* **app.use(cors(corsOptions))** **-** מאפשר בקשות ממקורות מורשים (ניהול CORS).
* **app.use(bodyParser.json())-** מאפשר קבלת נתוני JSON בבקשות API.
* **app.use('/api', ensureDbConnection)-** מוודא שיש חיבור למסד הנתונים לפני כל קריאה ל-API.
* **app.use('/uploads', express.static(...))-** מגיש קבצים סטטיים שהועלו למערכת.
* **app.listen(PORT, callback)-** מריץ את השרת על פורט מוגדר מראש.

**appointmentService.ts**

* **createAppointment(userId, petId, staffId, date, time, type, description)-** יוצר תור חדש עבור חיה, לקוח, ותאריך/שעה מסוימים.
* **cancelAppointment(userId, appointmentId)-** מבצע ביטול רך (status = CANCELLED) של תור לפי מזהה, רק עבור אותו לקוח.
* **getFutureAppointments(userId)-** מחזיר את כל התורים העתידיים של משתמש מסוים.
* **getActiveVets()-** מחזיר רשימת כל הווטרינרים הפעילים במערכת.
* **getAppointmentHistoryByPet(petId)-** שולף היסטוריית תורים (עבר) של חיית מחמד לפי מזהה.
* **getTakenTimesForDay(date)-** מחזיר רשימת שעות שכבר תפוסות (בתורים קיימים) בתאריך מסוים.
* **getAppointmentsByPet(petId)-** שולף את כל התורים (עבר/עתיד) של חיית מחמד מסוימת.
* **findAvailableVetForEmergency(now=Date)-** מחפש רופא פנוי בדחיפות לתור חירום - לוקח בחשבון אי-ביטול ניתוחים, בוחר את הווטרינר עם הכי פחות התנגשויות בלו"ז.

**treatmentService.ts**

* **getPetHistory(petId)-** מחזיר את רשימת כל הטיפולים של חיה מסוימת (מסודרים מהחדש לישן).

**authMiddleware.ts**

* **authMiddleware(req, res, next)-** Middleware המוודא שכל בקשת API מוגנת מכילה JWT תקין ב-header. אם תקין - מוסיף את המידע ל-req.user וממשיך, אחרת עונה 401.

**emailService.ts**

* **sendPasswordResetEmail(to, code)-** שולח למשתמש קוד לאיפוס סיסמה למייל עם עיצוב.
* **sendEmergencyCancelEmail(to, appointmentDate, vetName)-** מודיע למשתמש על ביטול תור עקב חירום (מפרט תאריך/רופא).
* **sendEmergencyVetAlertEmail(to, emergencyTime, description, emergencyReason?)-** שולח לרופא התראה על תור חירום שנקבע עבורו.
* **sendEmergencySecretaryAlertEmail({ userId, petId, vetId, date, time, description, emergencyReason? })-**שולח לכל המזכירות התראה על תור חירום שנקבע, כולל פרטי לקוח, חיה, וטרינר וזמן.
* **sendEmergencyOwnerConfirmationEmail({ to, petName, vetName, date, time, description, emergencyReason? })-** שולח אישור תור חירום לבעלים של החיה עם כל הפרטים.
* **sendNoVetAvailableEmergencyEmail({ to, petName })-** שולח למשתמש עדכון כי אין וטרינר זמין ברגע זה לתור חירום (כולל הנחיות המשך).

**users.route.ts**

* **loginUser(credentials)-** מאמת משתמש ומנפיק טוקן גישה.
* **registerUser(userData)-** רושם משתמש חדש עם סיסמה מוצפנת.
* **changePassword(passwordData) -** מחליף סיסמה קיימת לסיסמה חדשה.
* **getUserProfile(userId) -** מחזיר את פרטי המשתמש לפי מזהה.
* **updateUserProfile(userId, updatedData) -** מעדכן פרטים קיימים של המשתמש.
* **deleteUser(userId) -** מוחק משתמש מהמערכת.
* **verifyToken(token)-** בודק אם הטוקן תקף ומחזיר סטטוס התחברות.

**appointment.route.ts**

* **createAppointment(appointmentData) -** יוצר תור חדש עבור לקוח/חיה.
* **getAllAppointments()-** מחזיר את כל התורים במערכת.
* **getAppointmentsByDate(date**) **-** מחזיר תורים לפי תאריך מסוים.
* **getAppointmentsByClient(clientId) -** מחזיר את כל התורים של לקוח מסוים.
* **updateAppointment(appointmentId, updatedData) -** מעדכן פרטי תור קיים.
* **cancelAppointment(appointmentId) -** מבטל תור.
* **updateAppointmentStatus(appointmentId, newStatus)** **-** משנה סטטוס של תור.
* **getAvailableTimeSlots(date)-** מחזיר את כל הזמנים הפנויים לתורים.

**pet.route.ts**

* **addPet(petData) -** מוסיף חיית מחמד חדשה למערכת.
* **getPetsByOwner(ownerId) -** שולף את כל החיות של בעלים מסוים.
* **getPetById(petId) -** מחזיר פרטי חיה לפי מזהה.
* **updatePet(petId, updatedData)-** מעדכן פרטי חיה קיימת.
* **deletePet(petId)-** מוחק חיה מהמערכת.
* **getPetMedicalHistory(petId) -** מחזיר היסטוריה רפואית של חיה.
* **addMedicalRecord(petId, recordData) -** מוסיף רשומה רפואית לחיה.
* **uploadPetImage(petId, imageFile)** **-** מעלה תמונה חדשה של חיית מחמד ל-Cloudinary ומעדכן את כתובת התמונה במסד הנתונים של החיה.  
   **getPetImageUrl(petId)** **-** מחזיר את כתובת ה-URL של תמונת החיה (אם קיימת), לשימוש באפליקציה.

**prescription.route.ts**

* **createPrescription(prescriptionData)-** יוצר מרשם חדש עבור חיה/לקוח.
* **getAllPrescriptions() -** מחזיר את כל המרשמים במערכת.
* **getPrescriptionsByPet(petId) -** מחזיר מרשמים לפי חיה.
* **getPrescriptionsByClient(clientId) -** מחזיר מרשמים לפי לקוח.
* **updatePrescription(prescriptionId, updatedData) -** מעדכן מרשם קיים.
* **cancelPrescription(prescriptionId)-** מבטל מרשם.
* **updatePrescriptionStatus(prescriptionId, newStatus) -** משנה סטטוס מרשם.
* **getActivePrescriptions()-** מחזיר מרשמים פעילים במערכת.

**medicine.route.ts**

* **addMedicine(medicineData) -** מוסיף תרופה חדשה למאגר.
* **getAllMedicines()-** שולף את כל התרופות במערכת.
* **searchMedicines(query)-** מחפש תרופות לפי מחרוזת.
* **getMedicineById(medicineId) -** מחזיר פרטי תרופה לפי מזהה.
* **updateMedicine(medicineId, updatedData) -** מעדכן פרטי תרופה.
* **deleteMedicine(medicineId) -** מוחק תרופה מהמערכת.
* **updateMedicineStock(medicineId, newStock) -** מעדכן מלאי של תרופה.

**staff.route.ts**

* **addStaffMember(staffData) -** מוסיף איש צוות חדש.
* **getAllStaffMembers() -** מחזיר את כל אנשי הצוות.
* **getStaffMemberById(staffId) -** שולף פרטי איש צוות לפי מזהה.
* **updateStaffMember(staffId, updatedData) -** מעדכן נתוני עובד.
* **deleteStaffMember(staffId) -** מוחק איש צוות.
* **getStaffByRole(role) -** מחזיר אנשי צוות לפי תפקיד.
* **updateStaffSchedule(staffId, scheduleData) -** מעדכן לוח זמנים של איש צוות.

**statistics.route.ts**

* **getDashboardStats() -** מחזיר סטטיסטיקות כלליות (תורים, לקוחות, הכנסות וכו’).
* **getDailyAppointmentStats() -** מחזיר נתוני תורים יומיים.
* **getMonthlyAppointmentStats() -** מחזיר נתוני תורים חודשיים.
* **getNewClientsStats() -** מחזיר נתוני לקוחות חדשים.
* **getMonthlyRevenueStats() -** מחזיר נתוני הכנסות חודשיים.
* **getPetsByTypeStats() -** מחזיר סטטיסטיקות לפי סוגי חיות.
* **getPopularTreatmentsStats()-** מחזיר נתוני טיפולים פופולריים.

**treatment.route.ts**

* **addTreatment(treatmentData) -** מוסיף טיפול רפואי חדש.
* **getAllTreatments()-** מחזיר את כל הטיפולים במערכת.
* **getTreatmentsByPet(petId)-** מחזיר טיפולים של חיה מסוימת.
* **getTreatmentById(treatmentId) -** מחזיר פרטי טיפול לפי מזהה.
* **updateTreatment(treatmentId, updatedData) -** מעדכן טיפול קיים.
* **deleteTreatment(treatmentId) -** מוחק טיפול.
* **updateTreatmentStatus(treatmentId, newStatus) -** משנה סטטוס טיפול.

**chatbot.route.ts**

* **processMessage(messageData) -** מעבד הודעת משתמש ושולח תשובה מהבוט.
* **getSuggestions() -** מחזיר הצעות לשאלות נפוצות לצ’אטבוט.
* **submitFeedback(feedbackData) -** שומר משוב משתמש על מענה בוט.
* **getChatHistory(userId) -** מחזיר היסטוריית צ’אט למשתמש.
* **clearChatHistory(userId) -** מוחק היסטוריית צ’אט למשתמש.

**dbConnection.ts (utils)**

* **ensureDbConnection(req, res, next) -** Middleware המוודא חיבור פעיל ל-MongoDB לכל בקשת API.
* **connectToDatabase()-** יוצר חיבור חדש למסד הנתונים במידת הצורך.
* **handleConnectionError(error) -** מטפל בשגיאות חיבור למסד.
* **reconnectToDatabase() -** מתחבר מחדש אם החיבור למסד הנתונים נפל.

**צד הלקוח (Client):**

**App.tsx**

* **<BrowserRouter> -** ניהול ניווט SPA בין עמודי האפליקציה.
* **<ProtectedRoute> -** מגן על נתיבים רגישים ודורש התחברות.
* **useAuthContext() -** ניהול והצגת סטטוס התחברות המשתמש.
* **<Routes> / <Route> -** הגדרת נתיבי הניווט באפליקציה.

**main.tsx**

* **ReactDOM.createRoot()-** אתחול שורש אפליקציית React בדפדפן.
* **root.render(<App />) -** רינדור האפליקציה.

**SecretaryPage.tsx**

* **handleViewChange(view) -** מחליף בין תצוגות ראשיות (תורים, דוחות, מטופלים).
* **fetchDashboardData() -** טוען את נתוני הדשבורד עבור המזכירה.
* **handleLogout() -** מבצע התנתקות ומנקה מצב התחברות.
* **renderCurrentView() -** מציג את התצוגה הנוכחית לפי בחירת המזכירה.

**ClientPage.tsx**

* **fetchClientData() -** טוען את פרטי הלקוח המחובר.
* **fetchPetData() -** טוען את נתוני חיות המחמד של הלקוח.
* **fetchAppointments() -** טוען את כל התורים הרלוונטיים ללקוח.
* **fetchPrescriptions() -** טוען מרשמים פעילים של הלקוח.

**Login.tsx**

* **handleLogin(credentials) -** מבצע התחברות עם אימות סיסמה.
* **validateCredentials()-** בודק תקינות נתונים לפני שליחת התחברות.
* **handle2FA(code) -** מטפל באימות דו־שלבי (אם מופעל).
* **sendVerificationCode() -** שולח קוד אימות לאימייל של המשתמש.

**ChangePasswordModal.tsx**

* **handlePasswordChange(newPassword) -** מחליף את הסיסמה של המשתמש.
* **validateNewPassword(password) -** בודק תקינות של סיסמה חדשה.
* **checkCurrentPassword(password) -** מאמת סיסמה נוכחית.

**AppointmentView.tsx**

* **fetchAppointments() -** טוען את רשימת התורים מהשרת.
* **handleAppointmentCreate(data) -** יוצר תור חדש.
* **handleAppointmentUpdate(id, data) -** מעדכן תור קיים.
* **handleAppointmentCancel(id) -** מבטל תור.
* **filterAppointmentsByDate(date) -** מסנן תורים לפי תאריך.

**TailwindCalendar.tsx**

* **renderCalendarDays()-** מרנדר את ימי החודש בלוח השנה.
* **handleDateClick(date) -** מטפל בבחירת תאריך.
* **navigateMonth(direction) -** עובר בין חודשים בלוח השנה.
* **highlightAppointmentDays(dates) -** מדגיש ימים עם תורים קיימים.

**ManagePatientsView.tsx**

* **searchPatients(query) -** מבצע חיפוש מטופלים לפי שם/ת"ז.
* **handlePatientSelect(id) -** בוחר מטופל לעריכה או צפייה.
* **deletePatient(id) -** מוחק מטופל.
* **exportPatientData()-** מייצא את רשימת המטופלים לאקסל.

**TutorialModal.tsx**

* **highlightStep(index) -** מדגיש כפתור/רכיב רלוונטי במסך בהתאם לשלב הנוכחי.
* **handleNext()-** עובר לשלב הבא בהדרכה.
* **handleSkipOrClose()-** מדלג על כל ההדרכה או סוגר אותה.
* **handleGotIt()-** מסיים את ההדרכה ומסתיר את המודאל.

**useAuth (hook)**

* **login(credentials) -** מבצע התחברות.
* **logout()-** מנתק את המשתמש.
* **isAuthenticated()-** בודק האם המשתמש מחובר.
* **refreshToken()-** מחדש טוקן התחברות אם נדרש.

**useApi (hook)**

* **get(url, options?)-** קריאת GET לשרת (API).
* **post(url, data, options?) -** קריאת POST לשרת.
* **put(url, data, options?) -** קריאת PUT לשרת.
* **delete(url, options?) -** קריאת DELETE לשרת.
* **handleError(error) -** טיפול בשגיאות של קריאות API.

**useLocalStorage (hook)**

* **setItem(key, value) -** שמירת ערך ב-localStorage.
* **getItem(key) -** שליפת ערך מ-localStorage.
* **removeItem(key) -** הסרת ערך מ-localStorage.
* **clear()-**ניקוי כל הערכים ב-localStorage.

**authService**

* **login(credentials) -** שליחת בקשת התחברות לשרת.
* **register(userData) -** רישום משתמש חדש דרך ה־API.
* **changePassword(passwordData) -** שינוי סיסמה מהפרופיל.
* **verifyToken(token) -** בדיקת תקפות טוקן התחברות.

**appointmentService**

* **createAppointment(data)-** יצירת תור חדש בשרת.
* **getAppointments(filters) -** שליפת תורים לפי סינון מהשרת.
* **updateAppointment(id, data) -** עדכון תור קיים בשרת.
* **cancelAppointment(id) -** ביטול תור קיים.

**petService**

* **addPet(petData) -** הוספת חיה חדשה דרך ה־API.
* **getPets(ownerId) -** שליפת רשימת חיות של לקוח.
* **updatePet(id, data) -** עדכון חיה בשרת.
* **deletePet(id) -** מחיקת חיה.

**prescriptionService**

* **createPrescription(data) -** יצירת מרשם חדש דרך ה־API.
* **getPrescriptions(filters) -** שליפת מרשמים מהשרת.
* **updatePrescription(id, data) -** עדכון מרשם.
* **cancelPrescription(id) -** ביטול מרשם.

**שימוש ב- API בפרוייקט:**

**ארכיטקטורת ה-API במערכת:**

המערכת פותחה במבנה מודולרי המפריד באופן ברור בין שכבות הנתיבים (routes), הלוגיקה העסקית (services) ומודלי הנתונים (models). כל נתיב API (קבצי ‎.route.ts‎ בתיקיית routes) מייצג נקודת קצה (endpoint) אליה ניתן לגשת דרך פרוטוקול HTTP, ומיועד לטיפול בפעולות כמו קביעת תור, שליפת מידע רפואי, חיפוש צוות, שליחת אימיילים ועוד.

כל פנייה שמתקבלת לנתיב API מתועלת על ידי Express ל-router המתאים, שמעביר את הבקשה לשכבת השירותים (services). בשכבה זו ממומשת הלוגיקה העסקית המלאה - שליפת נתונים ממסד הנתונים (MongoDB), עדכון, יצירה ומחיקה, טיפול באימות והרשאות, שליחת התראות במייל, ולפעמים גם קריאות API חיצוניות (למשל Google Gemini).

הגישה המודולרית הזו מאפשרת תחזוקה קלה של המערכת, הוספה מהירה של פיצ'רים חדשים, יכולת לשדרג או להחליף רכיבים מבלי לפגוע בשאר המערכת, ושילוב פשוט של שירותים חיצוניים.  
בנוסף, מבנה זה תורם לאבטחת המערכת ולניהול הרשאות ובדיקות תקינות, ומאפשר ביצוע כל פעולה רלוונטית - הן מצד השרת (באופן פנימי), הן מהצ'אט-בוט והן מממשקי משתמש חיצוניים - דרך ה-API האחיד.

**שימוש ב-API בקובץ : chatbot.route.ts**

מהות השימוש ב-API:

* אינטגרציה עם Gemini API (Google) לצ'אט-בוט חכם.
* קריאות ל-API פנימיים בשרת באמצעות axios.
* שימוש בפונקציות שירות (services) למשל: קביעת תורים, שליפת וטרינרים, חיפוש מידע רפואי.

**מיקום:** server/routes/chatbot.route.ts

**תיאור כללי:**

מסלול הצ’אט-בוט מהווה שער אחיד לכלל האינטראקציה של משתמשים (לקוחות וצוות) עם הבוט הווטרינרי "Kayo".  
הבוט כולל זיהוי פקודות, תפריט ראשי, קביעת תורים, היסטוריית טיפולים, זיהוי צוות, שאלות חופשיות ועוד.

1. **Google Gemini API**

* **כתובת:**  
  https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent
* **שימוש:**  
  משמש כ-Fallback וכלי בינה מלאכותית עבור תשובות לשאלות חופשיות של משתמש (כולל הסבר טיפים, מידע רפואי, ועוד).
* **שיטת קריאה:**  
  קריאה ב-POST דרך axios, כולל ניהול מפתחות API (הגדרה ב-env)
* **אבטחה וטיפול בתקלות:**  
  המערכת בודקת אם קיים מפתח API ,כוללת מנגנון retry אוטומטי במקרה של תקלה זמנית (503).

**דוגמה:**

const GEMINI\_API\_URL =

  "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent";

const GEMINI\_API\_KEY = (process.env.GEMINI\_API\_KEY || "").trim();

/\*\* Minimal wrapper around Gemini Flash with 503 retry. \*/

async function askGemini(*prompt*: *string*, *retries* = 3): *Promise*<*string*> {

  // Check if API key is configured

  if (!GEMINI\_API\_KEY) {

    console.warn('GEMINI\_API\_KEY not configured');

    return "I'm sorry, but I'm currently unavailable. Please try the main menu options or contact support.";

  }

  const system = `You are Kayo, the friendly assistant for FurEver Friends.`

               + ` Answer briefly and warmly.\nUser: "${*prompt*}"`;

  try {

    const { data } = await axios.post(

      `${GEMINI\_API\_URL}?key=${GEMINI\_API\_KEY}`,

      { contents: [{ parts: [{ text: system }] }] },

      { timeout: 10000 } // 10 second timeout for Vercel

    );

    return data.candidates?.[0]?.content?.parts?.[0]?.text

        ?? "Sorry, I didn't understand.";

  } catch (*err*: *any*) {

    console.error('Gemini API error:', err.response?.data || err.message);

    if (*retries* && err.response?.status === 503) return askGemini(*prompt*, *retries* - 1);

    return "I'm having trouble connecting to my knowledge base. Please try the menu options or contact support.";

  }

}

1. **קריאות פנימיות ל־API בשרת**

במערכת קיימות קריאות פנימיות בין שירותים שונים, שמבוצעות מתוך השרת עצמו (ולא מהלקוח), בעיקר באמצעות axios. קריאות אלו מאפשרות לצ'אט-בוט לבצע מגוון פעולות במערכת בזמן אמת , כגון קביעת תורים, חיפוש רופאים או שליפת מידע - מבלי להפעיל לוגיקה כפולה או לגשת ישירות למסד הנתונים מתוך קוד הצ'אט-בוט.

* **חיפוש רופא/צוות:**

**:/api/staff/search (GET)**   
חיפוש רופא לפי שם. הקריאה נשלחת מתוך הצ'אט-בוט (chatbot.route.ts) באמצעות axios, כאשר המשתמש שואל על רופא מסוים  
הנתיב מופעל ב-routes/staff.route.ts ומעביר את הבקשה ללוגיקה העסקית שב-services/staffService.ts

**דוגמה:**

try {

    // Query the staff search endpoint

    const response = await axios.get(

      `${process.env.SERVER\_URL || "http://localhost:3000"}/api/staff/search`,

      { params: { name } }

    );

    const doc = response.data;

**/api/staff/search/specialty (GET):**  
חיפוש לפי תחום התמחות או תפקיד (למשל: "קרדיולוג", "מזכירה").  
הקריאה מתבצעת מתוך הצ'אט-בוט, שמעביר את ההתמחות שנבחרה לנתיב המתאים, שמעובד על ידי פונקציית שירות מתאימה בservices/staffService.ts.

* **קביעת תור חירום:**

**/api/appointments/emergency (POST) :**  
קביעת תור חירום אוטומטית, הכוללת איתור רופא זמין, עדכון הנתונים ושליחת התראות במייל.  
הקריאה מתבצעת מתוך הצ'אט-בוט כאשר המשתמש בוחר באפשרות "Emergency".  
הנתיב מנוהל ב-routes/appointment.route.ts , ומפעיל את הלוגיקה למציאת ושיבוץ רופא ב-services/appointmentService.ts, כולל שליחת התראות דרך services/emailService.ts..

* **פונקציות שירות:**

פונקציות כמו getFutureAppointments, getActiveVets, createAppointment, cancelAppointment ממומשות כולן בקובץ services/appointmentService.ts  
פונקציות אלו נקראות מתוך קוד הצ'אט-בוט או נתיבי Express, פועלות ישירות מול מסד הנתונים (MongoDB) באמצעות Mongoose, ומאפשרות לצ'אט-בוט להציג למשתמשים נתונים עדכניים ולבצע פעולות (כמו קביעת תור, ביטול תור, הצגת תורים עתידיים וכו').

**קובץ : services/emailService.ts**

**מיקום** server/services/emailService.ts **:**

**תיאור כללי:**

שכבת השירות האחראית על שליחת הודעות דוא"ל אוטומטיות במערכת.  
משמשת לשליחת התראות על תורים (כולל חירום), איפוס סיסמה, הודעות על ביטולים, התראות לצוות ולמזכירות.

לשליחת הודעות דוא"ל אוטומטיות נעשה שימוש בספריית Nodemailer שמבצעת את השליחה דרך שרת הדוא"ל של Gmail באמצעות פרוטוקול SMTP. כל פרטי הגישה (שם משתמש וסיסמה אפליקטיבית) נשמרים כמשתני סביבה בקובץ ‎.env, והודעות המייל מעוצבות כ־HTML דינמי לפי ההקשר.

**שימוש ב API -עיקרי:**

* **Nodemailer + SMTP (Gmail)**
  + **ספריה:**  
     Nodemailerשל Node.js לשליחת דוא"ל.
  + **שרת:**  
    Gmail SMTP (smtp.gmail.com)
  + **אימות:**  
    משתמש בסיסמה/קוד אפליקציה דרך משתני סביבה EMAIL\_USER) (EMAIL\_PASS.
  + **שימוש:**
    - שליחת מיילים עם עיצוב HTML עשיר.
    - אפשרות לשליחת קבצים/מידע עשיר (לא בשימוש כרגע).
    - תמיכה במיילים למספר נמענים במקביל (לדוג' התראה לכל המזכירות).

**דוגמאות:**  
**1**. **הגדרת transporter:**

// Create a Nodemailer transporter using Gmail SMTP.

// Credentials are stored in environment variables for security.

 const transporter = nodemailer.createTransport({

  service: "gmail",

  auth: {

    user: process.env.EMAIL\_USER, // your Gmail address ( youremail@gmail.com)

    pass: process.env.EMAIL\_PASS, // an app password or your Gmail password

  },

});

**2**. **שליחת אימייל (קטע HTML חתוך):**

async function sendPasswordResetEmail(*to*: *string*, *code*: *string*) {

  await transporter.sendMail({

    from: process.env.EMAIL\_USER,

    to,

    subject: "FurEver Friends - Password Reset Code",

    html: `

      <div st……….

**קובץ : pet.route.ts**  
**מיקום:** server/routes/pet.route.ts  
**תיאור כללי:**  
שכבת השירות האחראית על העלאת תמונות חיות מחמד למערכת, כולל שמירה בענן והצגת כתובת קבועה לתמונה לכל חיה.  
משמש להעלאת תמונות חדשות עבור חיות מחמד, אחסונן ב-Cloudinary ועדכון כתובת התמונה במסד הנתונים.

**שימוש ב-API עיקרי:**

* Cloudinary API שירות ענן לאחסון וקידוד תמונות.
* **ספריה:**  
  cloudinary של Node.js חיבור ל-Cloudinary API.
* **אימות:**  
  פרטי החיבור (cloud\_name, api\_key, api\_secret) נשמרים כמשתני סביבה בקובץ ‎.env.
* **שימוש:**  
  • העלאת תמונות חיות קבצים בפורמט/\* image))  
  • שמירה אוטומטית בענן ב folder -ייעודי (pet-images).   
   • קבלת URL קבוע ומאובטח לכל תמונה ועדכון המזהה במסד הנתונים.  
  • תומך בהעלאה מהירה דרך buffer (ללא שמירת קובץ זמנית בדיסק - חשוב עבור דיפלוי בענן).

1. **הגדרת החיבור ל-Cloudinary:**

// Configure Cloudinary

*cloudinary*.config({

  cloud\_name: process.env.CLOUDINARY\_CLOUD\_NAME,

  api\_key: process.env.CLOUDINARY\_API\_KEY,

  api\_secret: process.env.CLOUDINARY\_API\_SECRET

});

1. **העלאת תמונה בענן ועדכון כתובת התמונה לחיית מחמד:**

/\*\*

 \* POST /api/pets/upload-image

 \* Upload an image for a pet using Cloudinary

 \*/

petRouter.post("/upload-image", upload.single('image'), async (*req*: *Request*, *res*: *Response*) => {

  try {

    if (!*req*.file) {

      return *res*.status(400).send({ error: "No image file provided" });

    }

    const { petId } = *req*.body;

    if (!petId) {

      return *res*.status(400).send({ error: "Pet ID is required" });

    }

    // Upload to Cloudinary using buffer

    const uploadResult = await **new** *Promise*((*resolve*, *reject*) => {

      const uploadStream = *cloudinary*.*uploader*.upload\_stream(

        {

          folder: 'pet-images',

          public\_id: `pet-${petId}-${*Date*.now()}`,

          resource\_type: 'image'

        },

        (*error*: *any*, *result*: *any*) => {

          if (*error*) {

            reject(*error*);

          } else {

            resolve(*result*);

          }

        }

      );

      uploadStream.end(*req*.file!.buffer);

    });

    const imageUrl = (uploadResult as *any*).secure\_url;

    // Update the pet with the new image URL

    const updatedPet = await Pet.findByIdAndUpdate(

      petId,

      { imageUrl: imageUrl },

      { new: true }

    );

**קישור למסד נתונים (Mongo DB)**

* **סוג מסד נתונים ושימושו:**  
  המערכת מבוססת על מסד נתונים NoSQL מסוג **MongoDB**  בענן (MongoDB Atlas), שמאפשר שמירה ושליפה של נתונים מורכבים בצורה גמישה, מתאימה לאפליקציות מודרניות עם הרבה סוגי מידע שונים (חיות, תורים, טיפולים, צוות ועוד).
* **גישה למסד והגדרת מודלים:**  
  הגישה למסד נעשית באמצעות ספריית **Mongoose** שייכת ל-Node.js.  
  שמספקת מודלים ברורים, ולידציה (בדיקת תקינות), ושאילתות פשוטות ואסינכרוניות.  
  לדוגמה, כל אובייקט (User, Pet, Appointment) מוגדר כ-Schema בנפרד כקובץ עצמאי בתיקיית models.
* **ניהול החיבור עצמו:**  
  כל פעולת API (כגון קביעת תור, הצגת חיות, הרשמה) מחייבת חיבור למסד.  
  החיבור מנוהל דרך קובץ מרכזי **utils/dbConnection.ts** שמבטיח שיש תמיד חיבור פעיל לפני עיבוד הבקשה.  
  אם החיבור כבר פתוח (cached) לא יוצרים חיבור חדש , דבר שחוסך הרבה זמן וטעינת שרת, במיוחד בסביבות Serverless כמו Vercel.
* **Caching and Poolingחיסכון משאבים:**  
  הקוד שומר Promise (התחייבות להתחברות) ומצב החיבור במשתנה גלובלי.  
  כך, אם מגיעות הרבה בקשות במקביל לא נוצרים עשרות חיבורים חדשים (מה שעלול לקרוס או להאט את המערכת),  
  אלא משתמשים מחדש בחיבור קיים (Connection Pool).

// Global cache to store connection

let cached: *ConnectionCache* = (global as *any*).mongoose;

if (!cached) {

  cached = (global as *any*).mongoose = { conn: null, promise: null };

}

* **הגדרת מפתחות וסיסמאות:**  
  כל פרטי ההתחברות (URI , שם משתמש, סיסמה) נשמרים כ-environment variables (למשל ‎ MONGODB\_URI‎) בתוך קובץ ‎.env והם לא חשופים לעולם בקוד הגלוי רק בשרת.
* **שימוש ב- Middleware:**  
  בכל בקשה ל-API (כל route שמתחיל ב-/api ) מופעל ה-Middleware **ensureDbConnection**  שמוודא לפני כל פעולה שיש חיבור תקין.  
  אם לא מצליחים להתחבר - הלקוח יקבל תשובה עם סטטוס 500 ושגיאה ברורה.

/\*\*

 \* Middleware to ensure database connection before handling requests.

 \*/

export const ensureDbConnection = async (*req*: *any*, *res*: *any*, *next*: *any*) => {

  try {

    await connectToDatabase();

    next();

  } catch (error) {

    console.error('Database connection error:', error);

*res*.status(500).json({

      error: 'Database connection failed',

      message: error instanceof *Error* ? error.message : 'Unknown database error'

    });

  }

};

* **הגדרות התחברות אופטימליות:**  
  במימוש ההתחברות יש הגדרות מתקדמות כמו:
  + זמן מקסימלי לבחירת שרת (serverSelectionTimeoutMS) -מגדיר את הזמן המקסימלי (במילישניות) שמונגו יחפש שרת פנוי להתחברות לפני שיזרוק שגיאה.
  + Timeout לחיבור (socketTimeoutMS)- קובע את הזמן המקסימלי (במילישניות) שמותר להמתין לפעולה (כמו שליפה או כתיבה) לפני שהחיבור נסגר.
  + Pooling (maxPoolSize) - מגדיר את מספר החיבורים המקסימלי שיכול להיות פתוח במקביל (connection pooling).
  + ניסיונות כתיבה מחדש (retryWrites)- מאפשר ניסיון חוזר אוטומטי לפעולות כתיבה שנכשלו עקב תקלות רגעיות ברשת.  
    כל אלו מבטיחים שהחיבור יעיל, מאובטח ולא מבזבז משאבים.
* **טיפול בשגיאות ודיבאג:**  
  כל בעיית התחברות תיתפס ב-try/catch תירשם ביומן (console.error)   
  ותישלח תשובת שגיאה מסודרת למשתמש כך שאף פעולה לא תתבצע בלי גישה אמינה ל-DB.

**סביבות מיוחדות**

**צד שרת- סביבות וספריות מרכזיות:**

* **Express**  
  מסגרת שרת (Web Framework) קלה וגמישה ל-Node.js המאפשרת לבנות API מהירים, מאובטחים ומודולריים.  
  יתרונות:
  + קוד קצר, קריא וברור
  + קהילה ענקית והרבה דוקומנטציה
  + גמישות (אפשר לבנות כלREST API , ולהתאים לאפליקציות ענן/ שרתים/ serverless).
  + שילוב קל עם אמצעי אבטחה, לוגיקה עסקית ו-middleware.
  + מאפשרת תחזוקה ושדרוג מהירים של המערכת.  
    למה אצלנו? מאפשר להפריד בקלות בין נתיבים, לוגיקה עסקית ומודלים, ומאפשר גידול מהיר של מערכת המרפאה ללא תלות בספריות אחרות.
* **Mongoose**  
  ספריית ORM (Object-Relational Mapping) לעבודה עם MongoDB, מאפשרת להגדיר מודלים (Schemas) עם בדיקות תקינות, יחסים וקשרים בין ישויות, ולטפל בקלות בנתונים מורכבים.  
  יתרונות:
  + קל לנהל ולשנות מבני נתונים בזמן אמת
  + מאפשר עבודה אסינכרונית יעילה
  + כולל hooks מובנים ( before save, after updateוכו')  
    למה אצלנו? נותן שכבת אבסטרקציה נוחה לעבודה עם MongoDB בענן, גמיש להתרחבות והוספת פיצ'רים.
* **bcryptjs**  
  ספריה להצפנת סיסמאות (Hashing) כל סיסמה של משתמש נשמרת כ-Hash ולא כטקסט גלוי.  
  יתרונות:
  + מונע גניבת סיסמאות במידה ויש דליפת מידע
  + אלגוריתם חזק ומהיר  
    למה אצלנו? מבטיח עמידה בדרישות אבטחה מחמירות בשמירת פרטי משתמש.
* **jsonwebtoken**  
  ניהול הרשאות ואימות משתמשים בעזרת טוקן JWT כל משתמש מקבל טוקן חתום, שמצורף לכל בקשה רלוונטית ל-API.  
  יתרונות:
  + אימות מהיר וללא צורך בבדיקת סיסמה בכל פעולה
  + חלוקה קלה של הרשאות לפי סוג משתמש (user, secretary)
  + פתרון סטנדרטי למערכות מבוססות API  
    למה אצלנו? מאפשר זיהוי והרשאות בטוחות בין צד לקוח לשרת, מבלי להעביר פרטי התחברות בכל קריאה.
* **multer**  
  ספריה פשוטה ונפוצה להעלאת קבצים דרך ה-API (למשל, תמונות של חיות מחמד).  
  יתרונות:
  + טיפול יעיל בבקשות multipart (סוג מיוחד של בקשות HTTP שמשמש לשליחת טופס שמכיל גם קבצים)
  + שליטה על אחסון, סינון וסוגי קבצים  
    למה אצלנו? ניהול קל של קבצים, אחסון נפרד, והבטחה שקבצים מגיעים תקינים לשרת.
* **cors**  
  ספריה שמאפשרת שליטה על מדיניות הגישה ל-API מצד דפדפנים שונים (Cross-Origin Resource Sharing) .כלומר corsמאפשר להחליט מאילו דומיינים/אתרים מותר לגשת ל-API של השרת,  
  וכך למנוע גישה בלתי מורשית ממקורות לא ידועים או זדוניים.  
  יתרונות:
  + בקרה ואבטחה – רק אתרים/פרונט מורשים יכולים לגשת לשרת
  + עוזר להפריד בין סביבת פיתוח, טסטים ו-Production  
    למה אצלנו? מאפשר להריץ שרת (localhost) עם פרונט נפרד (Vite) מבלי לקבל שגיאות בדפדפן.
* **TypeScript**  
  שפת תכנות מבוססת JavaScript עם תמיכה מלאה בטיפוסים (types) ובדיקת קוד סטטית.  
  יתרונות:
  + גילוי באגים בזמן כתיבה, לא בהרצה
  + השלמה אוטומטית חכמה (Intellisense)
  + קוד קריא, מתועד, נוח להרחבה ולתחזוקה
  + מאפשר לעבוד בצוותים כל אחד יודע מה הפונקציה מקבלת ומחזירה  
    למה אצלנו? הפרויקט כולו כתוב ב-TypeScript מה שמקל על תחזוקה, שדרוג, והגנה על איכות קוד.

**צד לקוח- סביבות וספריות מרכזיות:**

* **React**  
  ספריית UI ל-Single Page Applications.  
  יתרונות:
  + ממשק דינמי, מהיר ונוח למשתמש
  + קומפוננטות – קוד ממוחזר, מופרד וברור
  + קהילה ענקית, הרבה תוספים וספריות עזר  
    למה אצלנו? מאפשר בניית מערכת דינמית (כמו צ'אט-בוט, טפסים חכמים, תצוגות שונות).
* **Tailwind CSS**  
  מסגרת CSS מודרנית בסגנון Utility-First לעיצוב מהיר וקל של רכיבים.  
  יתרונות:
  + אין צורך לכתוב קוד CSS "מלוכלך" כל עיצוב נעשה דרך מחלקות מוכנות
  + תומך בעיצוב רספונסיבי ונגיש
  + שומר על אחידות עיצובית בכל המערכת  
    למה אצלנו? חוסך זמן בעיצוב, מאפשר שינויים מיידיים, ויוצר ממשק משתמש עדכני ומרשים.

**ספריות עזר נוספות:**

* **axios -**קריאות HTTP בין לקוח לשרת ובין שירותים שונים.
* **lucide-react-**אייקונים ויזואליים.
* **react-router-dom** **-**ניווט בין עמודי האפליקציה.
* **react-calendar-**רכיב לוח שנה לטפסים/תורים.
* **eslint-**בדיקת איכות קוד.
* **exceljs, xlsx-**ייבוא/ייצוא נתונים לאקסל.
* **@tailwindcss/vite-**אינטגרציה קלה בין Tailwind ל-Vite.
* **@types/… -** תמיכה ב-TypeScript בספריות צד ג' .
* **cloudinary** **-** ספריית Node.js המאפשרת חיבור ל-Cloudinary API , העלאת תמונות חיות מחמד מהשרת לענן, וקבלת כתובת URL קבועה ומאובטחת עבור כל תמונה.

**קטעי קוד מיוחדים:**

1. **הגדרות CORS מתקדמות (server.ts)**

טיפול בהרשאות גישה (CORS) עם רשימת דומיינים מורשית והגדרות דינמיות לפי סביבה (פיתוח/ייצור).  
מקורות:

* + [Express.js CORS Documentation](https://expressjs.com/en/resources/middleware/cors.html)

1. **טיפול בבקשות Preflight ובכותרות ידניות (server.ts)**   
   טיפול ידני ב OPTIONS- כולל הוספת כותרות CORS, בהתבסס על דוקומנטציה של MDN ודוגמאות מ- Stack Overflow.  
   מקורות:
   * [MDN Web Docs – HTTP access control (CORS)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS)
   * [Stack Overflow – Express CORS preflight OPTIONS request handling](https://stackoverflow.com/questions/7067966/how-to-allow-cors-in-express-nodejs)
2. **הגדרת משתני סביבה וטעינת קונפיגורציה ((server.ts**  
   טעינת משתני סביבה עם dotenv, הגדרת פורט דינמית והגדרות שרת - דפוס שמופיע במדריכים רבים בענן ובפלטפורמות Deployment.  
   מקורות:
   * [Node.js Best Practices](https://github.com/goldbergyoni/nodebestpractices#2.-project-structure-practices)
   * [Express.js Documentation](https://expressjs.com/en/advanced/best-practice-performance.html#set-node_env-to-production)
   * [Heroku Config Vars](https://devcenter.heroku.com/articles/config-vars)
3. **Time Filtering Logic -AppointmentViewClient.tsx**  
   פונקציה להמרת שעות (12/24) למספר דקות מהחצות - בהשראת מדריכים ופתרונות .  
   מקורות:
   * [JavaScript – Working with Date and Time](https://www.freecodecamp.org/news/javascript-date-now-how-to-get-the-current-date-in-javascript/)
4. **קונפיגורציית Vite- (vite.config.ts)**  
   הגדרת פלאגינים ל-React ו-Tailwind על פי מדריכים רשמיים.  
   מקורות:
   * [Vite Documentation](https://vitejs.dev/config/)
   * [Tailwind CSS + Vite Guide](https://tailwindcss.com/docs/guides/vite)
5. **React Hooks Patterns & Express Route Structure**  
   שימוש ב-useEffect, useState, מבני נתיבים של Express ומודלים של Mongoose לפי Best Practices ודוקומנטציה רשמית.  
   מקורות:
   * [React Hooks – Official Documentation](https://react.dev/reference/react)
   * [Express.js Routing](https://expressjs.com/en/guide/routing.html)
   * [Mongoose Documentation – Models & Schemas](https://mongoosejs.com/docs/models.html)

**פרומפטים שבהם השתמשנו בכלי-AI Claude, Gemini ,ChatGPT, Copilot:**

במהלך הפיתוח השתמשנו בפרומפטים ייעודיים מול כלי AI כדי לקבל דוגמאות קוד, הסברים, או השראה לפתרון בעיות מורכבות ולמימוש לוגיקה חדשה. להלן מספר דוגמאות מרכזיות לפרומפטים:

* **פיתוח לוגיקת תור חירום**   
  ביקשנו מה-AI לנסח ולבנות לוגיקה המנהלת תורי חירום בצורה דינמית ומאובטחת, כולל זיהוי זמינות וטרינר, טיפול במצבים בהם אין רופא זמין, ותגובה מתאימה למשתמש.  
  דוגמת פרומפט:

"תעזור לי לכתוב לוגיקה בצד לקוח שמטפלת בקביעת תור חירום. המשתמש לוחץ על כפתור, יש לבדוק אילו תורים קיימים, האם יש רופא פנוי, ואם לא להציג הודעה מתאימה. הלוגיקה צריכה להיות ב-TypeScript ולכלול טיפול בשגיאות. אשמח לצרף לך קטעי קוד על מנת שתבין את ההקשרים."

* **ייעול עבודה עם React hooks והפרדת לוגיקה בקומפוננטות**  
  התייעצנו איך נכון להשתמש ב-useEffect כדי לבצע קריאה ל-API כשמשתנה מסוים משתנה, איך להימנע מלולאות אינסופיות, ומה הדרך הכי נכונה לכתוב את הלולאה בריאקט.  
  דוגמת פרומפט:

"איך נכון להשתמש ב-useEffect כך שיבוצע רק כאשר הערך X משתנה, ולמנוע הפעלה אינסופית? תן דוגמה של קוד ב-TypeScript".

* **שיפור פונקציות עזר (Utility) לעבודה עם תאריכים ומחרוזות**  
  בנינו פונקציה להמרת מחרוזת תאריך לפורמט Date בצורה אמינה וללא שגיאות.  
  דוגמת פרומפט:

"איך אפשר להמיר מחרוזת תאריך בפורמט DD/MM/YYYY לאובייקט Date ב-JavaScript בצורה בטוחה?"

* **השראה לעיצוב תפריטים, ממשקים ואופן טיפול במצבי קצה**  
  נעזרנו בפרומפטים לקבלת רעיונות להצגת תפריטים אינטראקטיביים, טיפול במצבים של אין נתונים להצגה, והצעות לארגון קוד ברכיבים מודולריים.

**משתמשים קיימים לטובת התחברות לאתר:**

**שם משתמש:**matantal606@gmail.com  **סיסמה:**123456  **סוג משתמש:** לקוח

**שם משתמש:**talyagudin@gmail.com  **סיסמה:**123456  **סוג משתמש:** מזכירה

**תיק למשתמש**

**מבוא**

האתר **FurEver Friends** מציע ממשק אינטרנטי נוח, ברור ונגיש לניהול מרפאה וטרינרית. המערכת מיועדת לבעלי חיות מחמד ולמזכירת המרפאה, ומאפשרת לכל צד לבצע פעולות רלוונטיות בצורה פשוטה ויעילה כגון הזמנת תורים, צפייה במידע רפואי ועדכון טיפולים.

האתר כולל את המאפיינים הבאים:

* **עיצוב מודרני ונעים לעין**

שימוש בצבעי פסטל רכים ליצירת תחושה מזמינה ונעימה, לצד מבנה מסודר וברור של התוכן.

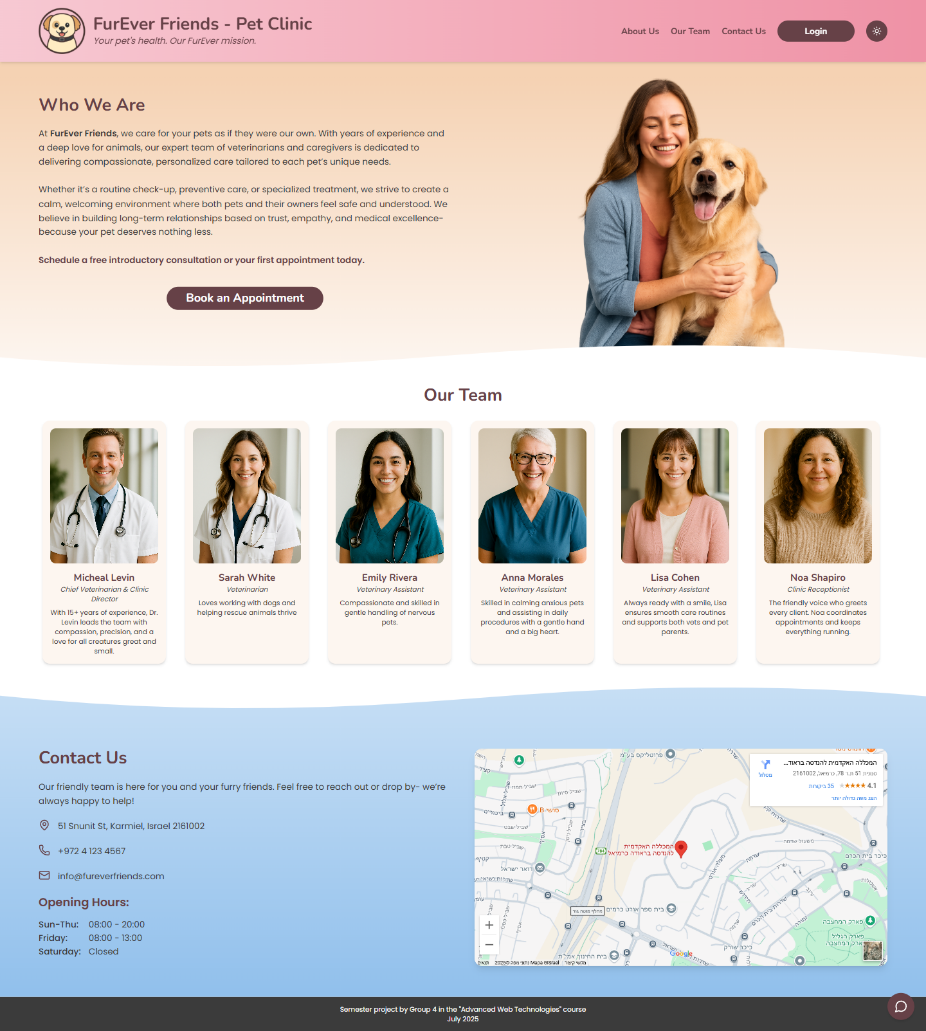
* **תמיכה מלאה במצב כהה (Dark Mode)**

התאמה אוטומטית או ידנית למצב כהה, לנוחות שימוש מרבית בתנאי תאורה משתנים.

* **רספונסיביות מלאה**

התאמה מלאה למגוון מכשירים: מחשבים, טאבלטים וסמארטפונים- לצורך גישה נוחה מכל מקום ובכל זמן.

**דף הבית**

דף הבית מהווה את עמוד הכניסה הראשי למערכת ומספק מידע תמציתי על המרפאה, הצוות, מיקום ושעות פעילות.

**רכיבי מפתח:**

* **סרגל ניווט (Navbar)**

ממוקם בחלק העליון של הדף ונשאר קבוע בעת גלילה או מעבר בין דפים. הסרגל כולל את האפשרויות הבאות:

* לחיצה על לוגו האתר מחזירה לדף הבית.
* כפתורים (About Us, Our Team, Contact Us) המבצעים גלילה אוטומטית לסקשן הרלוונטי בעמוד.
* כפתור Login מאפשר התחברות למשתמשים רשומים (לאחר התחברות מוחלף בכפתור Logout).
* כפתור החלפת מצב תצוגה מאפשר מעבר בין מצב בהיר, כהה או לפי ההגדרה הקיימת בדפדפן.
* לאחר התחברות נוסף כפתור Dashboard המנווט לדשבורד הרלוונטי למשתמש המחובר.
* **כפתור “Book an Appointment”**

מאפשר הזמנת תור בהתאם לסוג המשתמש:

* משתמש שאינו מחובר: מועבר לעמוד ההתחברות.
* לקוח מחובר: מועבר להזמנת תור חדש.
* מזכירת המרפאה: מועברת להזמנת תור חדש מממשק הניהול.
* **סקשן “Who We Are”**

מציג מידע כללי אודות המרפאה, עקרונות השירות והגישה המקצועית של הצוות.

* **סקשן “Our Team”**

מציג את צוות המרפאה הכולל וטרינרים, סייעות ומנהלת משרד, כולל תיאור קצר של תפקידיהם וניסיונם.

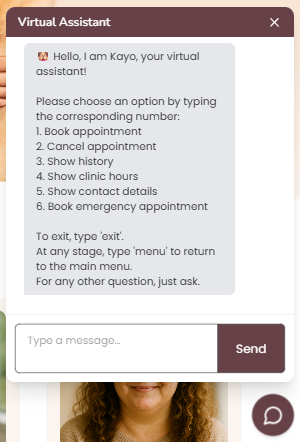
* **סקשן “Contact Us”**

כולל כתובת מלאה, פרטי יצירת קשר, שעות פעילות ומפת ניווט אינטראקטיבית של מיקום המרפאה.

* **כפתור צ’אטבוט**

מופיע תמיד בפינה הימנית התחתונה של המסך. מאפשר פתיחת חלון צ’אט אינטראקטיבי עם עוזר וירטואלי (ללא מעבר עמוד).

**העוזר הווירטואלי (צ'אטבוט)**

בפינה הימנית התחתונה של המסך מופיע כפתור לפתיחת חלון שיחה עם Kayo, העוזר הווירטואלי של המרפאה.

הצ'אטבוט מאפשר ניהול שיח ידידותי ויעיל לביצוע פעולות מגוונות, ללא צורך בגלישה לעמודים נוספים. ניתן לבחור פעולה על ידי הקלדת מספר מתאים מתוך התפריט הראשי:

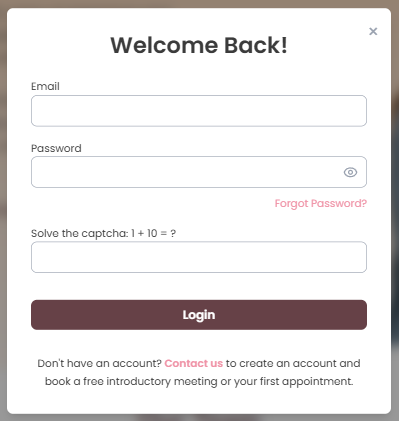
1. הזמנת תור
2. ביטול תור
3. צפייה בהיסטוריית טיפולים
4. צפייה בשעות פעילות
5. צפייה בפרטי יצירת קשר
6. הזמנת תור חירום

exit - ליציאה מהשיחה

menu - לחזרה לתפריט הראשי

העוזר הווירטואלי גם מסוגל להשיב על שאלות כלליות, במיוחד כאלה הקשורות לפעילות המרפאה, השירותים הניתנים, שעות הפעילות וחברי הצוות.

**התחברות למערכת**

מסך ההתחברות מאפשר גישה למערכת עבור משתמשים קיימים- לקוחות או מזכירת המרפאה.

המשתמש מתבקש להזין כתובת דוא"ל, סיסמה, ולפתור שאלה פשוטה מסוג CAPTCHA לצורך אימות בסיסי.

אפשרויות נוספות במסך ההתחברות:

* שכחתי סיסמה- קישור לשחזור סיסמה (מפורט להלן)
* צור חשבון- הפנייה לעמוד יצירת קשר לצורך פתיחת חשבון חדש והזמנת פגישת היכרות ראשונית

**תהליך שחזור סיסמה**

משתמשים ששכחו את סיסמתם יכולים לבצע שחזור באופן עצמאי דרך המסך הייעודי.

התהליך כולל שלושה שלבים:

תמונה שמכילה טקסט, צילום מסך, גופן, מספר

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

1. **הזנת כתובת דוא"ל**

המשתמש מתבקש להזין את כתובת הדוא"ל הרשומה במערכת. לאחר מכן, נשלח אליו קוד אימות בן 6 ספרות למייל.

1. תמונה שמכילה טקסט, צילום מסך, גופן, דף אינטרנט

   תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**הזנת קוד אימות**

המשתמש מזין את הקוד שהתקבל בדוא"ל לצורך אימות זהותו. הקוד תקף ל-15 דקות בלבד.

תמונה שמכילה טקסט, צילום מסך, מספר, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

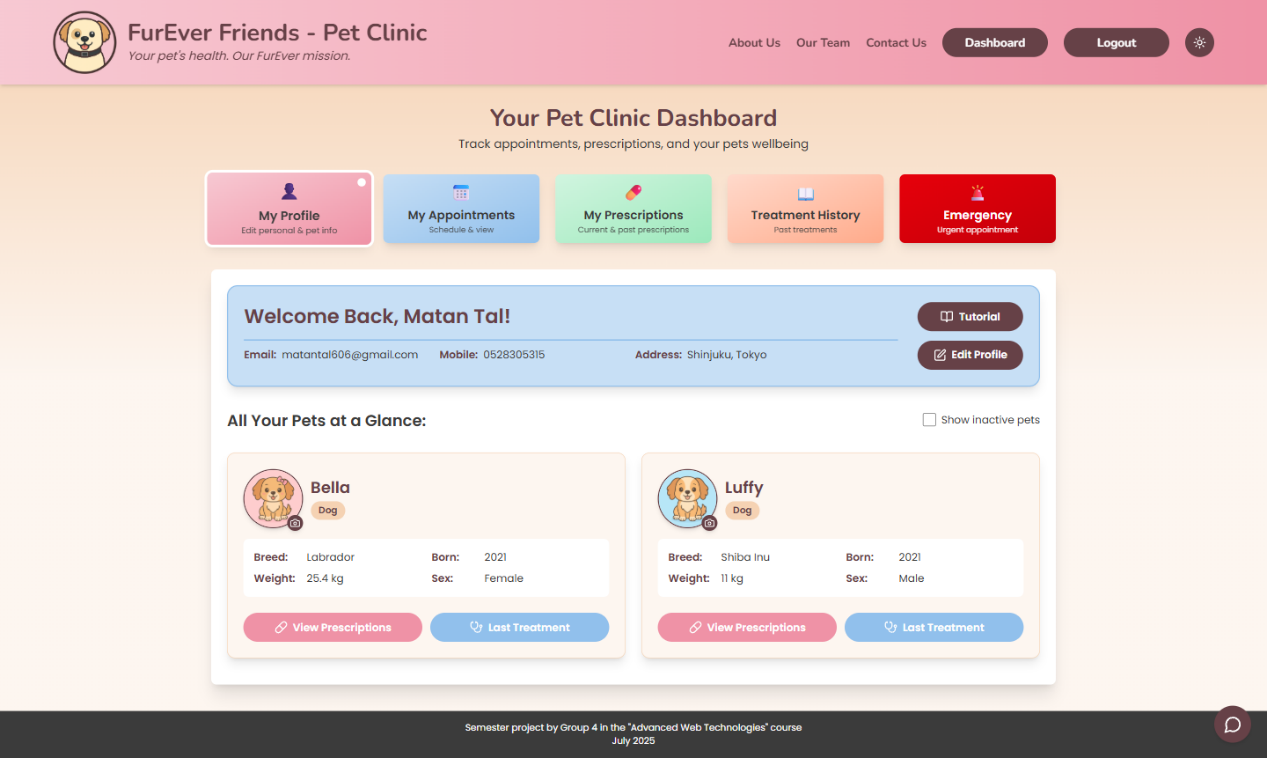
1. **תמונה שמכילה טקסט, צילום מסך, גופן, מספר

   תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.הגדרת סיסמה חדשה**

לאחר אימות הקוד, ניתן להזין סיסמה חדשה ולאשר אותה.

**ממשק הלקוח**

**דשבורד אישי**

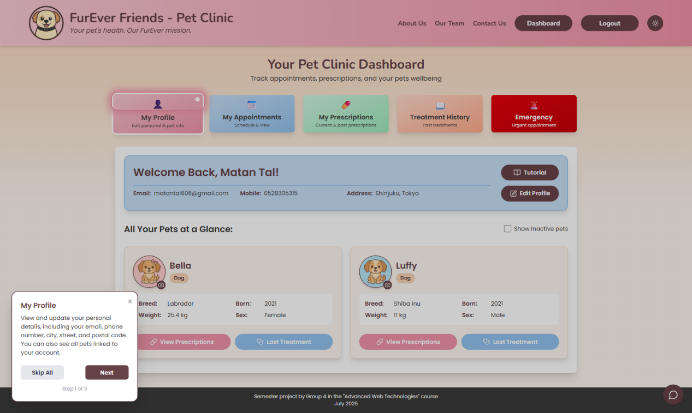


לאחר ההתחברות, מועבר המשתמש לדשבורד האישי, המרכז את כלל המידע הרפואי והתפעולי עבורו ועבור חיות המחמד שברשותו.

**מבנה הדשבורד:**

* **תפריט פעולות ראשיות**

תצוגה צבעונית וברורה של כפתורים המובילים לפעולות הנפוצות (כל פעולה תתועד בנפרד בהמשך תיק המשתמש):

* פרופיל אישי
* ניהול תורים
* מרשמים
* היסטוריית טיפולים
* הזמנת טיפול חירום
* **מדריך אינטראקטיבי (Tutorial)**

מוצג אוטומטית עם התחברות ראשונה ומסביר על שימוש בפונקציות השונות באתר. ניתן להפעילו מחדש בכל שלב דרך לחיצה על כפתור "Tutorial".

המדריך מתמקד בכל רכיב בתורו, תוך הדגשה ויזואלית (הילה) וניסוח ברור של מטרתו.

* תמונה שמכילה טקסט, צילום מסך, מספר, גופן

  תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**כרטיס משתמש אישי**

מוצג בחלק העליון של המסך. לחיצה על Edit Profile תפתח טופס עריכה שבו ניתן לעדכן פרטים אישיים ולשנות סיסמה.

תמונה שמכילה טקסט, חשמל, צילום מסך, תוכנה

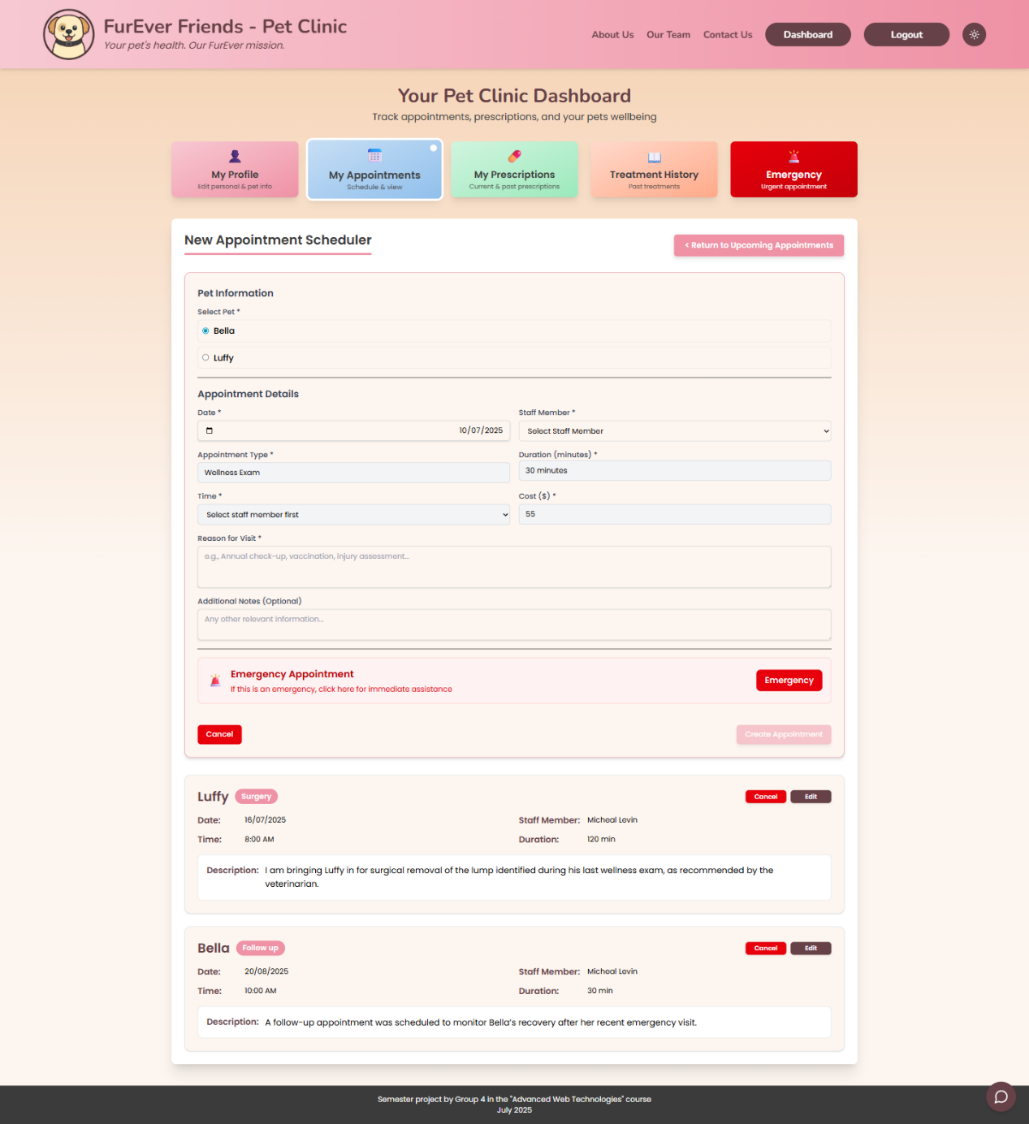
תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

* **רשימת חיות המחמד**

הצגת כלל חיות המחמד הרשומות למשתמש, כולל:

* תמונה מותאמת אישית או תמונת ברירת מחדל
* נתונים בסיסיים: גזע, שנת לידה, מין, משקל
* צפייה במרשמים, צפייה בטיפול האחרון
* ניתן לסמן תיבה לצפייה גם בחיות שאינן פעילות

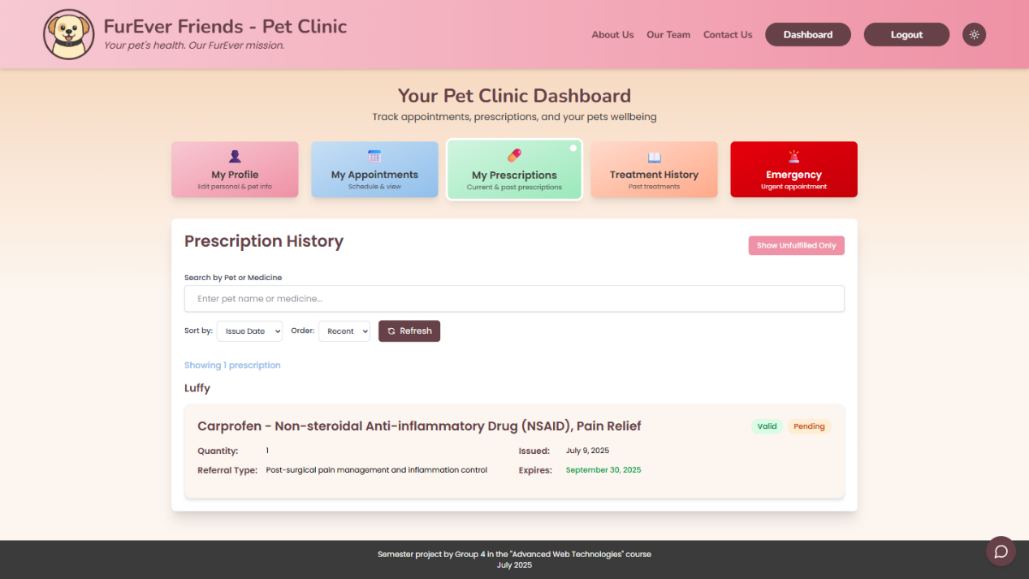
**הפגישות שלי**



ממשק זה מאפשר למשתמש לקבוע פגישה חדשה, וכן לצפות בתורים קיימים, לערוך או לבטל אותם.

* הזמנת תור חדש כוללת בחירת חיית מחמד, סוג פגישה, איש צוות, תאריך, שעה, משך ועלות.
* קיימת אפשרות להזמין תור חירום בלחיצה על כפתור ייעודי.
* רשימת תורים עתידיים מוצגת מתחת לטופס ההזמנה- עבור כל תור ניתן לצפות בפרטים, לערוך או לבטל.

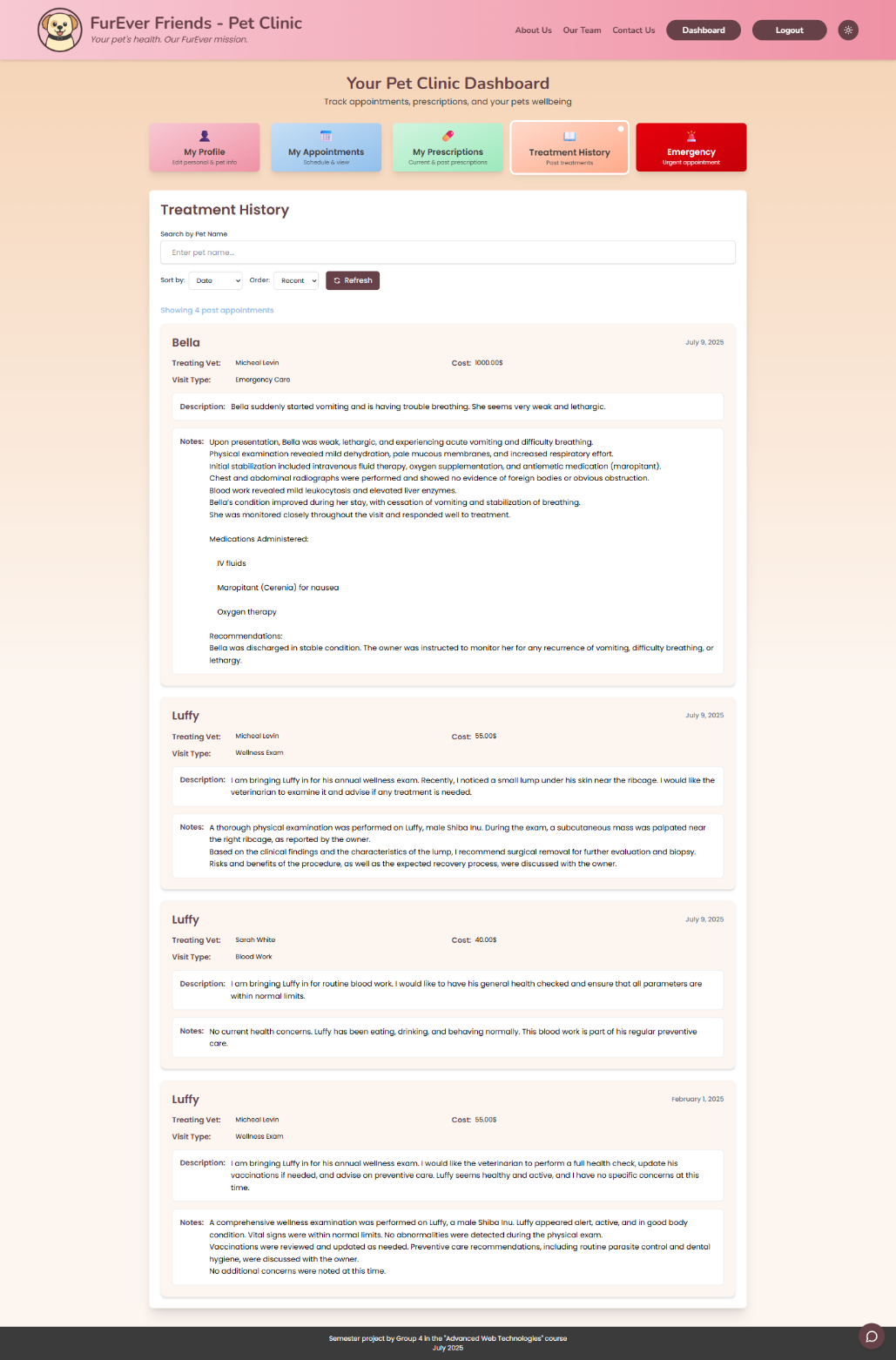
**המרשמים שלי**



ממשק זה מאפשר צפייה וניהול של כל המרשמים שנרשמו עבור חיות המחמד של המשתמש.

* חיפוש לפי שם חיה או שם תרופה
* סינון להצגת מרשמים שטרם מומשו בלבד (Unfulfilled Only)
* מיון לפי: תאריך הנפקה / תאריך תפוגה / שם החיה / שם התרופה
* סידור לפי: מהחדש לישן או להפך

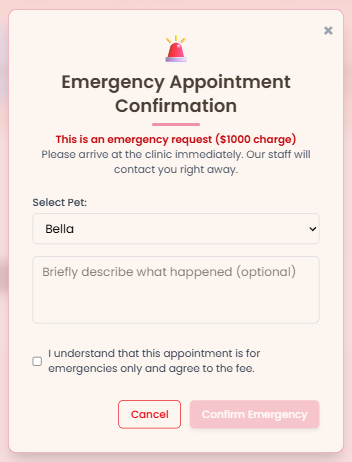
**היסטוריית טיפולים**



ממשק זה מציג את כל הטיפולים שבוצעו עבור חיות המחמד של המשתמש.

* חיפוש לפי שם חיה
* מיון לפי: תאריך טיפול / שם החיה / מחיר
* סידור לפי: מהחדש לישן או להפך

**תור חירום**



טופס ייעודי להזמנת פגישה דחופה במרפאה, המיועדת למקרים רפואיים דחופים בלבד.

* יש לבחור את חיית המחמד הרלוונטית.
* ניתן לתאר בקצרה את הבעיה (שדה רשות).
* יש לאשר כי המשתמש מבין שמדובר בתור חירום בלבד ומסכים לעלות הקבועה (1000 ₪).

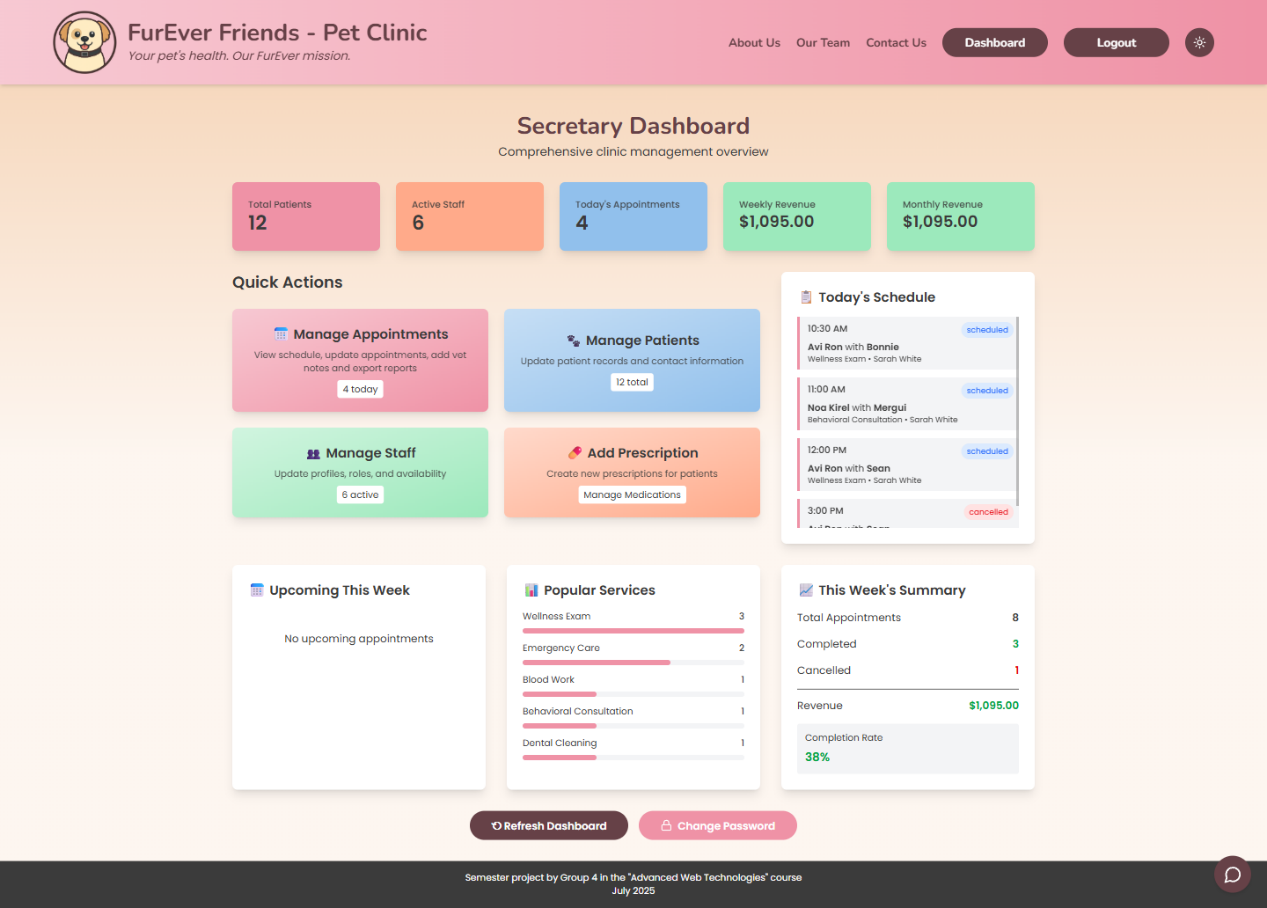
**לוגיקת פעולה (מאחורי הקלעים):**

בעת שליחת הבקשה, המערכת פועלת לפי השלבים הבאים:

1. בדיקה האם קיים וטרינר זמין מיידית.
2. אם לא- המערכת מאתרת את הווטרינר עם כמות התורים הנמוכה ביותר.
3. אם יש תורים ניתנים לביטול (שאינם ניתוחים או תורי חירום אחרים)- הם יבוטלו אוטומטית בטווח של שעתיים מהשעה הנוכחית, כדי לפנות מקום לתור החירום.

**ממשק המזכירה**

**דשבורד המזכירה**

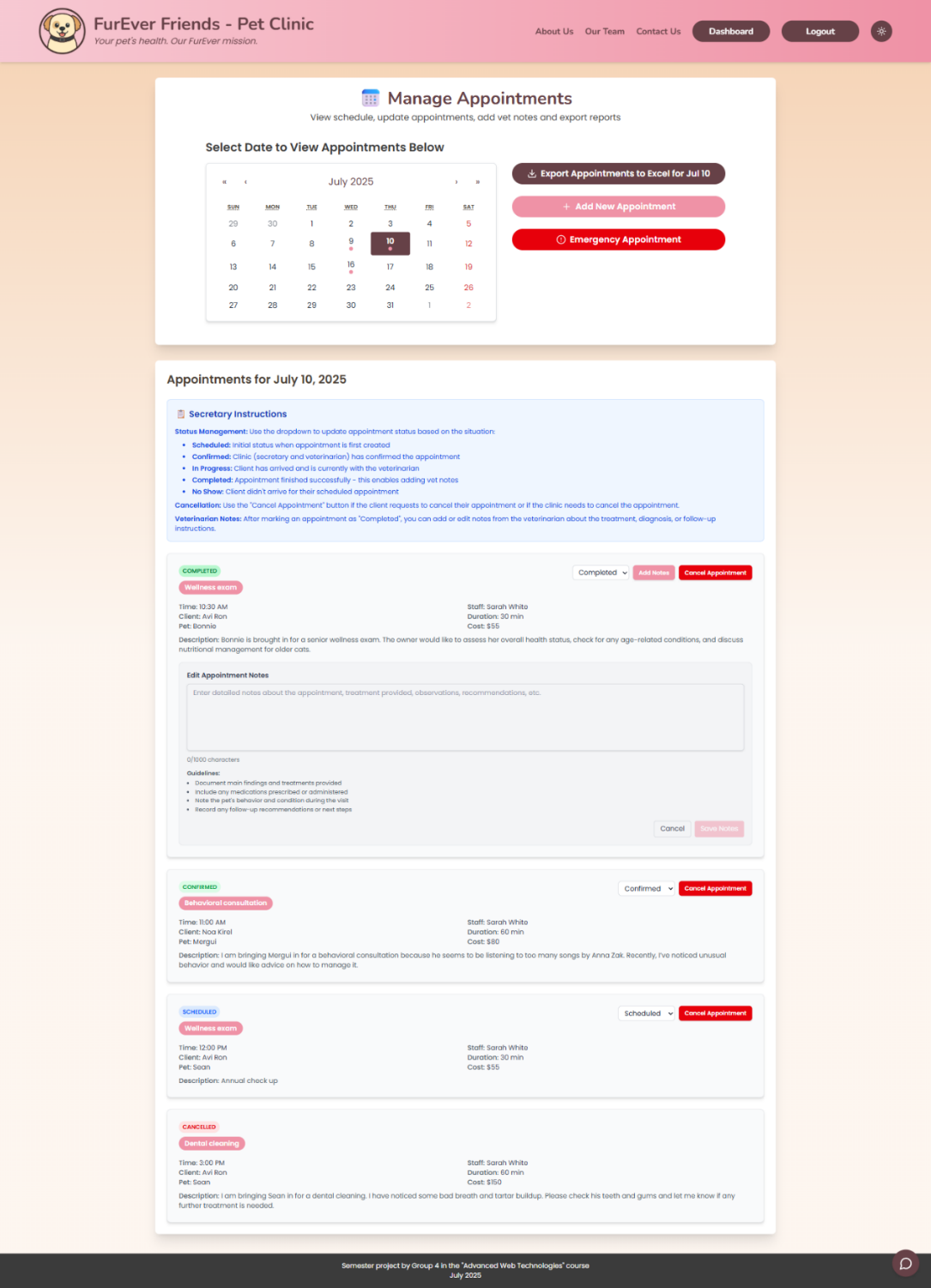


מסך הניהול המרכזי של מזכירת המרפאה, הכולל נתונים בזמן אמת, פעולות תפעוליות, ותצוגת לו"ז.

**תוכן הדשבורד:**

* נתונים סטטיסטיים
* תפריט פעולות מהירות:
* ניהול תורים
* ניהול מטופלים
* ניהול צוות
* הוספת מרשמים
* לו"ז יומי
* לוז שבועי עתידי
* טיפולים נפוצים
* סיכום שבועי
* אפשרות לשינוי סיסמה

**ניהול תורים**



ממשק זה מאפשר למזכירת המרפאה לנהל את כלל התורים הקיימים לפי תאריך, לבצע עדכונים, להוסיף פגישות חדשות ולייצא דוחות.

**יכולות הממשק:**

* בחירת תאריך מתוך לוח שנה לצפייה בתורים היומיים.
* פעולות לביצוע:
* הוספת תור חדש
* פתיחת תור חירום
* ייצוא תורים לאקסל ליום הנבחר
* ביטול תור קיים
* הזנת הערות וטרינר לאחר השלמת תור, כולל ממצאים, טיפול שבוצע והמלצות להמשך
* ניהול סטטוס תורים- ניתן לעדכן את מצב הפגישה בהתאם להתקדמות:
* Scheduled - נוצר
* Confirmed - אושר
* In Progress - מתבצע
* Completed - הושלם
* No Show - המטופל לא הגיע

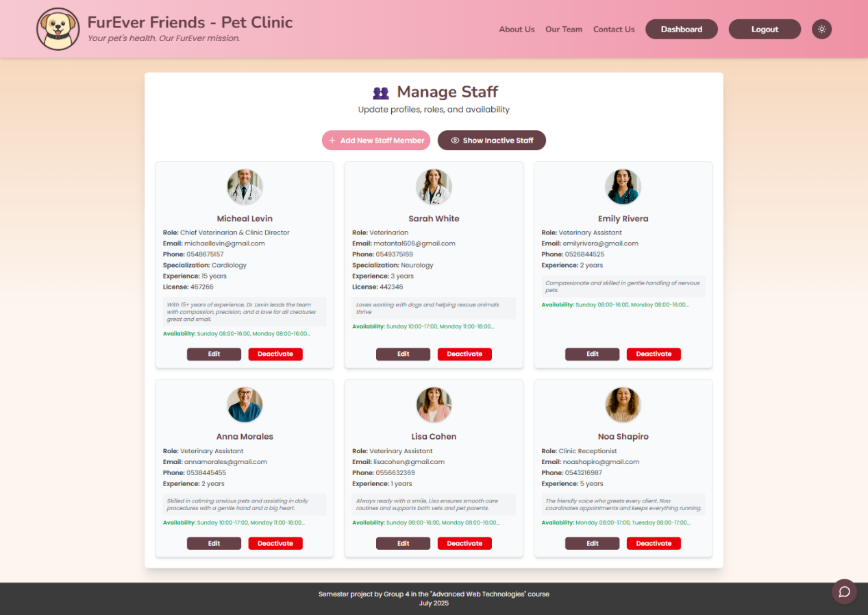
**ניהול מטופלים (לקוחות)**

ממשק זה מאפשר למזכירת המרפאה לנהל את כלל הלקוחות- בעלי חיות המחמד.

**פעולות מרכזיות:**

* חיפוש לקוחות לפי שם בעלים, פרטי קשר או שם חיית מחמד.
* הוספת מטופל חדש באמצעות טופס ייעודי
* שדות חובה מסומנים בכוכבית
* בעת ההוספה מוקצית סיסמה ראשונית קבועה: password
* בהתחברות הראשונה יתבקש המשתמש להגדיר סיסמה חדשה, כחלק ממנגנון אבטחה
* ניהול משתמשים קיימים: עבור כל לקוח ניתן לבצע:
* עריכת פרטים אישיים
* הוספת חיית מחמד חדשה
* עריכת פרטי חיה קיימת
* צפייה/הסתרה של חיות שאינן פעילות

**ניהול צוות**

ממשק זה מאפשר למזכירת המרפאה לנהל את אנשי הצוות הפעילים במערכת: וטרינרים, סייעות, מנהלה ועוד.

**יכולות עיקריות:**

* צפייה באנשי צוות קיימים
* עריכה של פרטי אנשי צוות קיימים
* הפסקת פעילות של איש צוות (Deactivation)
* הצגת אנשי צוות לא פעילים

תמונה שמכילה טקסט, חשמל, צילום מסך, תוכנה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.בנוסף: הוספת איש צוות חדש בטופס ייעודי הכולל את מילוי הפרטים הבאים:

* פרטים אישיים ופרטי קשר
* תפקיד (Role), תחום התמחות, שנות ניסיון, מספר רישיון
* תיאור חופשי
* העלאת תמונה
* בחירת ימי ושעות עבודה

**הוספת מרשם**

ממשק זה מאפשר למזכירת המרפאה ליצור, לנהל ולעדכן מרשמים רפואיים עבור חיות המחמד של הלקוחות בהתאם להנחיות הווטרינר.

**מבנה הממשק:**

* חלק עליון- נתונים ומעקב
* סטטיסטיקת מרשמים: סך הכול, בוצעו, ממתינים, שפג תוקפם
* התראות למרשמים שפג תוקפם תוך 15 ימים
* הוספת מרשם חדש ע"י:
* חיפוש ובחירת מטופל
* בחירת חיית המחמד מתוך הרשומות הקיימות
* הזנת פרטי המרשם: שם התרופה, סוג, סיבת ההפניה, כמות ותוקף
* חיפוש וסינון מרשמים קיימים
* חיפוש לפי שם תרופה, חיה או מטופל
* סינון לפי סטטוס (All, Fulfilled, Pending)
* סינון לפי תוקף (Active / Expired)
* צפייה וניהול מרשמים קיימים
* עדכון סטטוס (סומן כמומש / לא מומש)
* מחיקת מרשם לפי הצורך