**טכנולוגיות - WEBהגשה סופית**

**Group 10 - B14 Fitness Challenge App – WebFitPro**

**מגישים:**

|  |  |  |
| --- | --- | --- |
| **מס'** | **שם** | **ת.ז** |
| 1 | עידו סקל | 316139351 |
| 2 | אופק אידלוביץ' | 318307196 |
| 3 | שובל בן שושן | 207502428 |
| 4 | נועה סיוון | 209090034 |
| 5 | עדן קנטור | 316138403 |

**תקציר פרויקט**  
**תיאור הפרויקט שקיבלנו**:

\*\*Fitness Challenge App\*\*: Create a fitness application that allows users to participate in and track their progress in various fitness challenges. Include social features to encourage community engagement.

**למה האפליקציה שימושית ולמי היא מתאימה?**

האפליקציה מיועדת למתאמנים שמעוניינים להגדיר מטרות אישיות ולעקוב אחר ההתקדמות שלהם, מספקת חוויה נוחה ופשוטה לשימוש ובנוסף מאפשרת שיתוף מידע בקהילה לגבי אתגרי הכושר שבמערכת.

**פונקציונאלית מרכזית באפליקציה:**

* מעקב: המשתמשים יכולים לעקוב אחרי ההתקדמות שלהם באפליקציה לפי כמות האתגרים שהשלימו מתוך אלה שהם ראו.
* שיפור אישי וחוויית משתמש נוחה: הממשק הפשוט מאפשר גם למשתמשים ללא ידע טכנולוגי להרגיש בנוח ולהתחיל להשתמש באפליקציה תוך דקות.
* התאמה אישית :האפליקציה תומכת במצבי תצוגה שונים – מצב כהה ומצב בהיר, לבחירת המשתמש ותזכור את העדפות שלו לפעם הבאה.
* שיתוף מידע בין משתמשים: האפליקציה מאפשרת למשתמשים מהקהילה לשתף מידע על איזה אתגרים הם אהבו/ממליצים לבצע, והמידע ישותף למשתמשי אפליקציה אחרים.
* הצגת אתגרים שונים לפי סינון – המשתמשים יכולים להסתכל על עוד אתגרים מעבר למה שהאפליקציה סיפקה להם באותו יום תוך סינון ומיון לפי אזור גוף ורמת קושי.
* שיפור אישי**:** האפליקציה מחשבת את מדד ה BMI-של המשתמש לפי הנתונים שלו ומציגה ציטוטי מוטיבציה בעמוד הבית כדי לשמור על המוטיבציה.
* ניהול מערכת משתמשים פשוט: משתמשים בעלי הרשאות אדמין יוכלו בקלות לאשר, לערוך ולמחוק במידת הצורך משתמשים שנרשמו וכאלה שרשומים לאפליקציה.

**קישורים לקוד ולאתר**

|  |  |
| --- | --- |
| **Link (Ctrl and press it)** | **Name** |
| [**https://web-fit-pro-front-kappa.vercel.app/**](https://web-fit-pro-front-kappa.vercel.app/) | **Vercel – האפליקציה בקישור הזה** |
| [**https://github.com/EdenKantor/WebFitPro\_front**](https://github.com/EdenKantor/WebFitPro_front) | **GitHub Repo Front \*You can find the docx in here** |
| [**https://github.com/EdenKantor/WebFitPro\_back**](https://github.com/EdenKantor/WebFitPro_back) | **GitHub Repo Back** |

**תוכן עניינים**

1. תיאור עבודה, ומהנדס מערכת
2. דרישות
3. ארכיטקטורת אתר
4. Use Case
5. מבנה הקבצים
6. Components
7. פריטי מידע: מבנה DB
8. תיק מתכנת
9. תיק משתמש

**1) תיאור עבודה, ומהנדס מערכת**

**מהנדס מערכת** – אופק אידלוביץ'

אופן חלוקת העבודה - במהלך תכנון הפרויקט, חלוקת העבודה בין חברי הצוות נעשתה תוך התמקדות בכישוריהם האישיים ובתחומי האחריות שלהם.  
המשימות חולקו בצורה כזו שכל חבר צוות יכול למקד את הזמן והמאמצים בתחום בו הוא חזק ביותר, על מנת להבטיח את הצלחת הפרויקט בשיתוף פעולה פורה ויעיל.

ממשק בין חברי הצוות - במהלך הסמסטר, הצוות שלנו קיים תקשורת ישירה באמצעות פגישות ב-ZOOM, קבוצת WhatsApp ועדכון התוצרים דרך הGitHub המשותף. התמקדנו בחלק הזה בלהמשיך וליישם את האפליקציה הכמעט גמורה לקראת הגשתה הסופית. לקחנו את המשוב שקיבלנו מהתרגילי בית הראשונים ומהצגת הפרויקט בפני המרצה והתמקדנו בלתקן את החלקים בקוד ואת טפסי הפרויקט עליהם נטען כי דרוש שיפור. חילקנו את המשימות בין חברי הצוות. כל חבר צוות תכנן חלק מסוים והעלה את העדכונים שלו, ולאחר מכן כולנו התעדכנו בשינויים והעדכונים שבוצעו. חלוקת המשימות בצורה מסודרת הבטיחה שלכל חבר בצוות יהיה תחום אחריות ברור. שיתוף הפעולה המתמשך והתקשורת האפקטיבית בין כל חברי הצוות הביאו להשלמה מוצלחת של כל המשימות שהוקצו בשלב הנוכחי.

|  |  |  |
| --- | --- | --- |
| **שם חבר הצוות** | **משימות שהוקצו** | **משימות שהושלמו** |
| אופק | 1) המרה והעברת הקבצים מהחלק הראשון מHTML לPreact ויצירת הFrontend למערכת.  2) מידול הקומפוננטות בהם נשתמש במערכת  **3) מימוש פונקציונאליות הלקוח במערכת  4) בדיקה ותיקון רספונסיביות איפה שנדרש באפליקציה**  **5) הטמעת** Tailwind **באפליקציה בחלקים השונים של העיצוב בקוד לשיפור המראה ב**dark mode **ושיפור המראה הכללי.** | **1,2,3,4,5** |
| עידו | 1) יצירת הDB ויצירת הטבלאות בהם המערכת תשתמש עבור קבלת ועידכון מידע.  2) Backend – חיבור למסד נתונים ויצירת שאילתות לטבלאות בהם נשתמש.  3) Backend – יצירת API עבור כל עמוד של האפליקציה המאפשר קבלה, עדכון ומחיקת מידע במידת הצורך  4) מימוש הפונקצינליות של הלקוח עם הסרטוני אימון שלו בcontinue ובdiscover | **1,2,3,4** |
| שובל | 1) תרגום הפרויקט לדרישות הפונקציונאליות והלא פונקציונאליות שלו  2) יצירת תיק משתמש המפרט את כל התהליכים שקיימים באפליקציה  3) יצירת Use Case  4) לאתר ולדווח על באגים שנמצאו בעת הרצת האפליקציה ובפעילות השוטפת בה | **1,2,3,4** |
| עדן | 1) ניהול הGITHUB טיפול בבקשות pull וpush ושמירה על תקינות הפרויקט  2) עזרה במימוש הapi בbackend  3) מימוש הפונקציונאליות עבור פעולות למשתמש בעל הרשאות אדמין  4) יצירת קומפוננטות עבור עמודי האדמין | **1,2,3,4** |
| נועה | 1) יצירת הארכיטקטורה לאפליקציה  2) עזרה ביצירת הDB, עידכונו לערכים אמיתיים ופירוט התכונות השונות של כל טבלה  3) עזרה ביצירת עמודים המיעודים לפעולות האדמין השונים  4) לאתר ולדווח על אי רספונסיביות של דפים ו/או עמודים שלא עוברים בדיקות קצה | **1,2,3,4** |

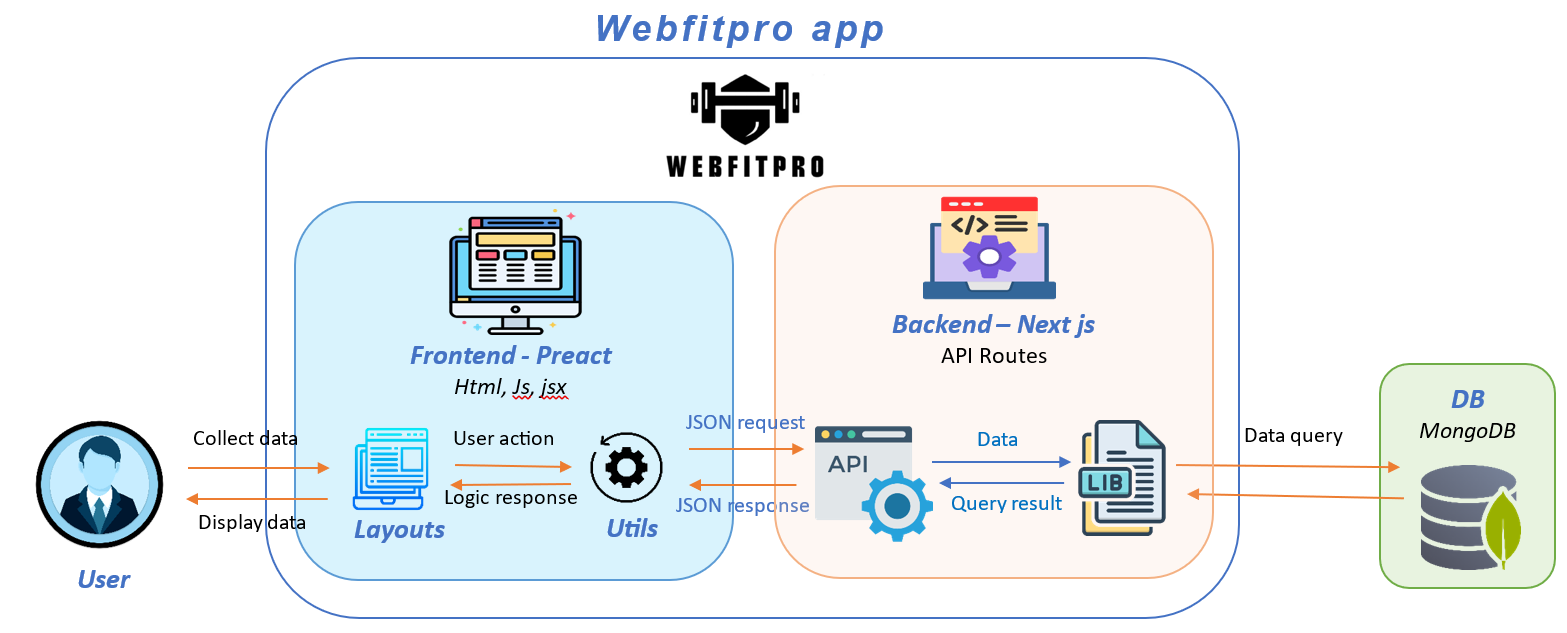
**2) דרישות**

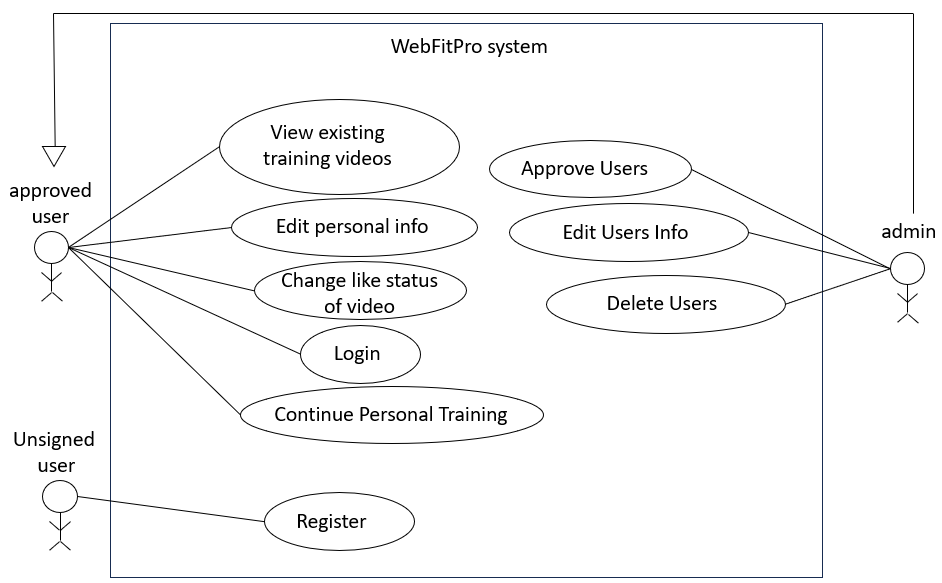
**דרישות פונקציונאליות**

|  |  |
| --- | --- |
| **Classification** | **Description** |
| FR | The system will allow users track their progress. |
| FR | The system will allow the user to view challenges beyond its daily recommendations. |
| FR | The system will allow filtering challenges by the body part targeted by the exercise. |
| FR | The system will allow sorting challenges by difficulty level. |
| FR | The system will allow sorting challenges alphabetically by their name. |
| FR | The system will allow sorting challenges by the number of likes the video has received. |
| FR | The system will support multiple different themes |
| FR | The system will provide different difficulty options for each challenge. |
| FR | The system will provide user registration |
| FR | The system will store personal user info: age, height and weight in the database. |
| FR | The system will allow registered users to log in. |
| FR | The system will calculate the user's BMI. |
| FR | The system will allow the approval of pending users. |
| FR | The system will allow to update non pending user account details. |
| FR | The system will allow to delete non pending user account. |
| FR | The system will allow admin-privileged users to sort the list of users. |
| FR | The system will allow navigation between pages. |
| FR | The system will allow users to update personal info after registration. |
| FR | The system will allow users to mark the challenges they liked. |
| FR | The system will store which challenges the user has marked as liked. |
| FR | The system will allow users to mark a challenge as completed. |
| FR | The system will allow the display of motivational quotes on the home screen. |

**דרישות לא פונקציונאליות**

|  |  |  |
| --- | --- | --- |
| **Classification** | **Description** | **Type by Wikipedia** |
| NFR | The system will store the user's progress by tracking the number of sessions opened by the user. | Reporting |
| NFR | The system will store the user's progress by tracking the number of sessions completed by the user. | Reporting |
| NFR | The system will provide an intuitive and user-friendly interface. | Usability (human factors) by target user community |
| NFR | Challenge selection will be displayed as a list. | Usability (human factors) by target user community |
| NFR | The challenges will be presented as clickable YouTube video thumbnails. | Usability (human factors) by target user community |
| NFR | Exercise names, difficulty levels, like counter, and targeted body part will be displayed. | Usability (human factors) by target user community |
| NFR | Themes will be light mode and dark mode | Usability (human factors) by target user community |
| NFR | In the registration process, users will fill in personal details: gender, age, weight, height, username, password. | Usability (human factors) by target user community |
| NFR | The BMI will calculate based on user's info: height, weight. | Functionality |
| NFR | An admin-privileged user will be able to approve pending newly registered users. | Management |
| NFR | An admin-privileged user will be able to update user's info and delete existing user accounts. | Management |
| NFR | When an admin approves, updates or deletes user accounts, they will be able to sort the list by username or by registration date. | Usability (human factors) by target user community |
| NFR | The personal info that can be updated: height, weight, and age. | Modifiability |
| NFR | The user will mark the challenges they liked by clicking heart symbol. | Usability (human factors) by target user community |
| NFR | The completion will be indicated by clicking "Mark as Done". | Usability (human factors) by target user community |
| NFR | The session will consist of 3 exercises that the user must complete. | Usability (human factors) by target user community |
| NFR | The completion of the session is conditional on performing all 3 exercises and clicking "Submit." | Usability (human factors) by target user community |
| NFR | The system will be optimized for different screen size to ensure a responsive experience. | Usability (human factors) by target user community |
| NFR | The system will allow multiple users to be active at the same time. | Scalability |

**3) ארכיטקטורת אתר**

**4) Use Case**

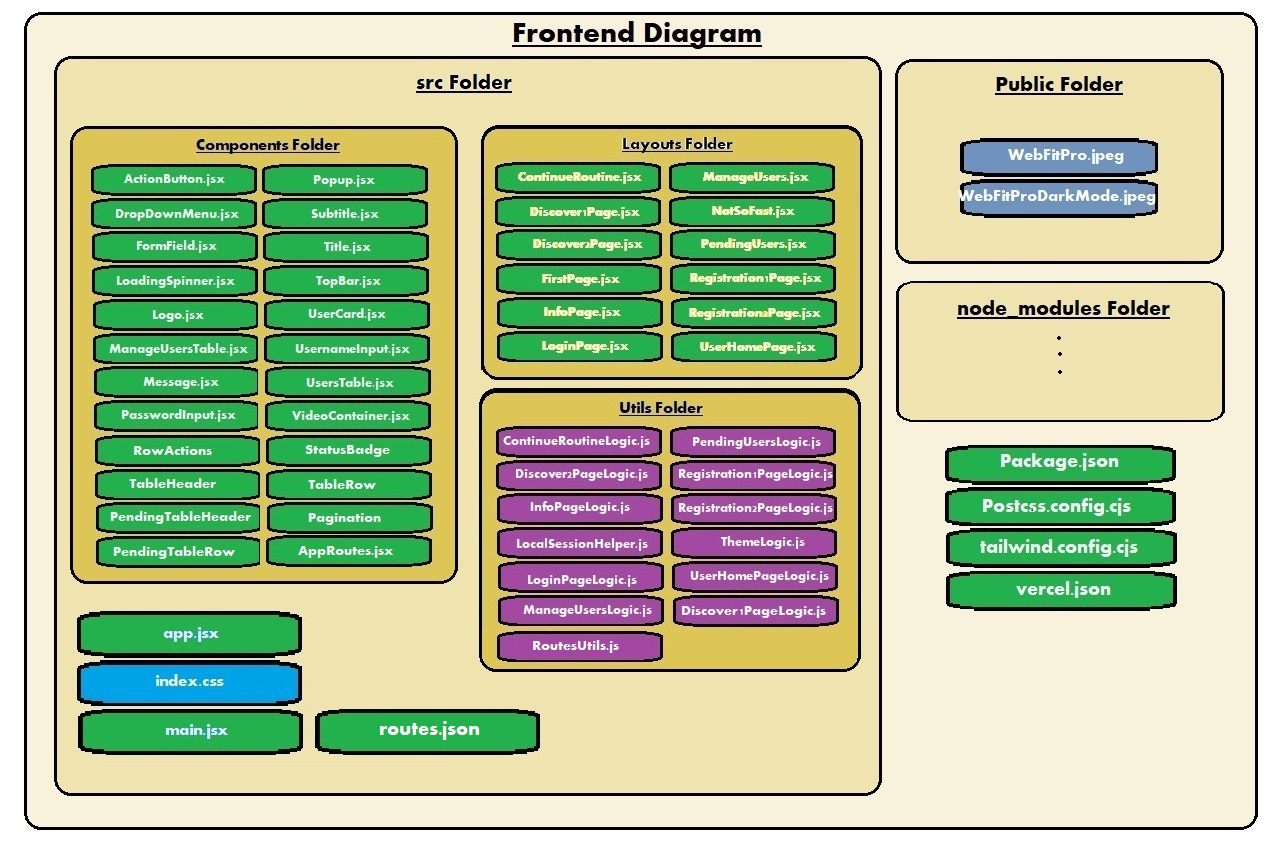
**פירוט הפעולות**  
כל **משתמש** באתר יוכל לבצע את הפעולות הבאות:

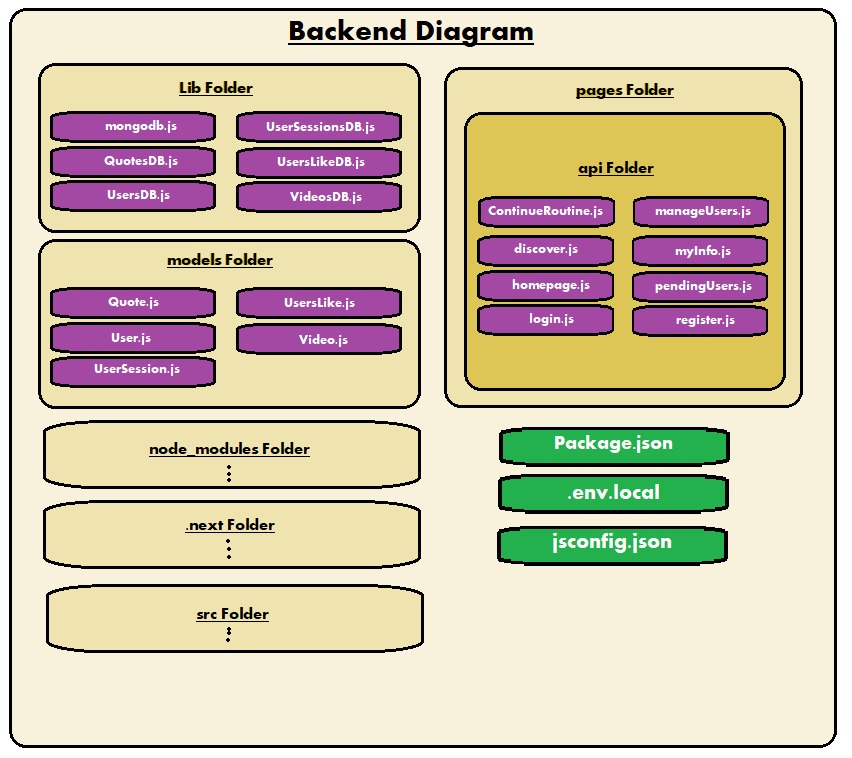
1. התחברות לאתר ע"פ שם משתמש וסיסמא
2. לערוך את פרטיו האישיים
3. לצפות בסרטוני וידיאו של אימונים שונים אשר קיימים במערכת
4. לסמן לייק על סרטון שאהב, ולהוריד לייק מסרטון
5. להמשיך באימון האישי שהותאם עבורו.

בנוסף, במקרה של **admin** הוא יוכל גם לבצע את הפעולות הבאות:

1. הadmin יאשר משתמשים חדשים שנרשמו לאתר.
2. הadmin יוכל לנהל משתמשים במערכת ולערוך את פרטיהם.
3. הadmin יוכל למחוק משתמשים שרשומים למערכת.

**משתמש לא רשום** יוכל להירשם לאתר.

**5) מבנה קבצים**

****

**6) Components**

* **Components Directory Overview:**  
  The components directory serves as a centralized location for reusable UI elements that can be imported and used across different pages and features of your application. These components are built with Preact and utilize Tailwind CSS for styling, supporting both light and dark modes.

**His files content and their functions:**

* + **ActionButton.jsx**   
    This component creates a versatile button element that appears throughout the application.

Key Parameters:

* + - label - The text displayed on the button
    - iconClass - Optional icon class for visual enhancement
    - onClick - The function to execute when clicked
    - className - Additional custom styling options
  + **DropdownMenu.jsx**   
    A reusable dropdown selection component that handles user choice from a list of options.

Key Parameters:

* + - options - Array of available choices
    - selected - Currently selected option
    - onSelect - Function that handles the selection change
  + **FormField.jsx**   
    A standardized form input component that maintains consistency across all forms in the application.

Key Parameters:

* + - label - Text label for the input field
    - placeholder - Hint text for the input
    - value - Current input value
    - onChange - Function to handle value changes
    - type - Input type (text, number, password, etc.)
  + **LoadingSpinner.jsx**   
    A full-screen loading indicator component that provides visual feedback during asynchronous operations. It creates a semi-transparent overlay with an animated spinner to indicate loading states.
  + **Logo.jsx**   
    A dynamic logo component that automatically switches between light and dark mode versions of the application logo. It listens for theme changes and updates accordingly, maintaining visual consistency with the current theme setting.
  + **ManageUsersTable.jsx**   
    A complex component that handles the display and management of user data in both desktop and mobile views.

Key Parameters:

* + - Responsive layout (table for desktop, cards for mobile)
    - Inline editing capabilities
    - Pagination for mobile view
    - User deletion functionality
    - users - Array of user data to display
    - onUpdate - Function to handle user data updates
    - onDeleteClick - Function to handle user deletion
  + **Message.jsx**   
    A dynamic feedback component that enhances user experience by providing visual feedback for system operations. It automatically determines the message type and applies appropriate styling, making it ideal for form submissions, action confirmations, and error notifications.

Key Parameters:

* + - message - The text content to be displayed with automatic success/error detection
  + **Pagination.jsx**   
    A navigation interface specifically designed for mobile card views, enabling smooth transitions between items while maintaining user context. It provides visual feedback for navigation limits and current position.

Key Parameters:

* + - currentIndex - Current position in the item list
    - totalItems - Total number of items available
    - onNext - Function to handle forward navigation
    - onPrev - Function to handle backward navigation
  + **PasswordInput.jsx**   
    An enhanced security input component that combines user convenience with security best practices. It includes a visibility toggle feature and comprehensive styling support for both light and dark themes.

Key Parameters:

* + - label - Descriptive text for the password field
    - placeholder - Helper text for the input field
    - onChange - Function to handle input changes
    - toggleVisibility - Function to toggle password visibility
    - isPasswordVisible - Current visibility state
  + **PendingTableHeader.jsx**   
    A structural component that defines the column layout for the pending users management table. It provides consistent header styling and organization for pending user data display.
  + **PendingTableRow.jsx**   
    A row component specifically designed for displaying pending user information and approval controls. It handles status display and approval action for each user entry.  
    Key Parameters:
    - user - User object containing profile data and registration status
    - onApprove - Handler function for user approval action
    - Internal status logic determines: Active/Pending display state, Appropriate status styling, Action button visibility.
  + **Popup.jsx**   
    A versatile modal component that handles multiple interaction scenarios including confirmations, success messages, and error displays. It includes backdrop management and responsive positioning.  
    Key Parameters:
    - isOpen - Controls popup visibility
    - onClose - Function to handle popup dismissal
    - onConfirm - Function to handle positive action
    - backToHome - Navigation function for success state
    - isError - Flag for error state styling
  + **RowActions.jsx**   
    A specialized action group component for managing user data operations. It handles different states including active, pending, and modified states with appropriate visual feedback.

Key Parameters:

* + - isActive - Indicates if the row is in active state
    - isPending - Indicates if the row is in pending state
    - hasChanges - Indicates if the row has unsaved changes
    - onUpdate - Function to handle update action
    - onDelete - Function to handle delete action
  + **StatusBadge.jsx**   
    A compact visual indicator that represents user states through color-coded badges, enhancing quick status recognition in user interfaces.

Key Parameters:

* + - isActive - Boolean that determines the badge state and styling
  + **Subtitle.jsx**   
    A typography component that maintains consistent secondary heading styling across the application, with built-in support for theme modes

Key Parameters:

* + - text - The subtitle content to be displayed
  + **TableHeader.jsx**   
    A structured header component for data tables that defines column layouts and maintains consistent styling. It's specifically designed for user management interfaces.
  + **TableRow.jsx**   
    A complex data row component that handles user data display and interaction. It manages data editing, status display, and row-level actions with appropriate state handling.

Key Parameters:

* + - user - User data object to display
    - onInputChange - Function to handle field changes
    - onUpdate - Function to process updates
    - onDeleteClick - Function to handle deletion
  + **Title.jsx**   
    A primary heading component that ensures consistent main title styling across the application, incorporating responsive design and theme support.

Key Parameters:

* + - text - The main title content to be displayed
    - className - Optional additional styling classes
  + **TopBar.jsx**   
    A global navigation component that provides consistent top-level navigation and theme control across the application. It dynamically adjusts its behavior and appearance based on the current route.

Key Parameters:

* + - showBackButton - Visibility of navigation button
    - buttonText - Context-specific button label
    - buttonIcon - Context-specific button icon
    - backPath - Navigation destination
  + **UserCard.jsx**   
    A versatile user information display component that supports both viewing and editing modes. It provides a compact, mobile-friendly interface for user data management.  
    Key Parameters:
    - user - User data object containing profile information
    - editData - Modified user data pending submission
    - hideDefaultActions - Toggle for standard action buttons
    - onInputChange - Handler for field modifications
  + **UsernameInput.jsx**   
    A specialized input component for username entry that maintains consistent styling and behavior across the application.  
    Key Parameters:
    - label - Field label text
    - placeholder - Input field placeholder text
    - value - Current input value
    - onChange - Handler for input changes
  + **UsersTable.jsx**   
    A responsive user management component that provides both desktop table and mobile card views for pending user administration.

Key Parameters:

* + - users - Array of user objects to display
    - onApprove - Handler for user approval actions
    - Internal states manage: Current card index for mobile view, Pagination controls, View mode switching.
  + **VideoContainer.jsx**   
    A comprehensive video display component that handles video playback, user interactions, and dynamic state management. It provides a complete interface for video content with like and completion tracking features.

Key Parameters:

* + - id - Unique identifier for the video
    - title - Display title for the video
    - videoUrl - Source URL for video playback
    - category - Video classification category
    - level - Difficulty/complexity level
    - numOfLikes - Initial like count
    - done - Initial completion status
    - liked - Initial like status
    - onLike - Handler for like/unlike actions
    - onDone - Handler for completion status updates
  + **AppRoutes.jsx**   
    Manages application routing while implementing animated page transitions for a smooth user experience. The component retrieves route configurations dynamically using getRoutes from RouteUtils.js and maps them to the Switch component for path-specific rendering. It integrates react-transition-group to apply CSS-based fade animations with a 500ms timeout, ensuring polished and seamless transitions between pages.

Key Parameters:

* + - wouter - library for declarative routing.
    - Integrates react-transition-group for fade-in/out animations during page transitions.

**7) פירוט פריטי מידע**

**מידע על הDB ועל החיבור אליו**

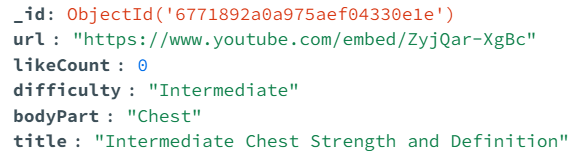
בחרנו להשתמש בDB של **MongoDB Atlas** שנמצא על שרת ענן ולא לוקאלית.  
החיבור למסד נתונים נעשה באמצעות הקובץ mongodb.js שנמצא בפרויקט הbackend שלנו תחת תיקיית lib ובעצם יצרנו חיבור באמצעות הדרייברים של monogdb ותיקשרנו מול הcluster של האפליקציה, ודרכו כמובן לכל הcollections הנחוצים לתהליך כלשהו.  
החיבור נעשה על ידי ניסיון חיבור ל **MONGO\_URI**שנמצא בקובץ **.env.local** שבתיקיית הפרויקט backend שלנו, כאשר כל פעם שנקרא לחיבור הוא יבדוק שאין כבר אחד קיים, שבמקרה כזה הוא יחזיר את החיבור הקיים ולא ייצור אחד חדש.  
תהליך הבקשה מה frontלback וחזרה: בקשות httpמגיעות מה frontend לapi ומשם קובץ הapi יפעיל פונקציות ששייכות לקובץ הרלוונטי לcollections שנמצא בתיקיית pages/api, לאחר השגת החיבור לdb והשגת החיבור לcollections הרלוונטים הקובץ יכין לנו שאילתה ויחזיר את התשובה לapi ומשם תשובה לfront שביצעו את הבקשה. **מבנה כללי של כל הdb שלנו ומבנה JSON סטנדרטי בכל טבלה**

WebFitPro DB Collections

Users



Videos



UsersLike

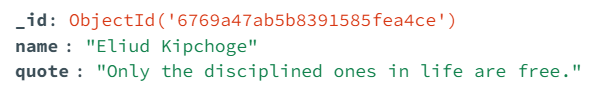


UserSessions

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

Quotes



**הסבר על כל טבלה ותכונותיה**

1. **Users**בטבלה זו נשמור את פרטי המשתמשים השונים שבמערכת וגם תכונות שונות חשובות עבור התחברות, הרשמה וחישוב BMI:
   1. פרטי התחברות: userName, password
   2. נתונים פיזיים: age, gender, weight ,height
   3. הרשאות (אם אדמין או לא): isAdmin
   4. האם כבר אושר על ידי האדמין: isRegistered
2. **Videos**בטבלה זו נשמור את תכונות האתגרים השונים שלנו וכמובן את הסרטונים שמייצגים את האתגר, כמה משתמשים במערכת אהבו את האתגר, ותכונות שונות:
   1. url של סרטון היוטיוב שמייצג לנו את האתגר
   2. כמות הלייקים שהאתגר קיבל ממשתמשים במערכת likeCount
   3. רמת קושי של האתגר, יש לנו 3 רמות שונות מנמוך לגבוה:  
       Beginner, Intimidate, Advanced
   4. אזור בגוף בו האתגר מתמקד לאמן
   5. שם האתגר, בעצם השם של הסרטון מיוטיוב
3. **UsersLike**טבלה מקשרת בין טבלת Videos לבין טבלת Users, כאשר לכל משתמש יש Json שמפרט במערך של url לאיזה סרטונים המשתמש עשה להם לייק
4. **UserSessions**טבלה שמטרתה לשמור לכל user את פרטי הסשן אימון הנוכחי שלו שנראה בContinue Routine הטבלה מכילה את התכונות הבאות:
   1. שם המשתמש
   2. מערך בגודל 3 שמכיל URLs שמייצגים את 3 האתגרים שעל המשתמש להשלים
   3. מערך עזר בגודל 3 שמטרתו לשמור לנו כמה אתגרים מתוך ה3 המשתמש השלים, מיקום האיברים הוא בהתאמה למיקום האתגרים במערך הurl
   4. כמות של הסשנים שהמשתמש פתח עד כה, כל פעם שהוא השלים את כל האתגרים ונכנס חזרה לעמוד המונה עולה ב1
   5. כמות הסשנים שמשתמש השלים עד כה, כל פעם שהמשתמש סיים סשן של 3 אתגרים המונה עולה ב1
   6. משתנה בינארי עזר שמטרתו להעיד אם משתמש כבר סיים את הסשן או שהוא צריך להמשיך אותו, נשתמש בו לטעינת העמוד כדי לדעת אם להתאים למשתמש 3 אתגרים חדשים או להראות לו את הישנים שהוא עוד לא סיים את כולם
5. **Quotes**טבלה שמתוכה נגריל לעמוד בית משפטי מוטיבציה למשתמש שלקוחים מציטוטים מפורסמים, נשמור בה את הציטוט ואת שם המצוטט/ת

**8) תיק מתכנת**

**Table of Contents**

1. Project Overview
2. Technology Used
3. Dependencies
4. Installation and Setup
5. Running the Project
6. Backend important files and functionality
7. Frontend important files and functionality
8. API calls in the application
9. Prompt used with AI

**1) Project Overview**

This application helps users participate in fitness challenges, track their progress, and stay motivated. It fosters community engagement by allowing users to share recommendations and discover new challenges while offering a simple and personalized user experience.

**2) Technology Used**

**Frontend**

* **Preact** for UI framework
* **Wouter** for lightweight routing
* **Tailwind CSS** for styling
* **React Transition Group** for animations
* **Vite** for development and build

**Backend**

* **Next.js** for server-side rendering and routing
* **MongoDB** for database management

**3) Dependencies**

**Frontend:**

* preact
* wouter
* vite

**Backend:**

* next
* mongodb

**4) Installation and Setup**

This guide provides step-by-step instructions to set up the WebFitPro project, which consists of two parts:

* Frontend: Built with Preact (Vite)
* Backend: Built with Next.js

Both parts should be placed under a main folder named webfitpro, with the following structure:

**webfitpro/**  
**│── frontend/** # Preact (Vite) project  
**│── backend/** # Next.js project

**Prerequisites:**

Before starting, ensure that you have the following installed:

* Node.js and npm (Check by running:  **node -v** and **npm -v**)
* Git (Check by running:  **git --version**)

**1. Clone the Repositories:**

Open a terminal and run the following commands to clone both repositories into the correct structure:

**1.1 Clone the Frontend Repository:**

git clone https://github.com/EdenKantor/WebFitPro\_front.git webfitpro/frontend

**1.2 Clone the Backend Repository:**

git clone https://github.com/EdenKantor/WebFitPro\_back.git webfitpro/backend

After running these commands, your project structure should look like this:

**webfitpro/  
│── frontend/   
│── backend/**

**2. Install Dependencies:**

**2.1 Install Frontend Dependencies:**

Navigate to the frontend folder and install the required dependencies:

cd webfitpro/frontend

npm install

**2.2 Install Backend Dependencies:**

Navigate to the backend folder and install the required dependencies:

cd ../backend

npm install

**3. Set Up Environment Variables for Backend:**

The backend requires a **MongoDB connection string** to function properly.

1. Inside the backend directory, create a new file named: .env.local

2. Open the file and add the following line (replace your-mongodb-uri with your actual MongoDB connection string):

MONGODB\_URI=your-mongodb-uri

**5)** **Running the Project**

**1. Start the Backend Server:**

Navigate to the backend directory and start the server:

cd webfitpro/backend

npm run dev

By default, the backend will run on **http://localhost:3000**.

**2. Start the Frontend Server:**

Open a new terminal window, navigate to the frontend directory, and start the frontend server:

cd webfitpro/frontend

npm run dev

By default, the frontend will run on **http://localhost:5173**.

**6) Backend Files & Functionality**

**Folder Structure files and logic**

* **Folder lib**/  
  Responsible for managing communication with MongoDB and establishing the connection. The directory also contains files named exactly like the collections, where each file is designed to handle queries and responses for the collection with the corresponding name.

**His files content and their functions:**

* + **mongodb.js**   
    manages the connection to a MongoDB database

Main functions:

* + - **connectToDatabase** : Connects to MongoDB, reuses existing connections if available, and handles errors during connection attempts.
  + **QuotesDB.js**  
    handles operations related to the Quotes collection in the database.  
    Main functions:
    - **getRandomQuote** : Retrieves a random quote from the database.
  + **UsersDB.js**  
    handles operations related to the Users collection in the database.  
    Main functions:
    - **getUserByUsername** : Retrieves a user by their username.
    - **addNewUser** : Adds a new user to the database.
    - **updateUserDetails** : Updates age, height, and weight for a specific user.
    - **deleteUser** : Deletes a user by their username.
    - **registerUser** : Marks a user registered status after approval.
    - **getPendingUsers** : Retrieves users who are not yet registered.
    - **getUsers** : Retrieves users who are not yet registered.
  + **UserSessionsDB.js**  
    handles operations related to user sessions in the database.  
    Main functions:
    - **createUserSession** : Creates a new user session with default values.
    - **getUserSessionByUserName** : Retrieves a specific user session
    - **updateVideos** : Updates the array of videos for a user session.
    - **updateFinishedStatus** : Updates the finished status of a user session.
    - **updateChecks** : Updates a that user completed specific challenge in the session
    - **updateAllChecks** : Resets the done array for a user session to default values.
    - **incrementCompleteSessions** : Increases the completed sessions count by 1 for a user session.
    - **incrementOpenedSessions** : Increases the opened sessions count by 1 for a user session.
    - **deleteFromUserSessions**  : Deletes a user session from the UserSessions collection.
  + **UsersLikeDB.js**  
    Handles operations for URLs associated with specific users and their likes on exercises.  
    Main functions:
    - **getURLsByUser** : Fetches all URLs associated with a specific user, representing the exercises that the user liked.
    - **checkURLForUser** : Checks if the given user has liked a specific exercise.
    - **addURLForUser** : Adds a new exercise to the user's "like" array.
    - **removeURLForUser** : Removes an exercise from the user's "like" array, indicating that the user has disliked it.
    - **createNewUserLike** : Creates a new document in the UsersLike collection for a user.
    - **deleteFromUsersLike** : Deletes a user from the UsersLike collection.
  + **VideosDB.js**  
    Handles operations related to managing videos in the database, including retrieval, sorting, and modification of video details.  
    Main functions:
    - **changeLikeCount:** Increases or decreases the like count of a video by 1.
    - **addNewVideo:** Adds a new video entry with data of difficulty, body part, and title.
    - **deleteVideo:** Removes a video from the database by its URL.
    - **checkVideoByUrl**: Checks if a video exists in the database by its URL and retrieves its details.
    - **getUniqueVideos**: Fetches three unique videos, one from each difficulty level (Beginner, Intermediate, Advanced).
    - **getVideosSortedByTitle**: Retrieves videos for a specific body part, sorted alphabetically by title.
    - **getVideosSortedByLikeCount**: Retrieves videos for a specific body part, sorted by like count in ascending or descending order.
    - **getVideosSortedByDifficulty**: Retrieves videos for a specific body part, ordered by difficulty level from Beginner to Advanced or vice versa.
    - **getVideosByBodyPart**: Retrieves all videos related to a specific body part.
* **Folder pages/api/**Contains API route handlers. Each layout page on the frontend that needs to retrieve, edit, or delete data from the database communicates with its corresponding API route file, which shares the same name. Detailed information about all API calls can be found later in this document under Section 8: 'API Calls in the Application.  
  **His files content:**
  + **continueRoutine.js**
  + **discover.js**
  + **homepage.js**
  + **login.js**
  + **manageUsers.js**
  + **myInfo.js**
  + **pendingUsers.js**
  + **register.js**
* **Folder models/**

folder is intended to contain Mongoose models for defining MongoDB collections. Each file in this folder represents a schema for a specific entity in the database, allowing structured interaction with the database using Mongoose.

Each file corresponds to a collection in my MongoDB cluster.  
**His files content and their functions:**

* + **User.js :**This file defines the Mongoose schema for the Users collection, which stores user-related data, including login credentials, personal details, and authorization status.  
    his data:
    - **userName** (String): The unique username for user authentication.
    - **password** (String):The user's password for login.
    - **age** (Number):The user's age**.**
    - **gender** (String): The user's gender (e.g., Male, Female, Other).
    - **weight** (Number): The user's weight**.**
    - **height** (Number): The user's height**.**
    - **isAdmin** (String): Indicates whether the user has admin privileges.
    - **isRegistered** (String): Marks if the user has been approved by an admin.
  + **Video.js :**   
    This file defines the Mongoose schema for the Videos collection, which stores information about different challenge videos, including their details, difficulty level, and user interactions.

his data:

* + - **url** (String): The YouTube link representing the challenge video.
    - **likeCount** (Number): The number of likes received from users.
    - **difficulty** (String): The challenge difficulty level (Beginner, Intermediate, Advanced).
    - **bodyPart** (String): The body part targeted by the challenge.
    - **title (String):** The name of the challenge, taken from the YouTube video title.
  + **Quote.js** **:**  
    This file defines the Mongoose schema for the Quotes collection, which stores motivational quotes to be displayed on the homepage.  
    his data:
    - **name (**String): The name of the person who said the quote.
    - **quote** (String): The motivational quote text.
  + **UserSession.js :**   
    This file defines the Mongoose schema for the UserSessions collection, which tracks each user's current training session and progress in the Continue Routine feature.  
    his data:
    - **userName** (String): The username associated with the session.
    - **videos** (Array of Strings): A list of three YouTube URLs representing the challenges the user needs to complete.
    - **checks** (Array of Booleans): A progress-tracking array indicating which challenges the user has completed, aligned with the videos array.
    - **openedsessions** (Number): The total number of sessions the user has started.
    - **completesessions** (Number): The total number of sessions the user has fully completed.
    - **finished** (Boolean): A flag indicating whether the user has completed the current session (used for determining whether to load new challenges or resume the previous ones).
  + **UsersLike.js :**   
    This file defines the Mongoose schema for the UsersLike collection, which maps users to the challenge videos they have liked.  
    his data:
    - **userName** (String): The username of the user who liked videos.
    - **url** (Array of Strings, Required): A list of YouTube URLs representing the challenge videos the user has liked.

**7) Frontend Files & Functionality**

**Folder Structure files and logic**

* **Root**  
  Contains essential configuration files that define the project's build settings, dependencies, styling, and deployment configurations for the Preact Vite application.

**His files content and their functions:**

* + **package.json**  
    Defines project metadata and dependencies, including Preact core dependencies, routing libraries, and development tools like Vite and Tailwind CSS. Also contains npm scripts for development, building, and preview.
  + **package-lock.json**  
    Auto-generated file that locks dependency versions to ensure consistent installations across different development environments.
  + **tailwind.config.cjs**Configures Tailwind CSS framework settings, including dark mode support, custom animations (fadeIn, pulseSlow, wiggle), custom colors, and additional plugins for forms and typography styling.
  + **postcss.config.cjs**Sets up PostCSS processing pipeline with Tailwind CSS and Autoprefixer plugins for handling CSS transformations during build.
  + **vite.config.js**Configures Vite build tool settings, sets up Preact plugin, and establishes React compatibility aliases for the Preact application.
  + **vercel.json**Handles deployment configuration for Vercel platform, specifically setting up URL rewriting to support client-side routing by redirecting all routes to index.html.
* **Folder src/**   
  Contains core application files responsible for routing, global state management, utilities, and main app initialization. These files serve as the foundation of the frontend.

**His files content and their functions:**

* + **app.jsx**  
    Manages the application's main structure, routing, and global settings, ensuring smooth navigation and theme initialization.
  + **main.jsx**  
    Sets up the routing for the application using wouter and renders the main App component into the DOM.
  + **Index.css**  
    imports Tailwind CSS utilities and defines a fade transition animation for entering and exiting elements with opacity and translation effects.
* **Folder src/layouts**/  
  Contains page-level components that define the main structure of different sections in the application. Each layout component assembles multiple smaller components and manages their interactions. These components serve as containers that coordinate between the UI elements and the business logic stored in the utils folder.

**His files content and their functions:**

* + **ContinueRoutine.jsx**  
    Displays the workouts the user needs to perform, allows marking workouts as completed or liked, and sends the data after completion.
  + **Discover1Page.jsx**   
    Allows users to choose a workout category (Hands, Chest, Shoulders, Legs, Stomach, Full Body) by clicking action buttons, which navigate them to workout videos based on their selection.
  + **Discover2Page.jsx**  
    Displays workout videos for a selected body area, allows sorting by different criteria (e.g., most liked, difficulty), and manages user interactions such as liking videos.
  + **FirstPage.jsx**   
    The initial page of the application, featuring a logo and two buttons for navigating to the login or registration pages.
  + **InfoPage.jsx**   
    Displays and allows users to update personal information (age, height, weight) while calculating BMI and tracking workout session progress.
  + **LoginPage.jsx**   
    Provides the login interface where users can enter their username and password. Includes a toggle for password visibility, displays login messages, and shows a loading indicator during authentication.
  + **ManageUsers.jsx**   
    Provides an interface for admins to manage users, including viewing, sorting, updating, and deleting user details.
  + **NotSoFast.jsx**   
    Informs users that their account is pending admin approval, preventing login. Offers an option to navigate back to the homepage.
  + **PendingUsers.jsx**   
    Displays a list of users awaiting approval. Allows the admin to view pending users, sort them by name or creation date, and approve individuals.
  + **Registration1Page.jsx**   
    Collects user details during the registration process, including gender, age, height, and weight.
  + **Registration2Page.jsx**   
    Handles username and password setup with validation, error messages, password visibility toggles, and a confirmation popup.
  + **UserHomePage.jsx**   
    Displays a user dashboard with navigation buttons for workouts, personal info, and admin controls (if applicable). Includes a motivational quotes.
* **Folder src/utils**/  
  Contains custom hooks and utility functions that manage the business logic for the application. Each layout component typically has a corresponding utility file that handles its state management, API calls, and data processing. These utilities abstract complex operations away from the UI components, making the code more maintainable.

**His files content and their functions:**

* + **ContinueRoutine.js**  
    Manages the state and logic for the user's workout routine, handling video data, completion status, and like functionality. Main functions:

Main functions:

* + - **fetchUserSessionInfo**: Retrieves the user's session data including videos, completion status, and likes.
    - **handleDone:** Updates the completion status of a specific exercise.
    - **handleLike:** Updates the like status for a specific video.
    - **handleSubmit:** Checks if all exercises are completed and shows appropriate feedback.
    - **loadUserData:** Initializes the component by loading user session data.
  + **Discover1PageLogic.js**Handles the logic for the first page of workout discovery where users select their preferred body area.

Main functions:

* + - **handleBodyAreaChoice**: Saves the user's selected body area to session storage and navigates to the second discovery page.
  + **Discover2PageLogic.js**Manages the workout discovery page logic, handling video filtering, sorting, and like functionality.

Main functions:

* + - **fetchBodyPartVideos**: Retrieves videos filtered by the selected body part.
    - **changeVideosOrderAfterSort:** Updates video order based on selected sorting criteria.
    - **handleLike:** Updates like status for videos.
    - **loadUserData:** Initializes the component with filtered workout videos.
  + **InfoPageLogic.js**Manages user profile information, including personal metrics and workout statistics.

Main functions:

* + - **fetchUserInfo**: Retrieves user's personal information (age, height, weight).
    - **fetchUserSessionsInfo:** Gets user's workout session statistics.
    - **handleUpdateClick:** Validates and updates user's personal information.
    - **calculateBMI:** Computes user's BMI based on height and weight.
    - **fetchInitialData:** Loads all user data when component mounts.
  + **LocalSessionHelper.js**Provides utility functions for managing session storage operations in the browser.

Main functions:

* + - **saveToSessionStorage**: Stores data with specified key in session storage.
    - **getFromSessionStorage:** Retrieves stored data by key.
    - **removeFromSessionStorage:** Deletes specific data by key.
    - **clearSessionStorage:** Removes all stored session data.
  + **LoginPageLogic.js**Handles user authentication and login process, including form state and validation.

Main functions:

* + - **sanitizeInput**: Cleans user input by removing spaces.
    - **handleLogin:** Processes login request and manages navigation.
    - **handleLoginClick:** Manages loading state during login process.
    - **togglePasswordVisibility:** Controls password field visibility.
  + **ManageUsersLogic.js**Manages the admin dashboard functionality for user management.

Main functions:

* + - **fetchUsers**: Retrieves all users from the server for display.
    - **updateUser:** Updates user information and maintains sorting state.
    - **deleteUser:** Removes a user from the system and updates the display.
    - **sortUsersByName:** Toggles between alphabetical and creation date sorting.
    - **handleDeleteClick:** Shows confirmation popup before user deletion.
  + **PendingUsersLogic.js**Manages the state and functionality for handling pending user registrations in the admin dashboard.

Main functions:

* + - **fetchPendingUsers**: Retrieves list of users pending approval.
    - **approveUser:** Changes a user's status from pending to active.
    - **sortUsersByName:** Toggles between alphabetical and default (creation date) sorting.
    - **sortUsers:** Helper function to sort users alphabetically.
  + **Registration1PageLogic.js**Handles the first step of user registration, managing form state and validation.

Main functions:

* + - **clearStorageOnLoad**: Cleans up any existing registration data in session storage.
    - **validateInputs:** Checks if all required fields are filled with valid data.
    - **checkForErrors:** Updates error messages based on validation results.
    - **handleNext:** Saves form data to session storage and navigates to next step.
  + **Registration2PageLogic.js**Handles the second step of user registration, managing username/password form state, validation, and submission to the backend.

Main functions:

* + - **validateInputs**: Validates username and password requirements.
    - **handleSaveClick:** Saves form data to session storage and opens confirmation popup
    - **handlePopupConfirm:** Submits registration data to backend API
    - **registerUser:** Handles the actual API call for user registration
  + **RouteUtils.js**Manages the dynamic mapping of component names to their instances, enabling seamless conversion of route configurations from JSON to actual React components.  
    This utility facilitates modular and maintainable routing logic.

Main functions:

* + - **getComponentByName**: Retrieves a component reference by its name from the component map.
    - **getRoutes:** Converts JSON-defined route configurations into React route objects, dynamically attaching the appropriate components.
  + **ThemeLogic.js**Manages the application's theme switching functionality between light and dark modes.

Main functions:

* + - **initializeTheme**: Sets initial theme based on localStorage settings**.**
    - **toggleTheme:** Switches between light/dark themes and persists choice to localStorage**.**
  + **UserHomePageLogic.js**Manages the user's home page state and functionality, including motivational quotes and user authentication status.

Main functions:

* + - **fetchQuote**: Retrieves a motivational quote from the backend API**.**
    - **loadUserData:** Gets user information from session storage.
* **Folder src/components**/  
  The components folder and his content details is explained in section 8

**8) API calls in the application**

\* Some of the API contains action, to distinguish between different actions with the same request (like GET for example)

**Api/ continueRoutine.js:**

* **Method:** GET  
  **Action:** getInitalUserSessionData  
  **Description:** Retrieves session data for a user, including videos and like statuses.  
  **Response:** Returns a JSON object with user session data, videos, and likes.
* **Method:** GET  
  **Action:** getDoneVideoArray  
  **Description:** Retrieves the user's progress on exercises and updates their session if all challenges are completed.  
  **Response:** Returns a JSON object with the count of completed challenges (counterChecks).
* **Method:** PATCH  
  **Action:** patchDone  
  **Description:** Updates the completion status of a specific exercise for a user.  
  **Response:** Returns a message indicating the completion of the update.
* **Method:** PATCH  
  **Action:** patchLikes  
  **Description:** Updates the like status of a video for a user. Adds or removes the video from the user's likes list and adjusts the video's like count accordingly.  
  **Response:** Returns a message indicating the completion of the update.

**Api/ discover.js:**

* **Method:** GET  
  **Action:** sortByTitle  
  **Description:** Retrieves videos sorted by title in ascending or descending order for a specific body part.  
  **Response:** Returns a JSON object with sorted video data and the user's like statuses array.
* **Method:** GET  
  **Action:** sortByLikes  
  **Description:** Retrieves videos sorted by like count in ascending or descending order for a specific body part.  
  **Response:** Returns a JSON object with sorted video data and the user's like statuses array.
* **Method:** GET  
  **Action:** sortByDifficulty  
  **Description:** Retrieves videos sorted by difficulty level (beginner-first or advanced-first) for a specific body part.  
  **Response:** Returns a JSON object with sorted video data and the user's like statuses array.
* **Method:** GET  
  **Action:** filterByBodyPart  
  **Description:** Retrieves videos filtered by a specified body part.  
  **Response:** Returns a JSON object with filtered video data and the user's like statuses array.
* **Method:** PATCH  
  **Action:** updateVideoLikes  
  **Description:** Updates the like status of a video for a user. Adds a like if not already liked, or removes a like if already liked.  
  **Response:** Returns a message indicating the success of the like status update.

**Api/ homepage.js**

* **Method:** GET  
  **Action:** N\A  
  **Description:** Fetches a random quote from the QuotesDB database.  
  **Response:** Returns a JSON object containing the quote and the quoted person.

**Api/ login.js**

* **Method:** GET  
  **Action:** N\A  
  **Description:** Authenticates a user by validating the provided username and password. If the username exists and the password matches, the user's data is retrieved.  
  **Response:** Returns a JSON object with user details

**Api/ manageUsers.js**

* **Method:** GET  
  **Action:** N\A  
  **Description:** Retrieves the list of all users from the database.  
  **Response:** Returns a JSON object containing an array of users.
* **Method:** PATCH  
  **Action:** N\A  
  **Description:** Updates user details such as age, height, and weight based on the provided parameters.  
  **Response:** Returns a message indicating the successful update of user details.
* **Method:** DELETE  
  **Action:** N\A  
  **Description:** Deletes a user and all associated data, including user sessions, likes, and video like counts.  
  **Response:** Returns a message confirming the complete removal of the user and their related data.

**Api/ myInfo.js**

* **Method:** GET  
  **Action:** getUserData  
  **Description:** Retrieves user data based on the provided username.  
  **Response:** Returns a JSON object containing user details.
* **Method:** GET  
  **Action:** getUserSessionsData  
  **Description:** Retrieves session data for a specific user based on their username.  
  **Response:** Returns a JSON object containing the user's session details.
* **Method:** PATCH  
  **Action:** N\A  
  **Description:** Updates user details such as age, height, and weight for a specific user.  
  **Response:** Returns a message indicating the successful update of user details.

**Api/ pedndingUsers.js**

* **Method:** GET  
  **Action:** N/A  
  **Description:** Retrieves a list of pending users.  
  **Response:** Returns a JSON object with the list of users.
* **Method:** PATCH  
  **Action:** N/A  
  **Description:** Updates the registration status of a user. Requires a username to mark the user as registered.  
  **Response:** Returns a success message indicating the user registration status update.

**Api/ register.js**

* **Method**: POST  
  **Action**: N/A  
  **Description**: Registers a new user with provided details and creates user records, likes, and sessions.  
  **Response**: Success or error message based on registration outcome.

**9) Prompt used with AI**

**🔹 Database Optimization Prompt**

**Prompt:** *"How can I optimize MongoDB queries to reduce loading times?"*

**Implementation in the Project:**

Implemented **Indexes** on relevant collections to speed up queries.

Reduced **redundant database calls** by caching frequently accessed data.

Used **MongoDB Aggregation Pipeline** to combine multiple queries into a single, more efficient operation, reducing server load.

**🔹 Documentation Prompt**

**Prompt:** *"Provide a structured template for documenting code clearly and effectively."*

**Implementation in the Project:**

Standardized **code comments and documentation format** across the project.

Used JSDoc for documenting **functions, parameters, and return values**.

Ensured all major **components, API endpoints, and logic flows** were well-documented for easier maintenance.

**🔹 Hook Usage in Preact Prompt**

**Prompt:** *"Give me an example of how to use a hook in Preact to understand different use cases."*

**Implementation in the Project:**

Implemented **useState and useEffect hooks** in multiple components for **state management and lifecycle control**.

Used hooks for handling **theme switching, user session management, and dynamic UI updates**.

Ensured better **component reactivity and efficiency** by managing side effects properly.

**🔹 Web Application Architecture Prompt**

**Prompt:** *"Provide examples of different architectures for a web application."*

**Implementation in the Project:**

Designed a **modular front-end architecture** using **Preact components** and **Tailwind CSS** for styling.

Built a **Next.js-based backend** with **RESTful API endpoints** and **MongoDB** for scalable data management.

Ensured **separation of concerns** by organizing files into components, utils, api, and models directories.

**🔹 User Engagement Optimization Prompt**

**Prompt:** *"Suggest ways to use ChatGPT in a fitness app to increase user engagement."*

**Implementation in the Project:**

AI suggested integrating a **smart recommendation system** that suggests personalized fitness challenges based on user history.

Implemented **dynamic sorting of popular challenges/videos** at the top of the homepage based on user preferences.

Ensured an **interactive experience** by displaying motivational quotes and AI-driven challenge suggestions.

**🔹 Code Implementation Prompt**

**Prompt:** *"Write a JavaScript function that allows a user to mark a video as 'liked' and store it in a MongoDB database."*

**Implementation in the Project:**

Developed a **Like feature** where users can **click a button to mark a video as "Liked"**.

The function was written in **Preact** and communicated with the backend via an **API call**.

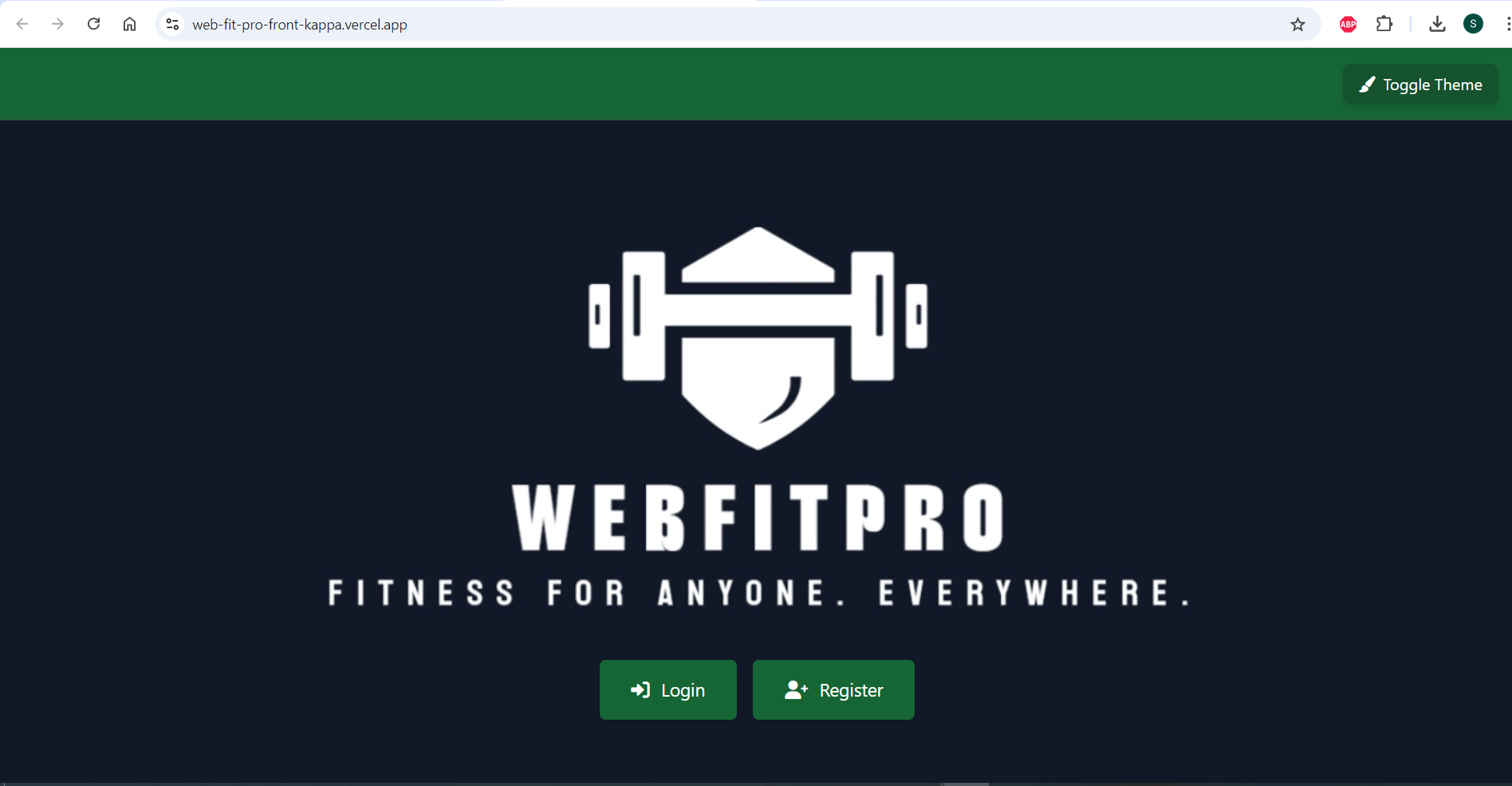
Updated the user's **personalized recommendations** based on their likes, enhancing engagement.

**9) תיק משתמש  
  
משתמשים אשר ניתן להשתמש בהם כדי לחוות את האפליקציה עם כל האפשרויות למטה**

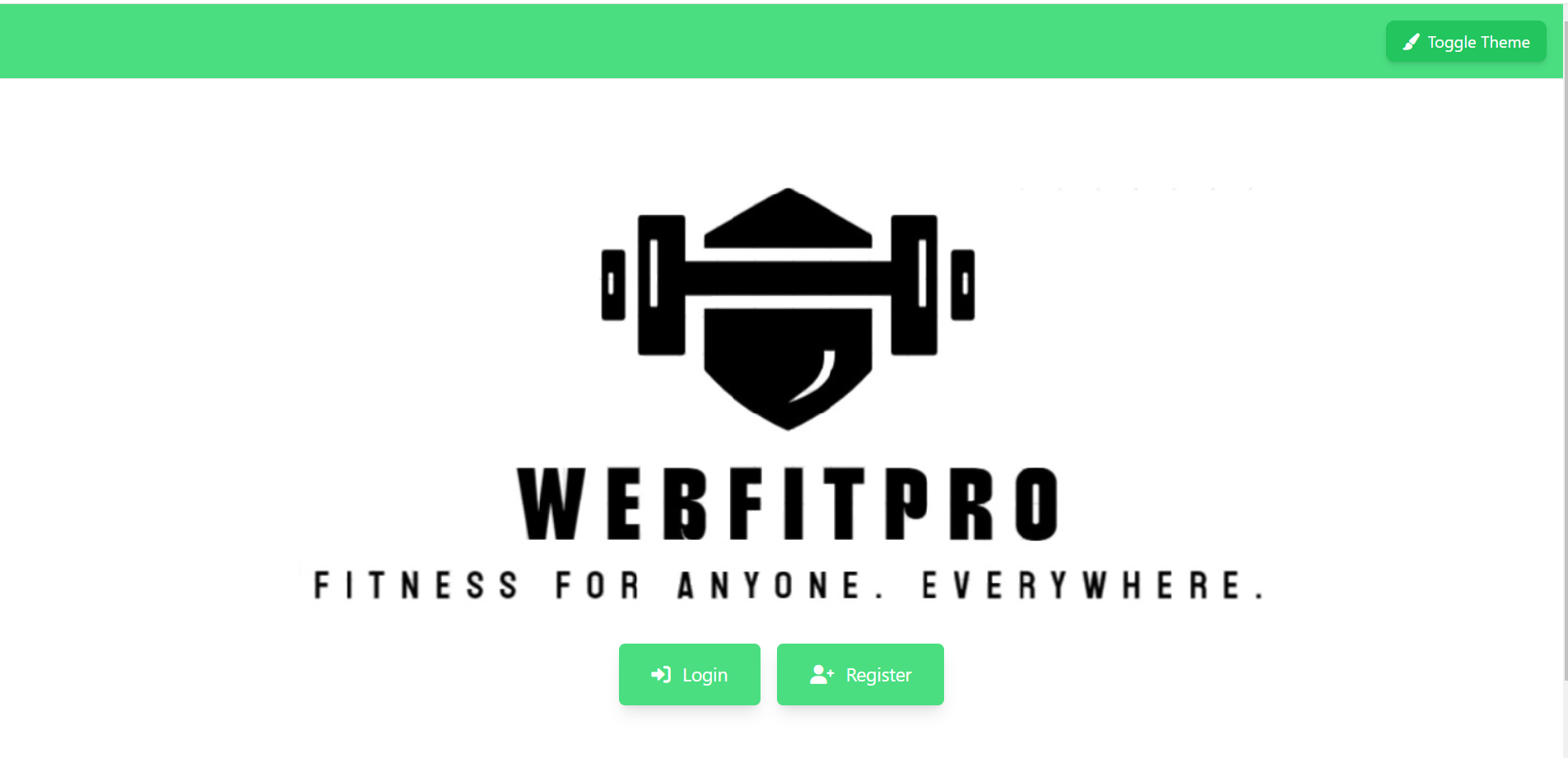
|  |  |  |
| --- | --- | --- |
| **פעולות המשתמש** | **שם המשתמש** | **סיסמא** |
| רגיל | Ido | Ido |
| רגיל + יכולות אדמין | admin | admin |
| משתמש חדש שעוד לא אושר במערכת על ידי האדמין | Almog | Almog |

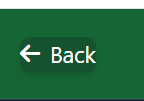
בכל מסך במערכת ניתן לשנות את הmode שבו המערכת תעבוד – light/dark. ע"י לחיצה על הכפתור: שמופיע בצד ימין עליון במסך.

Dark mode:



light mode:



בנוסף, בכל מסך במערכת (פרט למסך הכניסה) ניתן לחזור שלב אחורה ע"י לחיצה על הכפתור  המופיע בצד שמאל עליון במסך.

**מסך הכניסה:**

משתמשים חדשים:

משתמשים חדשים נדרשים להירשם לאתר. הרשמה לאתר תיעשה ע"י לחיצה על כפתור: תמונה שמכילה טקסט, צילום מסך, גופן, ירוק

התיאור נוצר באופן אוטומטי

שמופיע בצד ימין בתחתית המסך.

משתמשים קיימים:

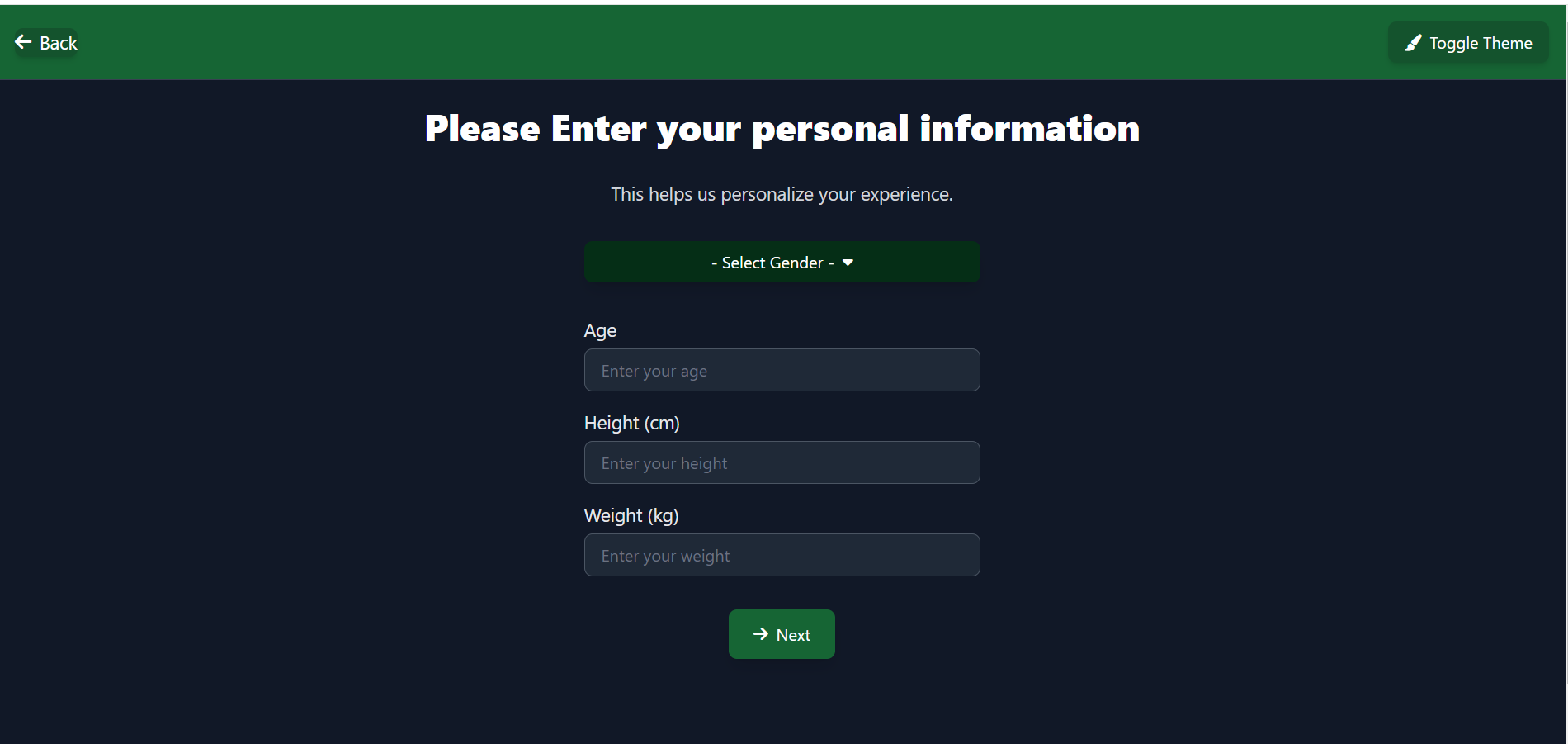
משתמשים קיימים יכולים להתחבר לאתר. ע"י לחיצה על כפתור: תמונה שמכילה צילום מסך, גופן, טקסט, ירוק

התיאור נוצר באופן אוטומטי שמופיע בצד שמאל בתחתית המסך.

תמונה שמכילה טקסט, צילום מסך, גופן, ירוק

התיאור נוצר באופן אוטומטי**מסך הרשמה לאתר:**

לאחר לחיצה על כפתור המופיע בדך הבית, נעבור למסך הבא:



במסך זה המשתמש נדרש להזין את הפרטים שלו, הכוללים גיל, גובה ומשקל, ולבחור את המגדר שלו.

נא לשים לב, שעל מנת להתקדם בהרשמה על המשתמש להזין פרטים תקינים. במידה והפרטים לא יהיו תקינים לא נוכל להתקדם ותופיע אחת מההערות הבאות: 

תמונה שמכילה גופן, גרפיקה, טקסט, עיצוב גרפי

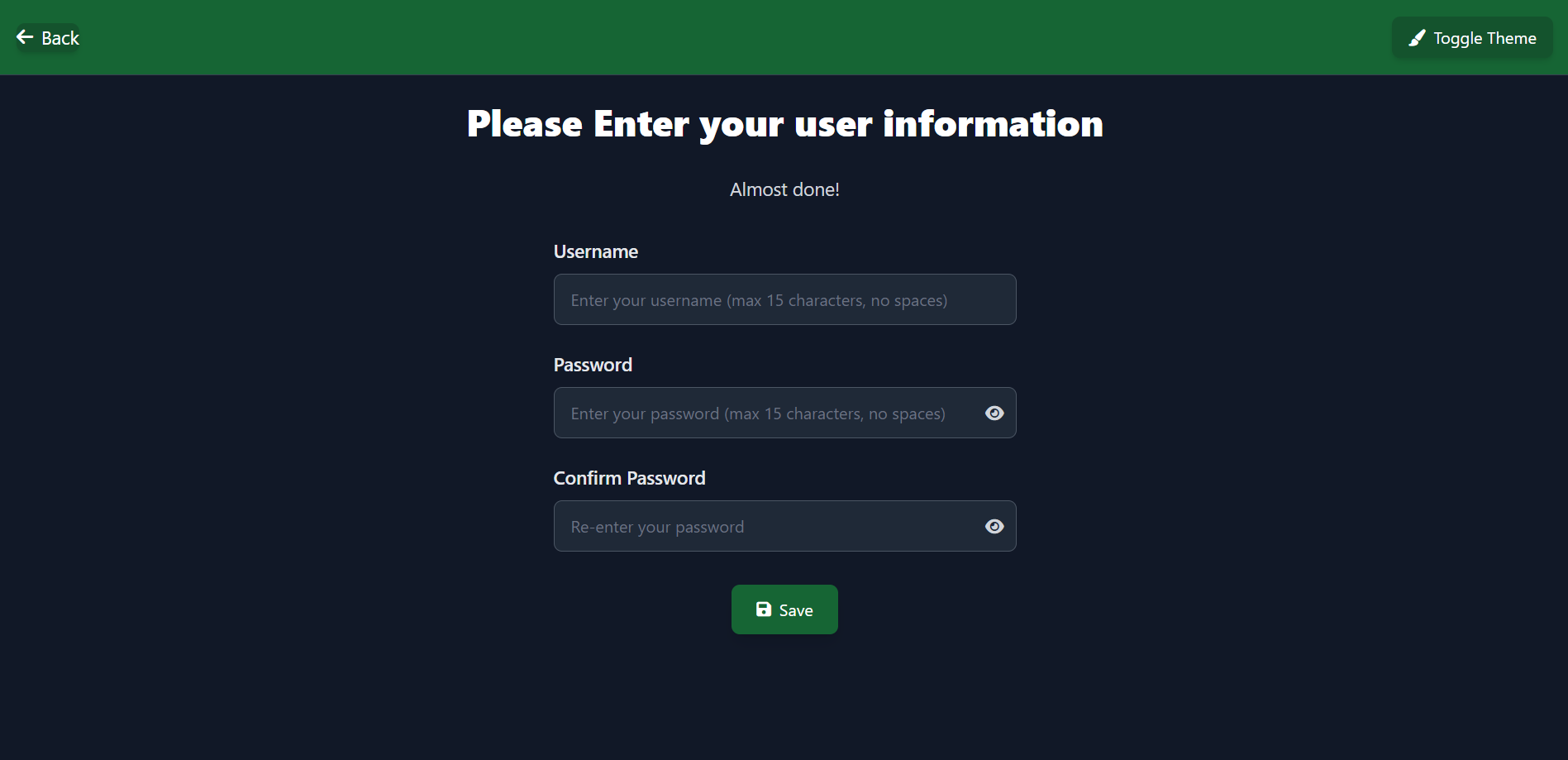
התיאור נוצר באופן אוטומטי



בהתאם לקלט השגוי שהוזן.

לאחר מילוי הפרטים ובמידה והם תקינים- נלחץ על הכפתור  על מנת להתקדם בתהליך.

לאחר מכן נעבור לדף הבא:



בדף זה על המשתמש לבחור ולהזין שם משתמש וסיסמא, ולאמת את הסיסמא ע"י הזנתה פעם נוספת.

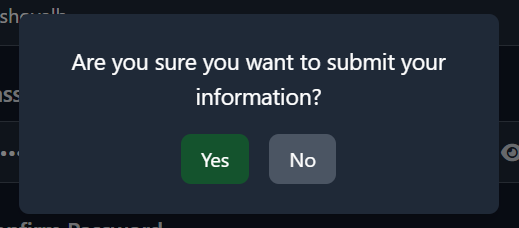
על המשתמש להזין קלט אל כל השדות, כאשר המספר המקסימלי של תווים שניתן להזין הוא 15 ואסור להזין רווחים. במידה והמשתמש יזין קלט ארוך יותר או בעל רווחים, המערכת אוטומטית תמחק את התווים הלא רלוונטיים- כלומר אם הוזן רווח הוא יימחק ואם הוזנו מעל 15 תווים הוא יחתוך את הקלט.

במידה והמשתמש לא הזין קלט אל כל השדות או שאימות הסיסמא לא תואם, לא נוכל להתקדם ויופיעו ההערות הבאות:



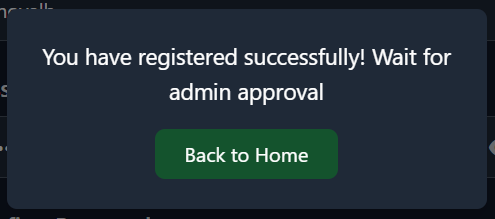




לאחר מילוי הפרטים ובמידה והם תקינים- נלחץ על הכפתור  על מנת להתקדם בתהליך.

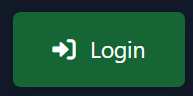
לאחר לחיצה על הכפתור נקבל את ההערה הבאה:

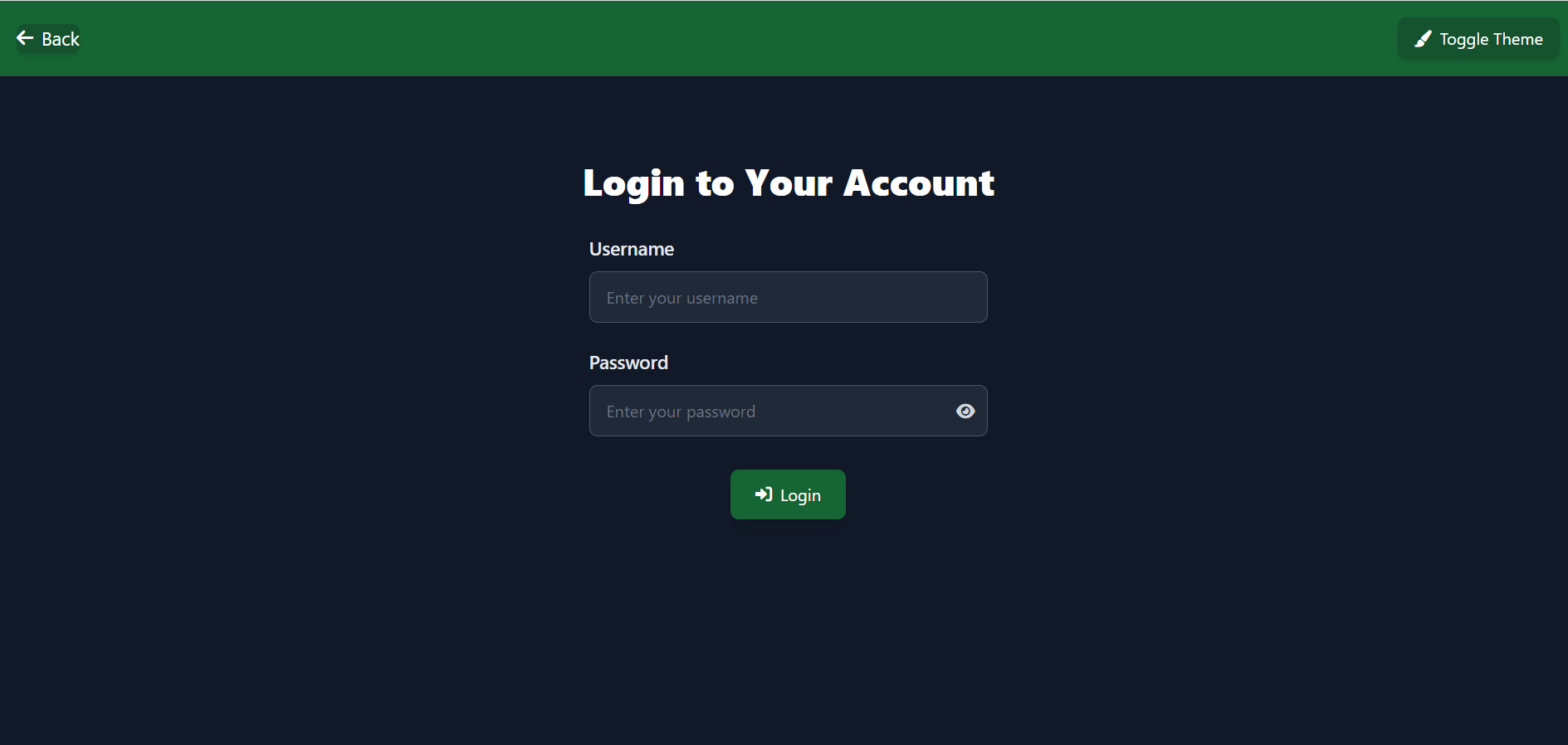
במידה ונרצה לשנות פרט כלשהו נלחץ על  מה שיאפשר לנו עריכה של הפרטים שהוזנו.

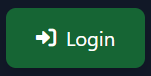
במידה ואנחנו בטוחים ומסכימים על הפרטים שהוזנו במסכים הקודמים, נלחץ על  ונקבל את ההערה הבאה:  כלומר נרשמנו בהצלחה ועלינו להמתין לאישור אדמין לפני שנוכל לבצע שימוש באתר.

**משתמשים קיימים:**

לאחר שמשתמש אושר ע"י אדמין (יש צורך באישור ראשוני ולאחר מכן ניתן להתחבר כל זמן) הוא יוכל לבצע התחברות לאתר.

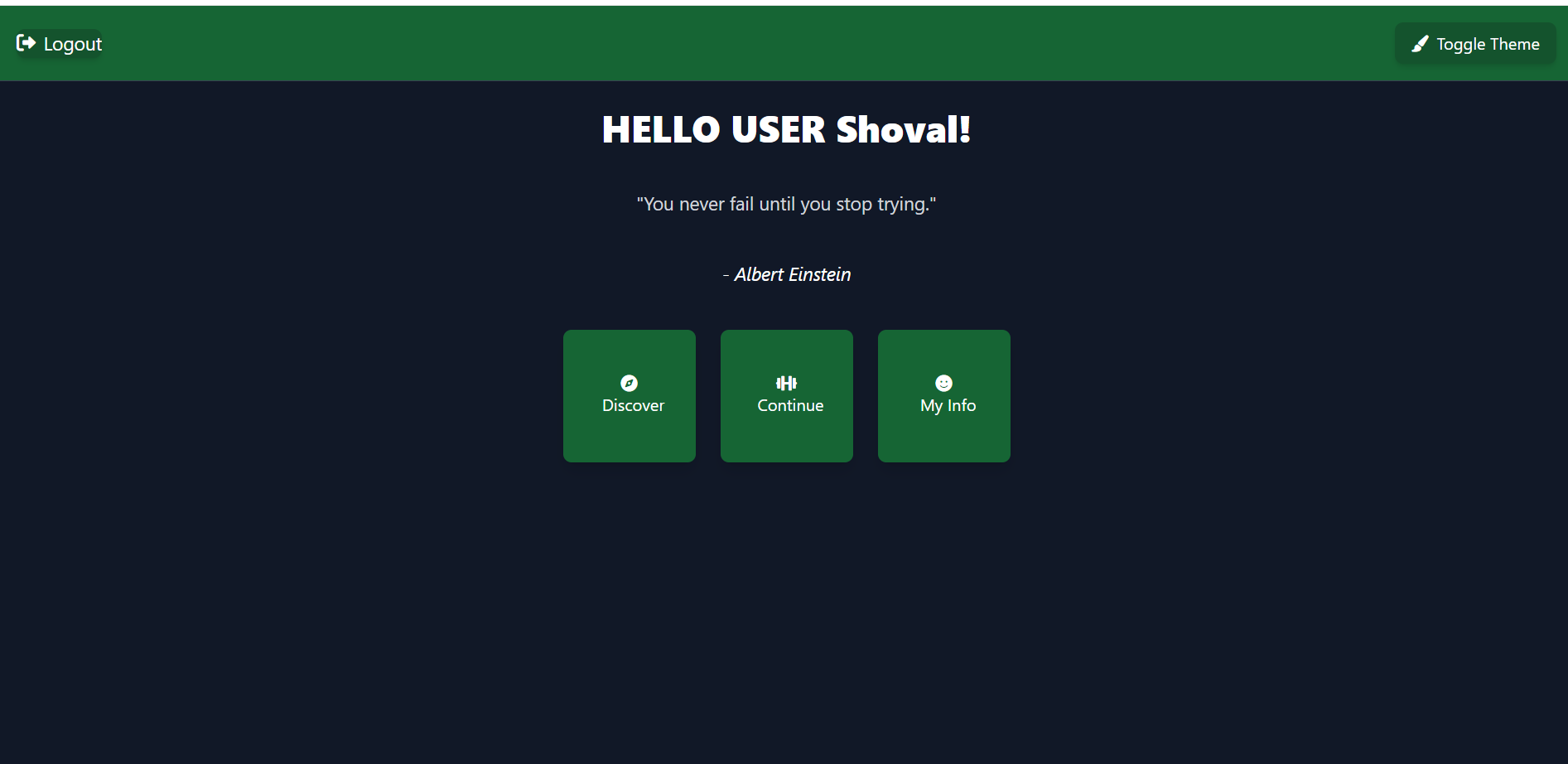
לאחר לחיצה על כפתור  המופיע בדף הבית, נעבור למסך הבא:



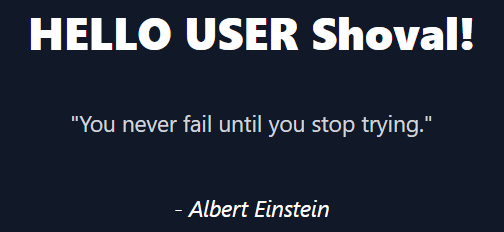
במסך זה על המשתמש להזין את שם המשתמש והסיסמא שבחר בעת ההרשמה למערכת, ולאחר מכן ללחוץ על הכפתור 

במידה והמשתמש הזין פרטים שגויים תופיע ההערה הבאה:

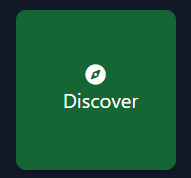
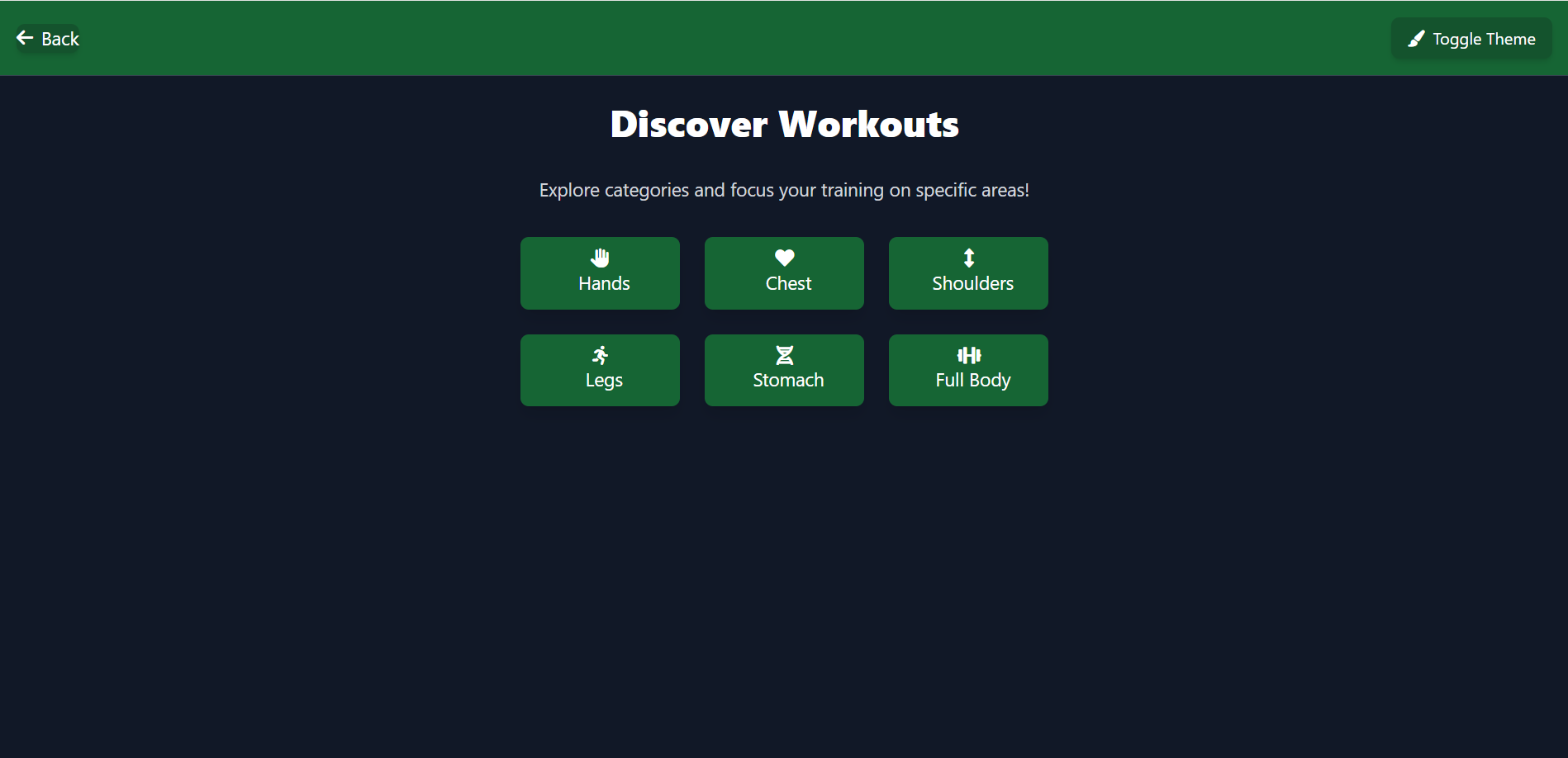
לאחר לחיצה על הכפתור ובמידה והמשתמש הזין פרטים תקינים, נעבור למסך הבא:



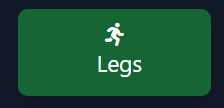
התחברנו בהצלחה!

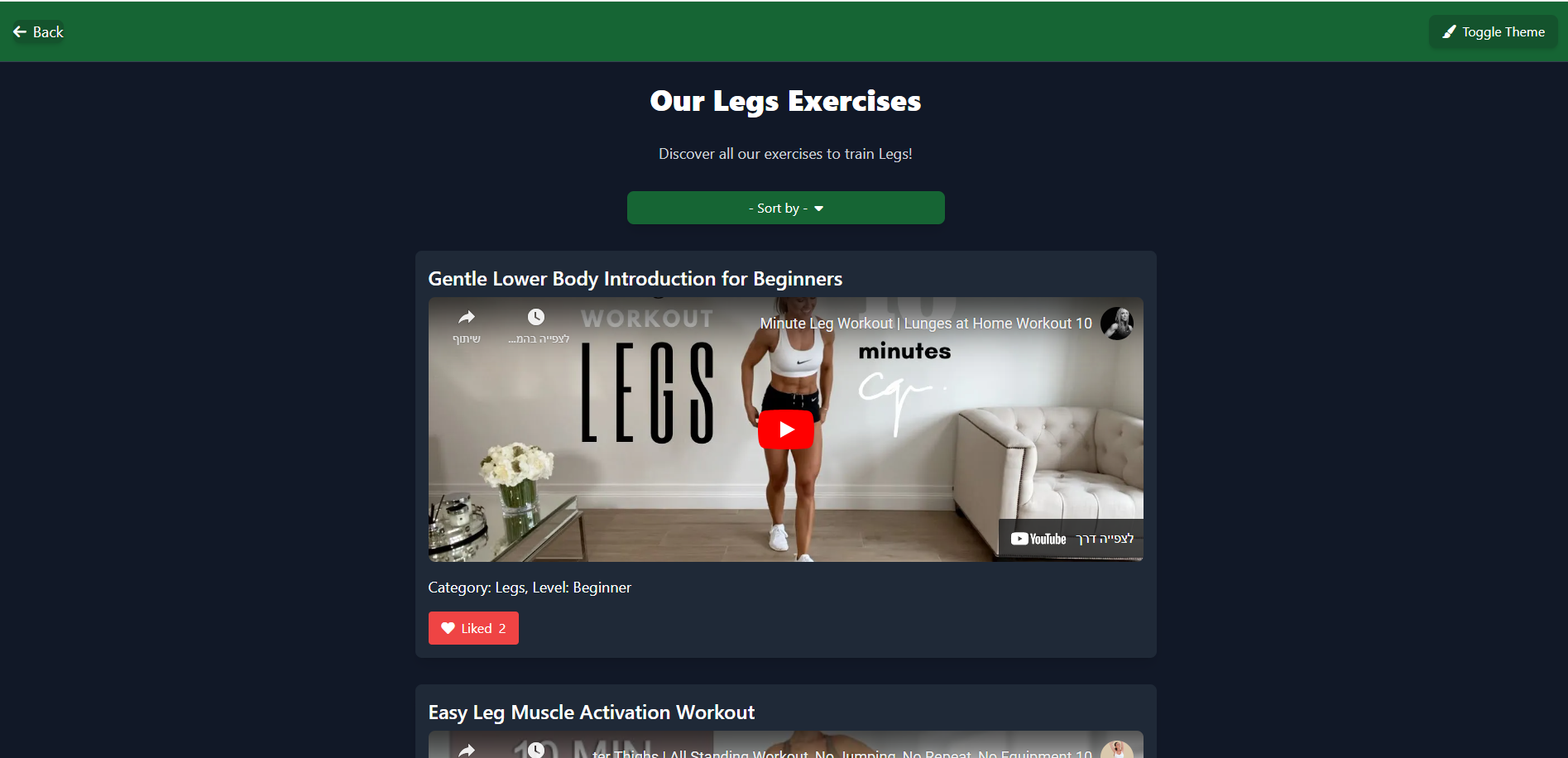
במסך זה נוכל לראות את הציטוט הנבחר עבורנו ואת שם המשתמש שלנו:

**Discover**:

ע"י לחיצה על הכפתור  נעבור למסך הבא: 

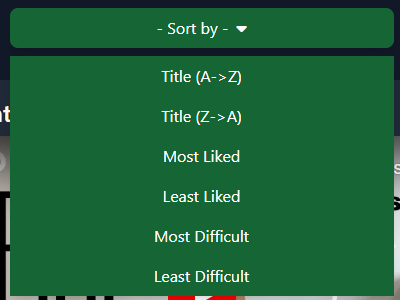
בחלק זה, נוכל לצפות בסרטוני תרגילים מתוך מאגר, המתמקדים בחלק בגוף שאותו נרצה לאמן.

לאחר בחירה בחלק אותו נרצה לאמן, למשל Legs, נלחץ על הכפתור המתאים לחלק אותו רוצים לאמן: 

ונעבור לעמוד הבא: 

בעמוד זה, יוצגו לנו מספר סרטונים מתוך מאגר, המתמקדים בחלק אותו בחרנו לאמן.

ע"י לחיצה על הכפתור  נוכל לבצע מיון ע"פ הקריוטריונים הבאים:



על כל סרטון נוכל לסמן לייק ע"י לחיצה על הכפתור 

ונוכל גם להוריד אותו ע"י לחיצה נוספת על הכפתור 

בנוסף, על כל סרטון תופיע רמת הקושי שלו, כך שנוכל להתאים את התרגילים אותם נרצה לבצע לרמת הקושי שלנו. 

**Continue Routine**:

ע"י לחיצה על הכפתור תמונה שמכילה צילום מסך, טקסט, גופן, גרפיקה

התיאור נוצר באופן אוטומטי נעבור למסך הבא:

תמונה שמכילה טקסט, צילום מסך, אתר, פרסום אונליין

התיאור נוצר באופן אוטומטי

עמוד זה מרכז את סשן האימון הנפתח עבורנו. כל סשן מכיל 3 תרגילים ברמת קושי שונה ומתמקד באיזור שונה בגוף.

גם כאן, כמו בעמוד discover, נוכל לסמן לייק או להסיר אותו מסרטונים, ע"י לחיצה על  או  בהתאמה.

לאחר ביצוע תרגיל כלשהו, נוכל ללחוץ על הכפתור תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

התיאור נוצר באופן אוטומטי ולסמן את הסרטון כ"הושלם". לאחר הלחיצה, הכפתור ישתנה ל. במידה ונרצה לבטל את הסימון של "הושלם", נוכל ללחוץ פעם נוספת על הכפתור על מנת לבטל את הסימון, והסרטון יחזור למצב המקורי.

לאחר ביצוע כל 3 התרגילים וסימונם, נלחץ על הכפתור תמונה שמכילה טקסט, גופן, צילום מסך, ירוק

התיאור נוצר באופן אוטומטי והסשן יסומן כסשן שהושלם בהצלחה, נקבל את ההערה הבאה: תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

במידה ולא סיימנו את כל התרגילים, ונלחץ על הכפתור תמונה שמכילה טקסט, גופן, צילום מסך, ירוק

התיאור נוצר באופן אוטומטי נקבל את ההודעה הבאה:

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי כלומר לא ניתן להתקדם מבלי להשלים את כל התרגילים.

**My info**:

ע"י לחיצה על הכפתור תמונה שמכילה צילום מסך, לוגו, גופן, גרפיקה

התיאור נוצר באופן אוטומטי נעבור למסך הבא: תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

המסך מכיל את הפרטים האישיים של המשתמש: גיל, גובה, משקל. על סמך פרטים אלו האתר מחשב ומציג למשתמש את הBMI שלו. תמונה שמכילה גופן, גרפיקה, לוגו, צילום מסך

התיאור נוצר באופן אוטומטי

בנוסף, האתר מציג למשתמש את ההתקדמות שלו בכך שהוא מראה לו את מספר הסשנים שהושלמו ואת מספר הסשנים שנפתחו עבורו לביצוע. תמונה שמכילה טקסט, גופן, גרפיקה, צילום מסך

התיאור נוצר באופן אוטומטי

תמונה שמכילה צילום מסך, גופן, טקסט, סמל

התיאור נוצר באופן אוטומטיהמשתמש יכול לבצע עדכון של הפרטים האישיים שלו: גיל, גובה ומשקל ע"י הזנה של הפרטים החדשים בשדות הרלוונטיים. על מנת לבצע שמירה של עדכון הפרטים נלחץ על הכפתור

במידה והפרטים יעודכנו בהצלחה נקבל את ההערה הבאה: 

במידה ויוזנו פרטים שגויים, נקבל את ההערות הבאות:







**Admin:**

לאחר לחיצה על כפתור תמונה שמכילה טקסט, צילום מסך, גופן, ירוק

התיאור נוצר באופן אוטומטי המופיע בדף הבית, נעבור למסך הבא:

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

במסך זה על האדמין להזין את שם המשתמש והסיסמא ולאחר מכן ללחוץ על הכפתור תמונה שמכילה גופן, טקסט, צילום מסך, סמל

התיאור נוצר באופן אוטומטי

במידה והאדמין הזין פרטים שגויים תופיע ההערה הבאה:

לאחר לחיצה על הכפתור ובמידה והאדמין הזין פרטים תקינים, נעבור למסך הבא:

תמונה שמכילה צילום מסך, טקסט, תרשים, עיצוב

התיאור נוצר באופן אוטומטי

ע"י לחיצה על אחד משלושת הכפתורים תמונה שמכילה טקסט, צילום מסך, גופן, ירוק

התיאור נוצר באופן אוטומטי

נעבור למסכים הרלוונטיים המוסברים לעיל (בחלק המדבר על משתמשים קיימים).

**Pending users**:

ע"י לחיצה על הכפתור תמונה שמכילה צילום מסך, טקסט, גופן, ירוק

התיאור נוצר באופן אוטומטי נעבור למסך הבא: תמונה שמכילה צילום מסך, טקסט, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

במסך זה ניתן לאשר משתמשים חדשים שנרשמו לאתר ומחכים לאישור אדמין.

ע"י לחיצה על הכפתור תמונה שמכילה טקסט, גופן, צילום מסך, ירוק

התיאור נוצר באופן אוטומטי ניתן לבצע מיון ע"פ שם המשתמש הנרשם.

במידה ונרצה לאשר משתמש כלשהו, נלחץ על הכפתור תמונה שמכילה גופן, צילום מסך, ירוק, גרפיקה

התיאור נוצר באופן אוטומטי המופיע בסמוך לשם המשתמש אותו נרצה לאשר. לאחר הלחיצה, המשתמש יאושר ולא יופיע יותר בעמוד זה. המשתמש יוכל להיכנס לאתר כ"משתמש קיים".

**Manage users**:

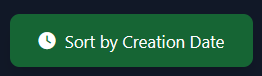
ע"י לחיצה על הכפתור תמונה שמכילה טקסט, צילום מסך, גופן, ירוק

התיאור נוצר באופן אוטומטי נעבור למסך הבא: תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

במסך זה נוכל לערוך פרטים אישיים של המשתמשים הקיימים באתר ולמחוק אותם.

ע"י לחיצה על הכפתור תמונה שמכילה טקסט, גופן, צילום מסך, ירוק

התיאור נוצר באופן אוטומטי ניתן לבצע מיון ע"פ שם המשתמש בסדר יורד, לחיצה נוספת תאפשר לנו למיין לפי סדר תאריך היצירה של המשתמשים באתר .

האדמין יכול לבצע עדכון של הפרטים האישיים של כל משתמש: גיל, גובה ומשקל ע"י הזנה של הפרטים החדשים בשדות הרלוונטיים. על מנת לבצע שמירה של עדכון הפרטים נלחץ על הכפתור תמונה שמכילה צילום מסך, גופן, טקסט, סמל

התיאור נוצר באופן אוטומטי

האדמין יכול למחוק משתמשים רשומים ע"י לחיצה על הכפתור תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

התיאור נוצר באופן אוטומטי לאחר לחיצה על הכפתור תופיע ההערה הבאה:

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

במידה ולא נרצה למחוק את המשתמש הנ"ל, נלחץ על הכפתור תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

התיאור נוצר באופן אוטומטי

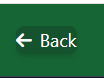
במידה ואכן נרצה למחוק את המשתמש הנ"ל, נלחץ על הכפתור  לאחר מכן, המשתמש ימחק מהאתר ולא יורשה לבצע כל פעולה.

במידה והמשתמש עדיין לא אושר באתר, לא נוכל לבצע פעולת עדכון/מחיקה והסטטוס שלו יופיע

לעומת משתמש רשום לאתר שהסטטוס שלו יופיע 

תמונה שמכילה צילום מסך, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

**התנתקות:**

במידה והמשתמש לא נמצא בדף הראשי, עליו לחזור לדף הראשי על ידי לחיצה על הכפתור

המופיע בצד שמאל עליון של המסך.

לאחר הגעה למסך הראשי של האתר, בצד שמאל עליון של המסך יופיע הכפתור תמונה שמכילה גופן, טקסט, צילום מסך, ירוק

התיאור נוצר באופן אוטומטי על מנת לבצע התנתקות מהמערכת, על המשתמש ללחוץ על כפתור זה.