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

**הגשת פרויקט**

**FitnessApp –** **B14 - G17**

|  |  |
| --- | --- |
| **שם חבר.ת הצוות** | **תז** |
| **אבישי חליפא** | **209404425** |
| **כפיר שרוני** | **206013559** |
| **עומר פטאל** | **318268935** |
| **תמיר נהרי** | 318189941 |

**GitHub**: <https://github.com/t5avish/WebFinalProject>

**Website**: <https://fitness-rho-nine.vercel.app>

**MoreThanWallet**: <https://www.morethanwallet.com/app/889>

תקציר הפרויקט:

**, המקום האולטימטיבי למסע הכושר שלך.FitnessAppברוכים הבאים ל**

האפליקציה שלנו מאפשרת למשתמשים רשומים להצטרף בקלות למגוון אתגרי כושר, לעקוב אחר ההתקדמות שלהם בצורה קלה ויעילה וכמו כן להוסיף אתגרים מותאמים אישית לרמתם.

! היום והשיגו את מטרות הכושר שלכם יחד עם הקהילה הפעילה שלנו.**FitnessApp**הצטרפו ל

**דף הבית:**במסך זה ניתן להירשם לאתר בתור משתמש וזאת על מנת שיתאפשר לבצע את מגוון הפעולות הקיימות באתר.  
במסך זה מוצגות הפעולות שניתן לבצע באתר כגון: התחברות לאתר ,יצירת פרופיל למשתמש, הצטרפות לאתגרי כושר, מעקב אחר ההתקדמות באתגרים, וכמו כן פרסום פוסטים לשיתוף ההישגים האישיים.

**דף האתגרים:**במסך זה מופיעים כלל אתגרי הכושר הקיימים במערכת (בין אם אלו אתגרים של האתר ובין אם אלו אתגרים שמשתמשים אחרים הוסיפו אל מאגר האתגרים).

עבור כל אתגר מוצג פירוט קצר על הנדרש שיש לבצע, וישנו כפתור הצטרפות לאתגר.  
לחיצה על כפתור זה משייכת את האתגר הנ"ל אל המשתמש שביצע זאת.  
  
כמו כן במסך זה ניתן להוסיף אתגרים בצורה עצמאית: לחיצה על כפתור הוספת אתגר פותחת טופס להוספת האתגר שמכיל שדות כגון כותרת, מספר ימים שיש לביצוע האתגר, המטרה היומית, סוג המדידה של האתגר ותיאור קצר שיסביר על האתגר.

**דף הפרופיל:**מסך זה מציג את פרופיל המשתמש ואת הפרטים האישיים, כגון שם, משקל, גובה וחישוב של ערך הבי.אם.אי העדכני.  
כמו כן ניתן לבחור תמונת פרופיל למשתמש (מתוך מאגר תמונות שקיימות במערכת).

במידה ומשתמש נרשם לאתגר מדף האתגרים, האתגר יוצג בדף הפרופיל שלו.  
לחיצה על כפתור "צפה באתגר" תפתח חלון פנימי שבו המשתמש יוכל לעדכן את ההתקדמות היומית שלו באתגר, ובנוסף יוכל לצפות בגרף ויזואלי המספק מידע ספציפי על קצב ההתקדמות של האתגר.

**דף הפוסטים:**  
בדף זה מוצגים כלל הפוסטים שהמשתמשים הרשומים למערכת פרסמו.  
המשתמש באתר יוכלו לפרסם פוסטים ובכך לשתף את התקדמותם באתגרים ואף לקבוע עם משתמשים אחרים לבצע אתגרים נוספים יחדיו.  
בנוסף ישנה אפשרות לאהוב פוסט שמופיע בדף, זאת על ידי לחיצה על כפתור "אהוב את הפוסט".

**טכנולוגיות מרכזיות שהשתמשנו בהם:**

**Frontend:**

**React -** library used for building dynamic user interfaces. It allows to create reusable UI components, making it easier to manage and update complex applications efficiently.

**Tailwind CSS** - framework that provides low-level utility classes to help style web applications. It allows for highly customizable designs without the need to write custom CSS, making it easy to build responsive and consistent interfaces.

**Chart.js** - library for creating interactive and visually appealing charts and graphs on web pages.

**Backend:**

**Next.js** - framework that enables server-side rendering and static site generation. It offers features like automatic routing and API routes.

**MongoDB** - database that stores data in flexible, JSON-like documents, making it ideal for handling large volumes of unstructured data.

**JWT** – (JSON Web Token) is a compact, URL-safe token used to securely transmit information between parties as a JSON object. It is commonly used for authentication and authorization in web application.

**CORS** - used in web development to manage and allow requests from different origins, enabling secure data exchange between servers and clients hosted on different domains.

**Vercel -** platform that enables developers to deploy, manage, and scale modern web applications with ease. It offers seamless integration with popular frameworks like Next.js, providing automated deployments and serverless functions

**מהנדס המערכת:** עומר פטאל

|  |  |  |
| --- | --- | --- |
| **שם חבר הצוות** | **משימות שהוקצו** | **משימות שהושלמו** |
| אבישי חליפא | בנית מסד הנתונים ב MongoDB + דף הבית (App.js) + דרישות פונקציונליות ולא פונקציונליות | כל המשימות הושלמו. |
| עומר פטאל | דף ה ChallengesPage + תרשים Use Case +יישום של Dark Mode | כל המשימות הושלמו. |
| כפיר שרוני | דף ה Posts + דף ה ProfilePage + התחברות למערכת | כל המשימות הושלמו. |
| תמיר נהרי | ארכיטקטורה+ תיק מתכנת + תיק משתמש | כל המשימות הושלמו. |

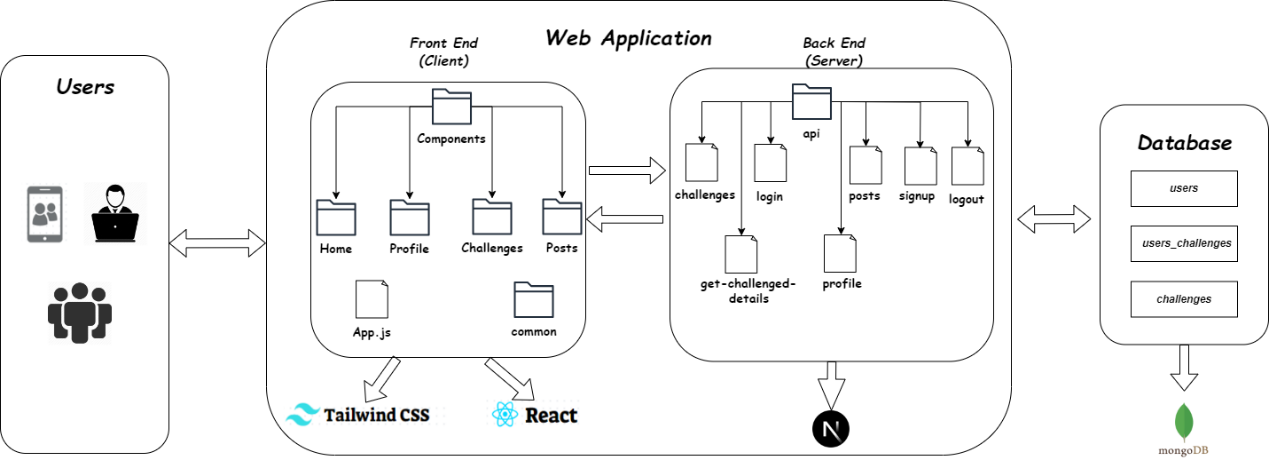
בנוסף ניתן להגיד שכן היה ממשק בין חברי הצוות, למשל:  
  
אבישי ועומר היו צריכים לשתף פעולה כדי להבטיח שהיישום של דף הבית וה-Dark Mode יהיו משולבים בצורה חלקה עם הגדרת מסד הנתונים.  
  
דוגמה נוספת היא שעומר שיתף פעולה עם כפיר כדי לשלב את דף האתגרים עם דף הפרופיל ודף הפוסטים (שזרימת הנתונים תהיה עקבית ונכונה).

**Functional Requirements:**

* **User Registration and Authentication:**
  + The app will allow users to create an account.
  + The app will allow users to log in and log out.
* **Profile Management:**
  + The app will allow users to create and update their profile details.
  + The app will allow users to view their profile information.
* **Challenges Management:**
  + The app will allow users to view a list of available fitness challenges.
  + The app will allow users to join a fitness challenge.
  + The app will allow users to create a new fitness challenge.
  + The app will allow users to view details of a challenge they joined.
* **Posts Management:**
  + The app will allow users to create a new post.
  + The app will allow users to view posts from other users.
  + The app will allow users to like posts.
  + The app will allow users to view the number of likes on each post.
* **Progress in challenges:**
  + The app will allow users to update their progress in the challenge.
  + The app will allow users to view visual charts showing their progress.
* **Mobile Responsiveness and Accessibility:**
  + The app will allow users a fully functional experience on various devices.
  + The app will allow users to switch between dark mode and light mode.

**Non - Functional Requirements:**

* **Usability:**
  + The user interface should be intuitive and user-friendly, allowing easy navigation in the app.
  + The app should adapt to various screen sizes and devices.
* **Data Integrity:**
  + The app shall ensure that data is consistently accurate, complete, and reliable across all operations.
* **Compatibility:**
  + The app shall be compatible with the latest two versions of major web browsers (Chrome ,Edge).
  + The app shall support iOS and Android platforms, ensuring consistent functionality across different operating systems.
* **Maintainability:**
  + The app's code shall be modular and well-documented to ensure easy updates and efficient maintenance.
  + The app shall allow for deployment of minor updates without requiring downtime or disrupting user experience.
* **Data Organization:**
  + The app shall categorize fitness challenges by the type of measurement (of time and length).

**Architecture:**

**UseCase:**

תמונה שמכילה טקסט, תרשים, קו

התיאור נוצר באופן אוטומטי

תיק משתמש

דף הבית:

**תמונה שמכילה טקסט, צילום מסך, תוכנה, דף אינטרנט

התיאור נוצר באופן אוטומטי**

**דף הבית מכיל בחלקו העליון את סרגל הניווט באתר.  
במרכז הדף יש מידע כללי על מטרת האתר ועל הפעולות הקיימות בו.**

**כמו כן ישנו כפתור יצירת משתמש על מנת לבצע רישום ראשוני לאתר.**

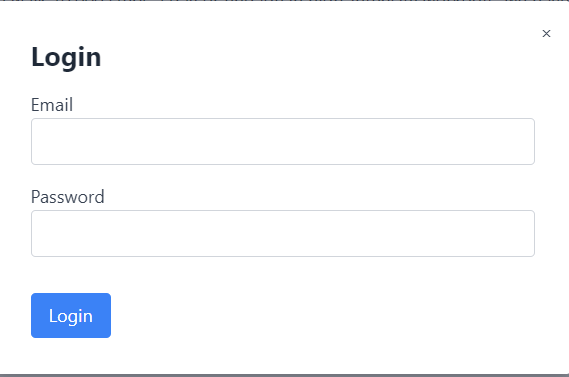
סרגל ניווט לפני ביצוע התחברות לאתר:

****

**סרגל ניווט זה מופיע בכל דפי האתר, ומאפשר בעצם ניווט קל וברור לכל הדפים שקיימים באתר.  
הכפתורים הקיימים בסרגל הם:**

**: Log In**

**לחיצה על כפתור זה פותחת טופס התחברות משתמש לאתר (אימייל וסיסמא).**

****

**: Dark Mode/ Light Mode**

**לחיצה על כפתור זה מאפשרת מעבר למצב לילה/מצב יום.**

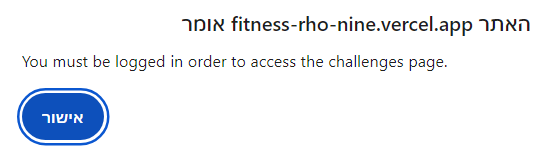
**: Challenges , Profile , Posts**

**במידה והמשתמש עדיין לא ביצע התחברות לאתר, אז לחיצה על כפתורים אלו תגרור הודעה למשתמש שעל מנת לעבור לדפים אלו, יש להיות מחוברים לאתר בתור משתמש.**

**:Home**

**לחיצה על כפתור זה מחזירה את המשתמש אל דף הבית.**

**במידה והמשתמש עדיין לא התחבר למערכת, אם הוא ינסה לעבור לשאר הדפים באתר, אז תוצג הודעת שגיאה:**

****

סרגל ניווט לאחר ביצוע התחברות לאתר:

****

**:Log Out**

**לחיצה על כפתור זה מבצעת התנתקות של המשתמש מהמערכת.**

**: Dark Mode/ Light Mode**

**לחיצה על כפתור זה מאפשרת מעבר למצב לילה/מצב יום.**

**: Challenges**

**לחיצה על כפתור זה מעבירה את המשתמש אל דף האתגרים.**

**: Profile**

**לחיצה על כפתור זה מעבירה את המשתמש אל דף הפרופיל של המשתמש.**

**: Posts**

**לחיצה על כפתור זה מעבירה את המשתמש אל דף הפוסטים.**

**:Home**

**לחיצה על כפתור זה מחזירה את המשתמש אל דף הבית.**

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

התיאור נוצר באופן אוטומטי**יצירת משתמש:

**לחיצה על כפתור יצירת משתמש פותח טופס רישום לאתר.  
טופס זה מכיל את פרטי הרישום הבאים:  
שם פרטי, שם משפחה, אימייל, סיסמא, גיל, משקל (בק"ג), גובה (בס"מ), מגדר.  
לאחר מכן על המשתמש ללחוץ על כפתור הרישום ובמידה והפרטים שהוא הזין תקינים, ייוצר משתמש חדש במערכת עם הפרטים.**

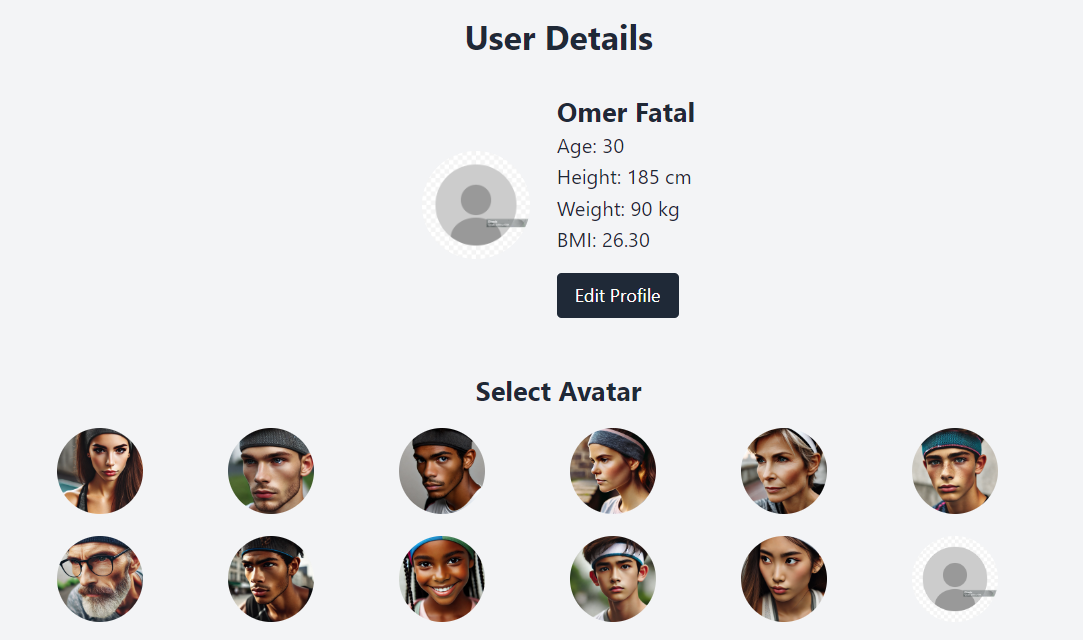
**:Profile**דף ה

**ואת תמונת הפרופיל.** BMI**דף זה מכיל את פרטי המשתמש איתם נרשם לאתר, חישוב של**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, דף אינטרנט

התיאור נוצר באופן אוטומטי**

**בלחיצה על תמונת הפרופיל, ניתן לבחור את תמונה מתוך מאגר תמונות קיימות באתר:**

****

**:Challenges**דף ה

**בנוסף ניתן לבצע עריכה של פרטי המשתמש (גיל, גובה, משקל) .**

**תמונה שמכילה טקסט, צילום מסך

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

התיאור נוצר באופן אוטומטיכמו כן דף זה מכיל את כל האתגרים אליהם המשתמש החליט להצטרף.**

**פותחת למשתמש חלון.View Challengeלחיצה על כפתור**

**בחלון זה ניתן לעדכן את ההתקדמות באתגר לפי תאריכים ספציפיים.  
לאחר לחיצה על תאריך מסוים, יפתח שדה טקסט שבו המשתמש יוכל לרשום ערך מספרי שמייצג את ביצוע האתגר באותו היום שנבחר. (ניתן לערוך שדה זה בכל רגע).  
כמו כן מופיע בחלק העליון של החלון את היעד היומי לביצוע האתגר.  
  
לאחר מילוי פרטים אלו יוצג גרף ההתקדמות באתגר.  
גרף זה מכיל את הערכים שהמשתמש מילא קודם לכן , לאורך התאריכים של האתגר.  
בגרף מופיע גם ממוצע מספרי של כלל הביצועים של המשתמש וקו אדום שמראה את היעד היומי הנדרש לאתגר.**

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

התיאור נוצר באופן אוטומטי**

**:Challenges**דף ה

**דף זה מכיל את כלל האתגרים הקיימים באתר.**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, דף אינטרנט

התיאור נוצר באופן אוטומטי**

**לכל אתגר בדף מופיעה הכותרת שלו, ותיאור קצר המפרט את מטרת האתגר.**

**.Join Challengeניתן להצטרף לאתגר על ידי לחיצה על כפתור**

**האתגרים שהמשתמש בחר להצטרף אליהם יופיעו לו בדף הפרופיל.**

**בנוסף, יש אפשרות להוסיף אתגר חדש אל דף האתגרים.  
 פותחת חלון למילוי פרטי האתגר החדשים.Add New Challengeלחיצה על כפתור**

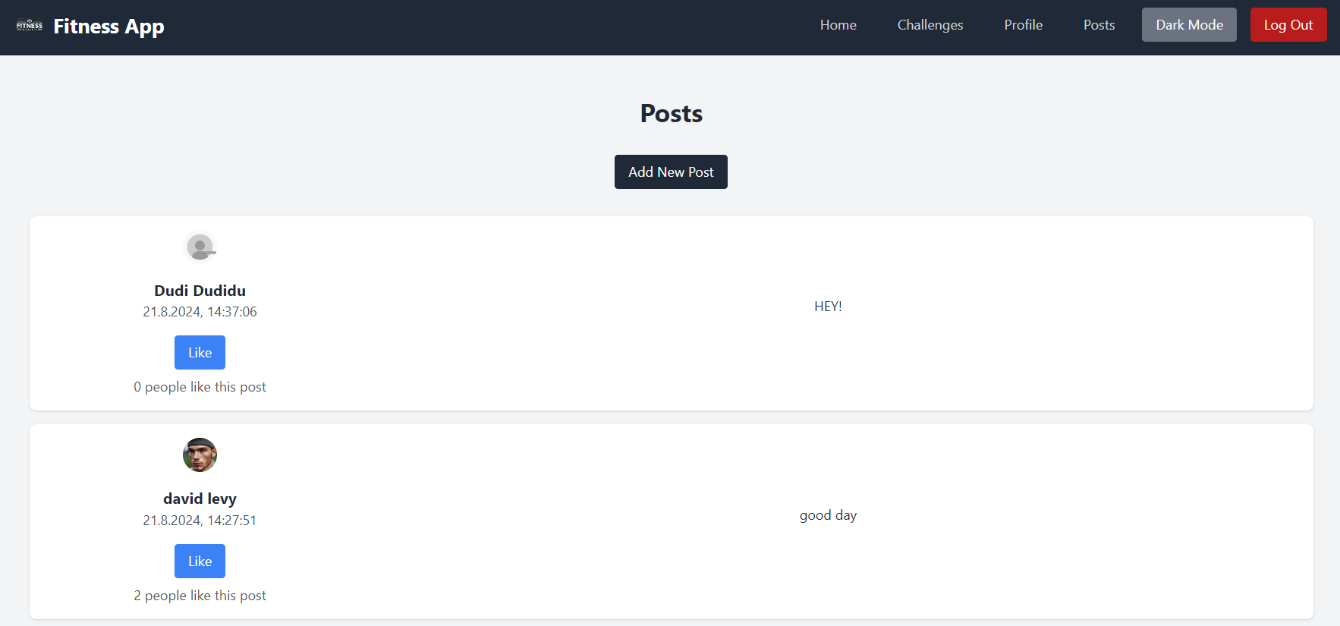
**תמונה שמכילה טקסט, צילום מסך, גופן, מספר

התיאור נוצר באופן אוטומטי**

**הפרטים שיש למלא כוללים: כותרת האתגר, מספר הימים של האתגר, יחידות המידה שבה ימדדו האתגרים, המטרה היומית, ותיאור קצר שיפרט על האתגר עצמו.  
 תוסיף את האתגר החדש אל דף האתגרים וניתן יהיה להצטרף לאתגר.Submitלחיצה על**

**:Posts**דף ה

**בדף זה ניתן לצפות בפוסטים של כל משתמשי האתר (רק מי שרשום יכול לפרסם פוסט).**

****

**לכל משתמש רשום יש אפשרות להעלות כמות בלתי מוגבלת של פוסטים.  
 נפתח חלון עם תיבת טקסט לכתיבת הפוסט.Add New Postבלחיצה על כפתור ה**

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

התיאור נוצר באופן אוטומטי**

**הפוסט שהמשתמש הקליד יוצג בדף הפוסטים.Submitלאחר לחיצה על כפתור**

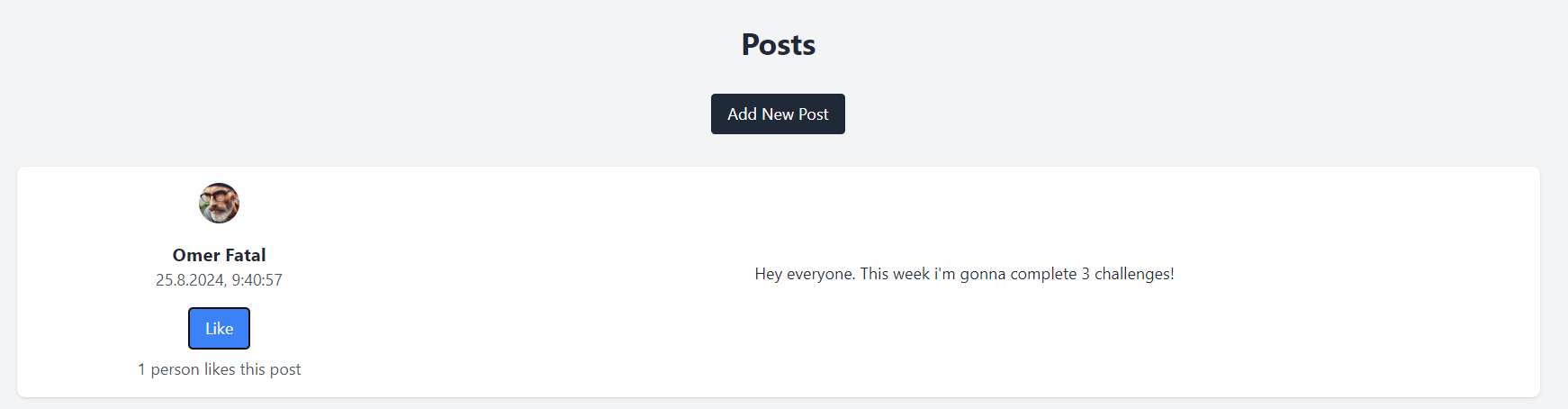
**תמונה שמכילה טקסט, צילום מסך

התיאור נוצר באופן אוטומטי**

**פעולה נוספת שהמשתמש יכול לבצע היא "לאהוב פוסט" שנרשם בדף.**

**תעלה את מספר הלייקים של אותו הפוסט באחד.Likeלחיצה על כפתור ה**

**ניתן לבצע פעולה זו לכל הפוסטים בקיימים אך פעם אחת בלבד עבור כל פוסט.**

****

תמונה שמכילה טקסט, צילום מסך, גופן, מלבן

התיאור נוצר באופן אוטומטי**במידה והמשתמש לחץ על כפתור הלייק עבור פוסט שהוא כבר ביצע לייק בעבר, הפעולה לא תתאפשר ותוצג למשתמש ההודעה:**

**:Dark Modeצילומי מסך של האתר במצב**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, תצוגה, תוכנה

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה צילום מסך, טקסט, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי**

**Programmer Portfolio**

Front-End

Our frontend is built using React and Tailwind CSS, offering a dynamic and visually appealing user interface. React provides a robust framework for creating interactive components and managing application state, while Tailwind CSS delivers a flexible, utility-first approach to styling, ensuring a responsive and modern design.

**App.js** - our home page, it handles user authentication, dark mode toggling, and navigation between pages like home and profile, with access depending on login status. The app also supports responsive navigation and dark mode.

Components

1. Home

A component that contains files related to the home page.

**1.1 HomeContent.js** - includes a welcoming introduction, a call-to-action button for creating an account, and a detailed "How it works" section that outlines the steps for users to sign up, join challenges, track progress, and share achievements.

**1.2 PageContent.js** - dynamically renders different pages based on the user's navigation. Depending on the currentPage state, it displays either the HomeContent, ChallengesPage, ProfilePage, or Posts components. Access to the profile and posts pages is restricted to logged-in users, ensuring content is tailored to the user's authentication status.

**1.3 SignUpForm.js** - manages user registration using the useSignUpForm hook for form handling. It includes fields for user details, displays error or success messages, and allows users to close the form.

**1.4 LoginForm.js** - form that allows users to log in. It uses the useLoginForm hook to manage form state and handle login logic.

**1.5 HomeHooks.js** – JavaScript file that contains all the hooks related to home component:

**useLoginForm**: Manages the state and submission of a login form. It handles email and password input, performs login via an API request, and manages error and success states.

**useSignUpForm**: Manages the state and submission of a sign-up form. It handles various user inputs, performs sign-up via an API request, and provides feedback on success or error.

2. Challenges

A component that contains files related to the challenges page.

**2.1 ChallengesPage.js** - manages and displays a list of challenges, along with a form for adding new challenges. It utilizes custom hooks to handle fetching challenges, submitting new ones, joining existing ones, and controlling the visibility of the challenge form.

**2.2 ChallengesHeader.js** serves as the header section for the challenges page, displaying a title and a prompt to add a new challenge. It includes a button that, when clicked, triggers the ‘handleAddChallengeClick’ function to open the form for adding a new challenge.

**2.3 ChallengeForm.js -** a form for creating a new challenge, including fields for the challenge title, duration, measurement type, daily goal, and description.

**2.4 ChallengeList.js -** displays a list of challenges in a responsive grid layout, with each challenge presented in a card format. Each card includes the challenge title, description, and a button to join the challenge, which triggers the ‘handleJoinChallenge’ function when clicked.

**2.5 ChallengesHooks.js** – JavaScript file that contains all the hooks related to home component:

**useChallenges**: This custom hook manages the state of the challenge-related data, including the list of challenges, form visibility, and form input fields like title, number of days, measurement, goal, and description. It returns these state variables along with their corresponding setter functions, providing centralized control for the challenge form.

**useFetchChallenges**: This custom hook is responsible for fetching the list of challenges from the server when the component is mounted. It uses the ‘useEffect’ hook to trigger the data fetching and updates the challenges state with the fetched data, handling any errors that occur during the process.

**useHandleSubmitChallenge**: This hook handles the form submission process for adding a new challenge. It validates the form inputs, sends a POST request to the server with the challenge data, updates the challenge list upon successful submission, and resets the form fields, including hiding the form after submission.

**useHandleJoinChallenge**: This custom hook manages the process of joining an existing challenge. It sends a PUT request to the server with the challenge ID, ensuring the user is authenticated via a token stored in local storage, and provides feedback based on the success or failure of the operation.

**useHandleFormVisibility**: This hook controls the visibility of the challenge form and resets the form fields when the form is closed. It provides functions to either show the form for adding a new challenge or close it, resetting all input fields to their initial state.

3. Profile

A component that contains files related to the challenges page.

**3.1 ProfilePage.js** - manages the display and interaction of a user's profile, incorporating hooks for fetching user data, handling avatar selection, and managing challenge-related actions. It conditionally renders user details, a list of challenges, and specific challenge details, including a graph that visualizes the user's progress toward their fitness goals.

**3.2 UserDetails.js -** displays and manages the user's profile information, including their age, height, weight, and BMI, with an option to edit these details. It also includes an avatar section, allowing users to change their profile picture through an avatar selector.

**3.3 ChallengeDetails.js -** display detailed information about a selected challenge, including a daily goal and a graph of progress over time. It utilizes Chart.js to render a line chart and allows users to interact with individual days within the challenge through clickable buttons. If the user selects a day, they can input a number value related to that day's progress and save it.

**3.4 ChallengesList.js -** display a list of challenges that a user has joined. It provides a responsive and interactive interface where users can view details of individual challenges.

**3.5 AvatarSelector.js -** provides an interactive interface for users to select an avatar from a predefined list. It is designed with a responsive grid layout and includes hover effects to enhance user interaction.

**3.6 ProfileHooks.js** – JavaScript file that contains all the hooks related to profile component:

**useUserProfile** - manages the state related to user profile data, including user information, loading status, error handling, avatar selection, and edit mode. It provides state variables and setter functions for handling user profile details and UI states.

**useFetchUser** - performs as a fetch operation to retrieve user profile data. It handles authentication, updates state variables with user information, and manages error and loading states.

**useHandleAvatarSelect** - manages the selection of a user avatar by sending the chosen avatar and updating the profile. It handles the opening and closing of the avatar selector and error management for the avatar update process.

**useHandleChallenge** - responsible for managing challenge-related states and operations, such as viewing challenge details and saving changes to challenge values. It includes functions for handling user interactions with challenges and updating challenge data.

4. Posts

A component that contains files related to the posts page.

**4.1 Posts.js -** integrates various hooks to manage the state and behavior of a posts feature, including fetching posts, handling form visibility, and managing error popups. It renders a button to add new posts, a form for creating posts, a list of posts, and an error popup based on the state managed by the custom hooks.

**4.2 PostList.js -** displays a list of posts with user information, including profile picture, name, and the number of likes. It also provides a button to like each post, updating the like count dynamically.

**4.3 PostForm.js -** provides a modal for users to create new posts, including a text area for input and a submit button.

**4.4 ErrorPopup.js -** displays an error message in a modal when a user trying to like a post that he already liked before.

**4.5 PostsHooks.js -** JavaScript file that contains all the hooks related to posts component:

**usePostsState**: manages the state for posts, form visibility, and user data, including text input and error messages. It provides functions and state variables to control and update the post-related UI and handle interactions.

**useFetchPostsAndUserProfile**: fetches posts and user profile data from the server, handling authentication and updating state based on the response. It sets error messages if data fetching fails and manages the initial data load when the component mounts.

**useHandleSubmitPost**: handles form submission for creating new posts, sending the post data to the server, and updating the posts list on success. It also manages error messages if the post submission fails and resets the form state accordingly.

**useHandleLikePost**: manages the process of liking a post, sending a request to update the like count on the server. It updates the posts list or displays an error popup if the like action fails, providing user feedback.

**useHandleFormVisibility**: controls the visibility of the post creation form, allowing it to be shown or hidden. It resets the form text when closing the form and manages the state for showing the form when adding a new post.

**useHandleErrorPopup**: manages the state of an error popup, including showing and hiding it based on error messages. It provides a way to reset error states when the popup is closed, ensuring users receive feedback on issues.

5. Common

**5.1 NavBar.js -** provides a responsive navigation bar for the Fitness App, including links to various sections, a dark mode toggle, and conditional login/logout functionality.

**5.2 Footer.js -** a simple, responsive footer for the Fitness App that displays a copyright notice at the bottom of the page.

**Key functions we used:**

**Fetch -** used to make network requests to a specified URL, returning a promise that resolves to the response of the request. It allows for easy handling of asynchronous operations and supports various HTTP methods, including GET, POST, PUT, and DELETE.

Example, POST request to the signup API:

**A screen shot of a computer code

Description automatically generated**

**localStorage.getItem -** retrieves a token stored in the browser's local storage. If the token is not found, it means the user is not authenticated and the function returns. This code is used to check if a user is logged in before allowing them to access certain functionality or parts of an application.

Example, checking if user is logged in before letting him see content:

**A screen shot of a computer

Description automatically generated**

Back-End

For out backend we used serverless functions on vercel.

We used MongoDB as our database, link to the database:

mongodb+srv://admin:1234@fitnessapp.nc7w2x0.mongodb.net/?retryWrites=true&w=majority&appName=FitnessApp

we stored the database link + JWT secret in **Environment Variables** on our vercel site.

Lib

mongodb.js - sets up a connection to a MongoDB database, The ‘connectToDatabase’ function returns a connected database client, handling and logging connection errors appropriately.

A screen shot of a computer program

Description automatically generated

API

**signup.js** - processes POST requests for user registration, integrating with a MongoDB database. It ensures that email addresses are unique by converting them to lowercase and checking for existing records before inserting new user data.

**profile.js** - processes user profile requests, including retrieving, updating, and modifying profile data. It verifies the user's JWT for authentication, calculates BMI based on user height and weight, and manages user challenges. The function handles errors and supports different HTTP methods (GET, POST, PUT) to fetch user details, update the avatar, and adjust personal metrics, respectively.

**posts.js** - handles GET requests to retrieve and sort posts by date, POST requests to create new posts with user data and content and PUT requests to add a like to a post while checking for duplicates.

**login.js** - designed for user authentication. It processes POST requests to validate user credentials by checking the email and password against stored data. On successful authentication, it generates and returns a JSON Web Token (JWT).

**logout.js** - For POST requests, it returns OK status with a message indicating that the user has been logged out.

**challenges.js** - provides endpoints for managing challenges in a MongoDB database. It supports GET requests to retrieve all challenges, POST requests to add new challenges, and PUT requests to join or update a user's participation in a challenge.

**get-challenge-details.js** - manages user challenge data in a MongoDB database. It supports POST requests to retrieve a user's challenge details and PUT requests to update the challenge data for a specific date.

Prompts we used in AI tools

- Can you help me understand how to use JWT for user authentication in next.js?

- Show me an example to how I can use Fetch function with a POST request

- How can I draw a line graph in JavaScript?

- What is CORS?

- How can I fix CORS policy issues?

- Can you show me how to connect my MongoDB to JavaScript application with next.js?