CS353 Database Systems Project
Final Report
Fitness Tracker and Trainer

20.05.2024

Bartu Albayrak - 22101640
Emre Melih Güven - 22003944
Hikmet Kaan Oktay- 22103766
Süleyman Yağız Başaran - 22103782
Serhat Yılmaz - 22002537

Group Number: 25

https://github.com/FitnessTrackerApp/FitnessTracker

1.) Description of Application

This project is a web based solution designed to support individuals in their fitness journey, from setting goals to achieving desired results. The app serves both fitness enthusiasts and professional trainers and offers a range of features to meet their unique needs. Users can create personalized profiles which contain their personal information, fitness goals, and achievements. These profiles are for tracking progress and accessing personalized recommendations. The app includes a diverse repository of workout routines applicable to various fitness levels, goals, and preferences. Users can explore and choose routines that match with their objectives, whether it's building muscle, losing weight or improving cardio. In addition to workout routines, the app provides access to nutrition plans designed to support users' fitness goals. These plans offer guidance on meal planning, calorie intake and helping users make informed dietary choices. Users are able to track their fitness progress over time such as workout sessions completed, calories burned, and other relevant information. Based on users' fitness goals, preferences and progress, the app delivers personalized recommendations for workouts and nutrition adjustments. These recommendations help users stay motivated and make continuous progress towards their goals. Professional trainers can use the web app to help trainees, creating customized workout plans, nutrition plans, monitoring progress, and providing real-time feedback. Trainers have access to trainee profiles and progress data, allowing for personalized coaching and support. Lastly, an admin dashboard provides oversight and management capabilities, allowing administrators to monitor user activity, generate reports, and ensure data integrity and security.

2) Contributions of Each Group Member

Regarding the development of the project, all group members have worked together exchanging ideas and commonly contributing to make the system fully functional and including all the necessary features. Also please checkout the ***GitHub*** commit logs to see where group members worked on. Individual members are focused mainly as follows:

Emre Melih Güven
The triggers that are deleting the user’s corresponding areas and automatically setting the age, start-end date triggers are written by me. Also schema.sql’s tables are created by me and to test the overall program, I wrote the sample data. On the frontend, I only organized the pages that are corresponding to the backend parts that are done by me. Also I wrote the trainer and admin’s nutrition and workout log display pages on the frontend using HTML. On the backend, I mainly focused on program requests of trainees. On the trainer, the requests are visible too, by the queries that I wrote. Also the nutrition and workout log queries are written by me. Other than those main parts, I changed some queries, backend connections with frontend and changed front and backend parts to display or take the data correctly. Worked on goals to save in the backend and fetch goal data of trainees too. Main part done by me was the program requests on the backend. To do these I changed homepages and added futures there too. Also Docker files are added by me to deploy the project.

Süleyman Yağız Başaran
I was working full-stack for this application, and started the project with creating all functions and their pages. Think of it like the skeleton of the project. I created the backend and their corresponding html pages, connected them with their related app.py functions. Most of the app.py code with routes, redirects, renders. Also I made queries for the login/register pages. Created conditions for trainer and trainees homepages and showed desired data such as all coaches, their data, changing information of the user, banning, adding goals, adding workouts etc. For instance pages that I created; adminhome, adminlogwatch, login, register, nutritionprog, programs, req-program, workoutprog, addgoal, add-pt, homepg, my-goals, profile, settings, workoutses, mealassign, trainerhomepg, trainerprofile, workoutassign. Also fixed some of the frontend related bugs in the way for the trainer home page.

Bartu Albayrak
I was responsible for the backend side of the project. I have written the logic of the nutrition assign, workout assign and addgoalfunctions and their corresponding queries. In addition, i have tried to connect the front side of the project to my code to properly work.
Hikmet Kaan Oktay
I was responsible for the frontend of the web application. By using html/css, I created the UI of the pages of our application. I also designed all the pages on Figma and made sure that we did not get far from these designs while implementing the pages of our web app. I also made changes in the app.py file to make sure that all the routes work and all the pages I created are fully usable in the project. I contributed to the process of creating the reports as all group members did. I helped my teammates who are responsible for the database of the project while we decide the general structure of the tables.

Serhat Yılmaz
I mainly contributed to the debugging of the project and some edits to the profile page, personalized workout and nutrition programs page, workout session page, and settings page. On the profile page, I added checks for height, weight, and fat percentage in boundaries and also added BMI calculation. On the personalized workout page and personalized nutrition programs page I checked the trainee's BMI and offered relevant nutrition programs according to his/her BMI and checked the trainee's fitness level and offered relevant workout programs. On the workout session page, I added a filtration in order to view the filtered list of workouts according to the filters. I added a report button on the settings page to report anything problematic as sending mail to admins via the mail application.

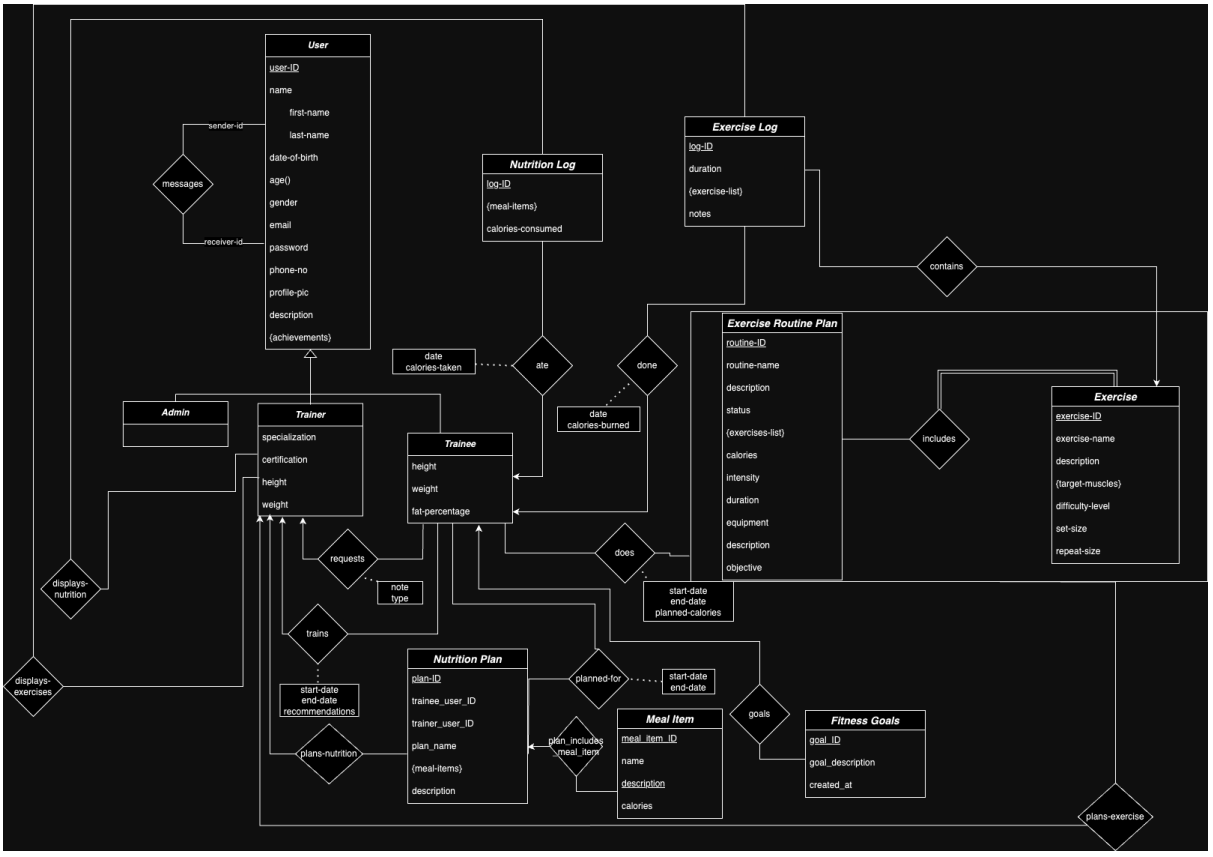
It is important to note that the contributions of each member are not limited to the above list as members interactively helped each other on their parts and worked together to overcome the challenges of implementing the application.

3) E/R (Revised)

Some minor changes were made to the schema.sql, which is MySql's volume on Docker, specifying the tables and initial inserts, triggers, and views. As seen in the table, some relations were added or deleted.

During the progress of the project we revised the E/R Diagram as follows:

- Fitness goals in the Trainee table moved to its table as Fitness Goals with relation named goals(between Trainee and Fitness Goals tables) in order to manage goals efficiently.
- trainee_user_ID and trainer_user_ID added to the Nutrition Plan table.
- Meal Item table added in order to manage meals effectively, plan_includes meal_item relation also added here between newly added Meal Item table and Nutrition Plan table.



4) Tables

1. User
    - **Relational Mode:**
      User (user_ID, first_name, last_name, date_of_birth, age, gender, email, password, phone_no, profile_pic, description, achievements, isTrainer)
    - **Primary Key:** user_ID
    - **Candidate Keys:** {(email)}
2. Admin

- **Relational Mode:**
  Admin (user_ID)
- **Primary Key:** user_ID
- **Foreign Key:** user_ID references User(user_ID)
- **Candidate Keys:** {(user_ID)}

3. Trainer
   - **Relational Mode:**
     Trainer (user_ID, specialization, certification, height, weight)
   - **Primary Key:** user_ID
   - **Foreign Key:** user_ID references User(user_ID)
   - **Candidate Keys:** {(user_ID)}
4. Trainee
   - **Relational Mode:**
     Trainee (user_ID, height, weight, fat_percentage)
   - **Primary Key:** user_ID
   - **Foreign Key:** user_ID references User(user_ID)
   - **Candidate Keys:** {(user_ID)}
5. FitnessGoals
   - **Relational Mode:**
     FitnessGoals (goal_ID, user_ID, goal_description, created_at)
   - **Primary Key:** goal_ID
   - **Foreign Key:** user_ID references User(user_ID)
   - **Candidate Keys:** {(goal_ID)}
6. NutritionLog
   - **Relational Mode:**
     NutritionLog (log_ID, meal_items, calories_consumed)
   - **Primary Key:** log_ID
   - **Candidate Keys:** {(log_ID)}
7. ExerciseLog
   - **Relational Mode:**
     ExerciseLog (log_ID, duration, exercise_list, notes)
   - **Primary Key:** log_ID
   - **Candidate Keys:** {(log_ID)}
8. NutritionPlan
   - **Relational Mode:**
     NutritionPlan (plan_ID, trainee_user_ID, trainer_user_ID, plan_name, meal_items, description)
   - **Primary Key:** plan_ID
   - **Foreign Keys:**
     - trainee_user_ID references Trainee(user_ID)
     - trainer_user_ID references Trainer(user_ID)
   - **Candidate Keys:** {(plan_ID)}
9. MealItem
   - **Relational Mode:**
     MealItem (meal_item_ID, name, description, calories)
   - **Primary Key:** meal_item_ID
   - **Candidate Keys:** {(meal_item_ID)}
10. PlanIncludesMealItem

- **Relational Mode:**
  PlanIncludesMealItem (plan_ID, meal_item_ID, quantity)
- **Primary Key:** plan_ID, meal_item_ID
- **Foreign Keys:**
  - plan_ID references NutritionPlan(plan_ID)
  - meal_item_ID references MealItem(meal_item_ID)
- **Candidate Keys:** {(plan_ID, meal_item_ID)}

11. PremiumAccount
    - **Relational Mode:**
      PremiumAccount (user_ID, premiumAcc_ID, start_date, end_date, payment_method)
    - **Primary Key:** user_ID, premiumAcc_ID
    - **Foreign Key:** user_ID references User(user_ID)
    - **Candidate Keys:** {(user_ID, premiumAcc_ID)}
12. Messages
    - **Relational Mode:**
      Messages (sender_id, receiver_id, message)
    - **Primary Key:** sender_id, receiver_id
    - **Foreign Keys:**
      - sender_id references User(user_ID)
      - receiver_id references User(user_ID)
    - **Candidate Keys:** {(sender_id, receiver_id, message)}
13. ate
    - **Relational Mode:**
      ate (user_id, log_id, date, calories_taken)
    - **Primary Key:** user_id, log_id
    - **Foreign Keys:**
      - user_id references User(user_ID)
      - log_id references NutritionLog(log_ID)
    - **Candidate Keys:** {(user_id, log_id, date)}
14. done
    - **Relational Mode:**
      done (user_ID, log_ID, date, calories_burned)
    - **Primary Key:** user_ID, log_ID
    - **Foreign Keys:**
      - user_ID references User(user_ID)
      - log_ID references ExerciseLog(log_ID)
    - **Candidate Keys:** {(user_ID, log_ID, date)}
15. trains
    - **Relational Mode:**
      trains (trainee_user_ID, trainer_user_ID, start_date, end_date, recommendations)
    - **Primary Key:** trainee_user_ID, trainer_user_ID
    - **Foreign Keys:**
      - trainee_user_ID references Trainee(user_ID)
      - trainer_user_ID references Trainer(user_ID)
    - **Candidate Keys:** {(trainee_user_ID, trainer_user_ID, start_date, end_date)}
16. planned_for

- **Relational Mode:**
  planned_for (user_ID, plan_ID, start_date, end_date)
- **Primary Key:** user_ID, plan_ID
- **Foreign Keys:**
  - user_ID references User(user_ID)
  - plan_ID references NutritionPlan(plan_ID)
- **Candidate Keys:** {(user_ID, plan_ID, start_date, end_date)}

17. Exercise
    - **Relational Mode:**
      Exercise (exercise_ID, exercise_name, description, target_muscles, difficulty_level, set_size, repeat_size)
    - **Primary Key:** exercise_ID
    - **Candidate Keys:** {(exercise_ID)}
18. ExerciseRoutinePlan
    - **Relational Mode:**
      ExerciseRoutinePlan (routine_ID, trainee_user_ID, trainer_user_ID, routine_name, description, calories, intensity, duration, equipment, status, exercises_list)
    - **Primary Key:** routine_ID
    - **Foreign Keys:**
      - trainee_user_ID references Trainee(user_ID)
      - trainer_user_ID references Trainer(user_ID)
    - **Candidate Keys:** {(routine_ID)}
19. PlansExercise
    - **Relational Mode:**
      PlansExercise (routine_ID, exercise_ID, quantity)
    - **Primary Key:** routine_ID, exercise_ID
    - **Foreign Keys:**
      - routine_ID references ExerciseRoutinePlan(routine_ID)
      - exercise_ID references Exercise(exercise_ID)
    - **Candidate Keys:** {(routine_ID, exercise_ID)}
20. Contains
    - **Relational Mode:**
      Contains (log_ID, exercise_ID)
    - **Primary Key:** log_ID, exercise_ID
    - **Foreign Keys:**
      - log_ID references ExerciseLog(log_ID)
      - exercise_ID references Exercise(exercise_ID)
    - **Candidate Keys:** {(log_ID, exercise_ID)}
21. Requests
    - **Relational Mode:**
      Requests (request_id, user_ID, trainer_ID, note, type)
    - **Primary Key:** request_id
    - **Foreign Keys:**
      - user_ID references User(user_ID)
      - trainer_ID references Trainer(user_ID)
    - **Candidate Keys:** {(request_id)}
22. CoachingRequests

- **Relational Mode:**
  CoachingRequests (request_id, trainee_user_ID, trainer_user_ID, request_date, status)
- **Primary Key:** request_id
- **Foreign Keys:**
  - trainee_user_ID references User(user_ID)
  - trainer_user_ID references User(user_ID)
- **Candidate Keys:** {(request_id)}

23. plans_nutrition
    - **Relational Mode:**
      plans_nutrition (user_ID, plan_ID)
    - **Primary Key:** user_ID, plan_ID
    - **Foreign Keys:**
      - user_ID references Trainee(user_ID)
      - plan_ID references NutritionPlan(plan_ID)
    - **Candidate Keys:** {(user_ID, plan_ID)}
24. does
    - **Relational Mode:**
      does (user_ID, routine_ID, exercise_ID, start_date, end_date, planned_calories)
    - **Primary Key:** user_ID, routine_ID, exercise_ID
    - **Foreign Keys:**
      - user_ID references Trainee(user_ID)
      - routine_ID references ExerciseRoutinePlan(routine_ID)
      - exercise_ID references Exercise(exercise_ID)
    - **Candidate Keys:** {(user_ID, routine_ID, exercise_ID, start_date, end_date)}

5) Implementation Details

5.1) Docker

With the containers and images, the application uses Docker. It creates an image of the application. Using the Dockerfile, docker-compose.yaml, requirements.txt, it is creating a container that is OS dependent. Hence, it becomes easier to deploy the program without specifically downloading all of the application programs.

5.2) MySql

The application uses MySql. Since we are using Docker, the volume is used. Hence, the application has a schema.sql file that is creating tables, views, triggers and inserting needed initialization data. To connect MySql to Flask, applications used special methods that will be mentioned in the Flask - Python section.

5.3) Flask - Python

On flask, we used a file called app.py to connect each front end page to the backend with functions. Each function processes the relevant backend needs. Queries, data

manipulations, if-else blocks are on this file. Also paths, methods like get and post, redirect urls etc. occur in this file. To connect the MySql, the app used connection cursors. After writing fetch statements, successful connections were done.

5.4) HTML - CSS

Html/css was used for the frontend of the project. This was a preference made by the team in order to keep the learning process shorter and simpler, since html/css is one of the simplest frontend tools to learn. Layouts and contents of all the pages were created using html and all their styling was made in css.

5.5) Difficulties

During the project implementation, we faced some challenges. The way we overcame these challenges was by researching on the internet and considering each other's ideas. Keeping the user interface functioning while the backend and the database of the project were changing and evolving was also challenging since it required regular checks and efforts. Also, schema.sql was a bit different in the beginning. While writing the back and front end, team struggled to establish some connections and queries that are derived from schema.sql.

One of the hardest part was connecting the frontend with the backend and database. Firstly, we had to write query for getting the info of the wanted instance, then give it to frontend to check and show. Put some buttons make their methods 'get' or 'post' to also get information from frontend. Then check them at backend side, make queries, make if-else conditions get errors then do the whole process again.

Also we had to debug all of our project by hand, by checking the database instances from frontend. This is because we used Docker. Instead of Docker if we used local repositories, we could've used PostMan which makes critically easy to debug backend and database code segments.

6) Advanced Database Components

6.1) Views

6.1.1) TrainerTraineeInfo

```
CREATE VIEW TrainerTraineeInfo AS
SELECT
   t.user_ID AS trainer_ID,
   u1.first_name AS trainer_first_name,
   u1.last_name AS trainer_last_name,
   tn.user_ID AS trainee_ID,
   t.specialization AS trainer_specialization,
   u2.first_name AS trainee_first_name,
   u2.last_name AS trainee_last_name,
   tr.recommendations
FROM
   User u1
JOIN Trainer t ON u1.user_ID = t.user_ID
```

```
JOIN trains tr ON t.user_ID = tr.trainer_user_ID
JOIN Trainee tn ON tr.trainee_user_ID = tn.user_ID
JOIN User u2 ON tn.user_ID = u2.user_ID;
```

- Purpose:

The TrainerTraineeInfo view is designed to provide detailed information about trainers and their trainees, including names, specializations, and any recommendations.

- Columns in the View:

trainer_ID: The unique identifier for the trainer (t.user_ID).
trainer_first_name: The first name of the trainer (u1.first_name).
trainer_last_name: The last name of the trainer (u1.last_name).
trainee_ID: The unique identifier for the trainee (tn.user_ID).
trainer_specialization: The trainer's area of specialization (t.specialization).
trainee_first_name: The first name of the trainee (u2.first_name).
trainee_last_name: The last name of the trainee (u2.last_name).
recommendations: Any recommendations the trainer has made for the trainee (tr.recommendations).

- Joins:

The view joins several tables to gather the necessary information:
User u1 joined with Trainer t to get trainer details.
Trainer t joined with trains tr to link trainers with trainees.
trains tr joined with Trainee tn to identify trainees.
Trainee tn joined with User u2 to get trainee details.

6.1.2) UserGoalsDetails

```
CREATE VIEW UserGoalsDetails AS
SELECT
   u.user_ID,
   u.first_name,
   u.last_name,
   fg.goal_description,
   fg.created_at
FROM
   User u
JOIN FitnessGoals fg ON u.user_ID = fg.user_ID;
```

- Purpose:

The UserGoalsDetails view is designed to provide information about users and their fitness goals, including the descriptions and creation dates of these goals.

- Columns in the View:

user_ID: The unique identifier for the user (u.user_ID).
first_name: The first name of the user (u.first_name).
last_name: The last name of the user (u.last_name).
goal_description: The description of the user's fitness goal (fg.goal_description).

created_at: The date when the fitness goal was created (fg.created_at).

- Join:

The view joins the User table with the FitnessGoals table on the user_ID to combine user information with their respective fitness goals.

- **Summary**

TrainerTraineeInfo View:
Provides comprehensive details about trainers and their trainees.
Includes trainer's ID, names, specialization, trainee's ID, names, and any recommendations given by the trainer.
Uses multiple joins to aggregate data from the User, Trainer, trains, and Trainee tables.

UserGoalsDetails View:
Provides details about users and their fitness goals.
Includes user ID, names, goal description, and goal creation date.
Uses a join between the User and FitnessGoals tables to merge user information with their fitness goals.

These views are useful for extracting and displaying relational data in a convenient, readable format without writing complex queries each time this data is needed.

6.2) Triggers

6.2.1) First Trigger: before_user_insert

```
DELIMITER //
CREATE TRIGGER before_user_insert BEFORE INSERT ON User FOR EACH ROW
BEGIN
   SET NEW.age = YEAR(CURDATE()) - YEAR(NEW.date_of_birth);
END;
//
```

- Trigger Type: BEFORE INSERT

This trigger is executed before a new row is inserted into the User table.

- Trigger Name: before_user_insert

The name of the trigger is before_user_insert.

- Target Table: User

This trigger is associated with the User table.

This trigger acts on each row that is inserted.

- Trigger Action:

Calculate Age: The trigger calculates the age of the new user based on their date_of_birth.
YEAR(CURDATE()): Gets the current year.

YEAR(NEW.date_of_birth): Extracts the year from the date_of_birth of the new row being inserted.
SET NEW.age = YEAR(CURDATE()) - YEAR(NEW.date_of_birth): Sets the age field of the new row to the difference between the current year and the year of birth.

6.2.2) First Trigger: after_user_delete

```
//
CREATE TRIGGER after_user_delete AFTER DELETE ON User FOR EACH ROW
BEGIN
   IF EXISTS (SELECT * FROM Trainer WHERE user_ID = OLD.user_ID) THEN
      DELETE FROM trains WHERE trainer_user_ID = OLD.user_ID;
      DELETE FROM NutritionPlan WHERE trainer_user_ID = OLD.user_ID;
      DELETE FROM ExerciseRoutinePlan WHERE trainer_user_ID = OLD.user_ID;
      DELETE FROM Trainer WHERE user_ID = OLD.user_ID;
   END IF;

   IF EXISTS (SELECT * FROM Trainee WHERE user_ID = OLD.user_ID) THEN
      DELETE FROM NutritionPlan WHERE trainee_user_ID = OLD.user_ID;
      DELETE FROM ExerciseRoutinePlan WHERE trainee_user_ID = OLD.user_ID;
      DELETE FROM trains WHERE trainee_user_ID = OLD.user_ID;
      DELETE FROM Trainee WHERE user_ID = OLD.user_ID;
   END IF;
END;
//
```

- Trigger Type: AFTER DELETE

This trigger is executed after a row is deleted from the User table.

- Trigger Name: after_user_delete

The name of the trigger is after_user_delete.

- Target Table: User

This trigger is associated with the User table.

This trigger acts on each row that is deleted.

- Trigger Action:
    - Conditional Checks and Deletions:
        - Check if the User is a Trainer:

```
IF EXISTS (SELECT * FROM Trainer WHERE user_ID = OLD.user_ID) THEN
```

Checks if the deleted user is listed as a trainer.

- Cascading Deletes for Trainer:

```
DELETE FROM trains WHERE trainer_user_ID = OLD.user_ID;
DELETE FROM NutritionPlan WHERE trainer_user_ID = OLD.user_ID;
DELETE FROM ExerciseRoutinePlan WHERE trainer_user_ID = OLD.user_ID;
DELETE FROM Trainer WHERE user_ID = OLD.user_ID;
```

Deletes all related records in trains, NutritionPlan, ExerciseRoutinePlan, and Trainer tables where the user_ID matches the deleted user's ID.

- Check if the User is a Trainee:

IF EXISTS (SELECT * FROM Trainee WHERE user_ID = OLD.user_ID) THEN
Checks if the deleted user is listed as a trainee.

- Cascading Deletes for Trainee:

DELETE FROM NutritionPlan WHERE trainee_user_ID = OLD.user_ID;
DELETE FROM ExerciseRoutinePlan WHERE trainee_user_ID = OLD.user_ID;
DELETE FROM trains WHERE trainee_user_ID = OLD.user_ID;
DELETE FROM Trainee WHERE user_ID = OLD.user_ID;
Deletes all related records in NutritionPlan, ExerciseRoutinePlan, and trains tables where the user_ID matches the deleted user's ID.

- **Summary**

before_user_insert Trigger: Automatically calculates and sets the age of a new user before they are inserted into the User table.
after_user_delete Trigger: Ensures that when a user is deleted, all associated records in related tables (Trainer, Trainee, trains, NutritionPlan, and ExerciseRoutinePlan) are also deleted to maintain referential integrity. This trigger handles cases where the deleted user could be a trainer or a trainee and deletes records accordingly.

6.3) Constraints

- The system cannot be used without logging in.
- A user cannot choose their birthday to be in the future.
- A trainee cannot display other trainees' profiles.
- Trainees can not take the same nutrition or exercise plan more than once.
- A Trainer cannot attach more than 20 exercises on a single exercise plan
- A Trainee cannot request nutrition or exercise plan more than once from the same trainer
- A trainee cannot save logs of more than 20

6.4) Reports

We hold reports as logs, NutritionLog and ExerciseLog:

```
CREATE TABLE NutritionLog (
   log_ID INT PRIMARY KEY AUTO_INCREMENT,
   meal_items VARCHAR(255),
   calories_consumed INT
);

CREATE TABLE ExerciseLog (
   log_ID INT PRIMARY KEY AUTO_INCREMENT,
   duration NUMERIC(4,1) DEFAULT 0,
   exercise_list VARCHAR(255) DEFAULT NULL,
   notes VARCHAR(255) DEFAULT NULL,
```

```
    CONSTRAINT max_duration_check CHECK (duration <= 180.0)
);
```

7) User's Manual

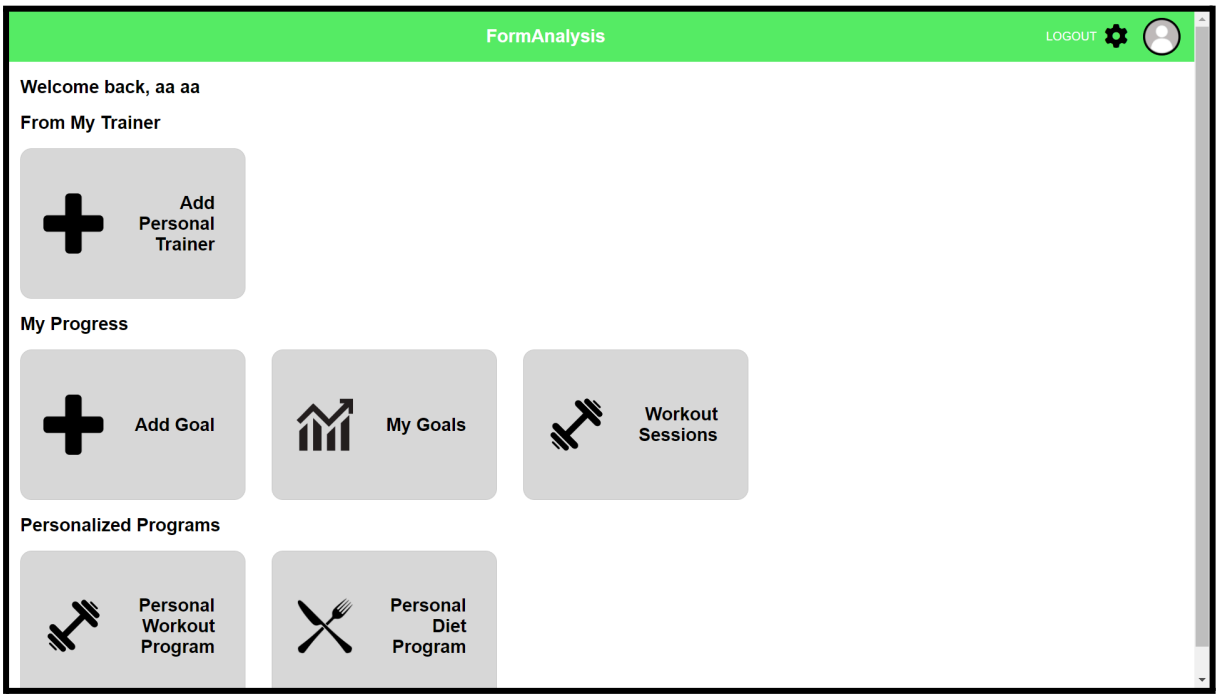
Login Screen

Users with an account can write their emails and passwords to login. Users without an account can go to the sign up link.

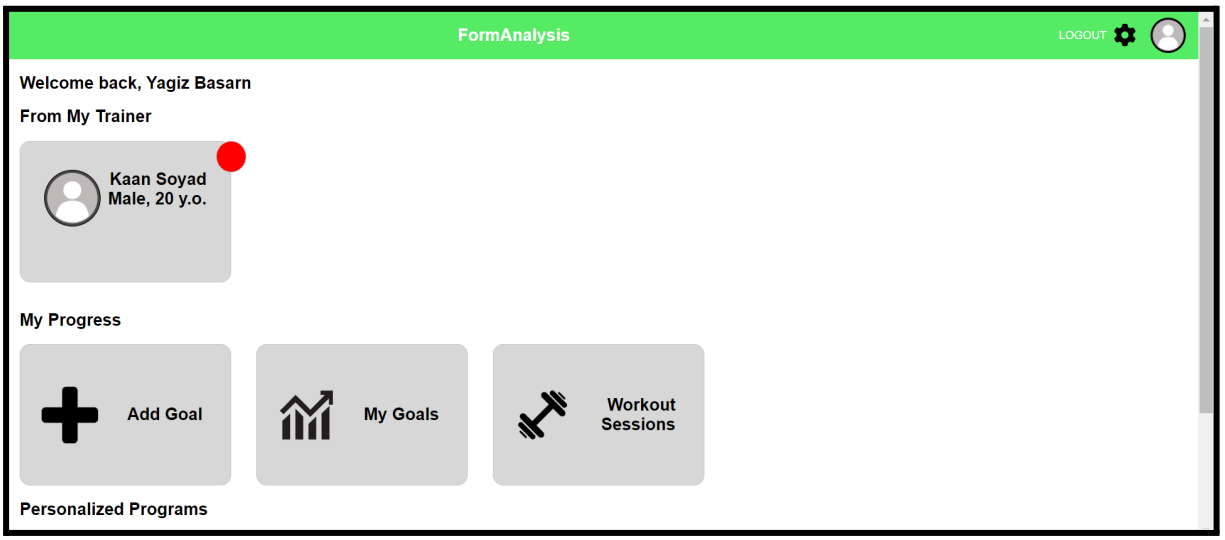
Sign Up Page

Specifying the first name, last name, email, birth date, matching passwords, gender and trainee or trainer button, users can sign up to the application.

Trainee Homepage



(If trainee does not have a trainer)

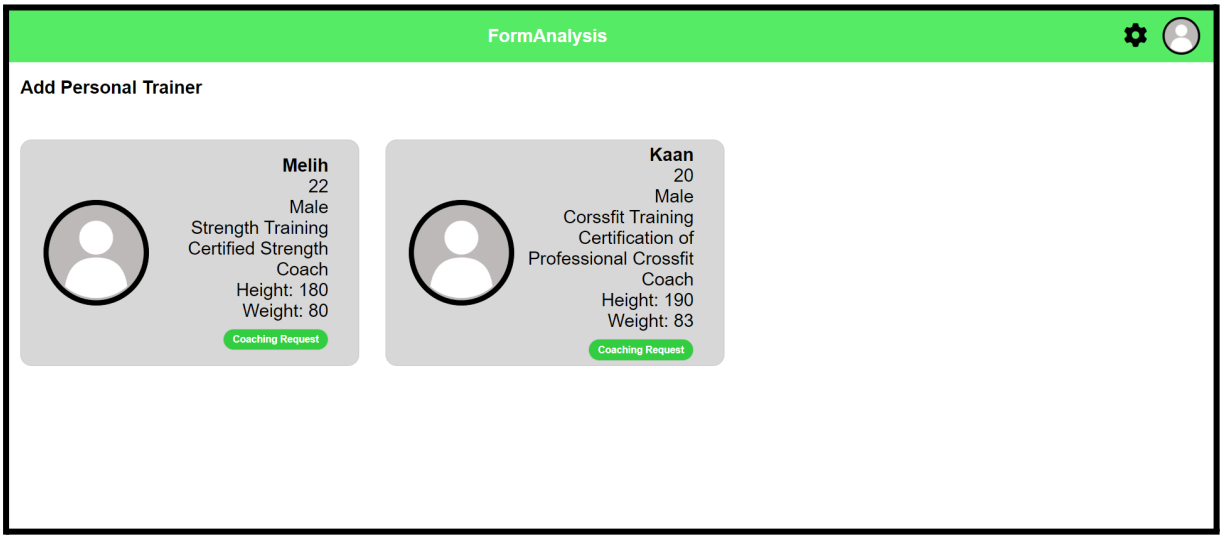


(If trainee has a trainer)

Trainees can click to the profile section on the upper right corner. Also they can click the name of trainer to go to trainer related menu. On “My Progress” section, trainees can add a goal to themselves, see their goals and see the workout sessions that are recommended by

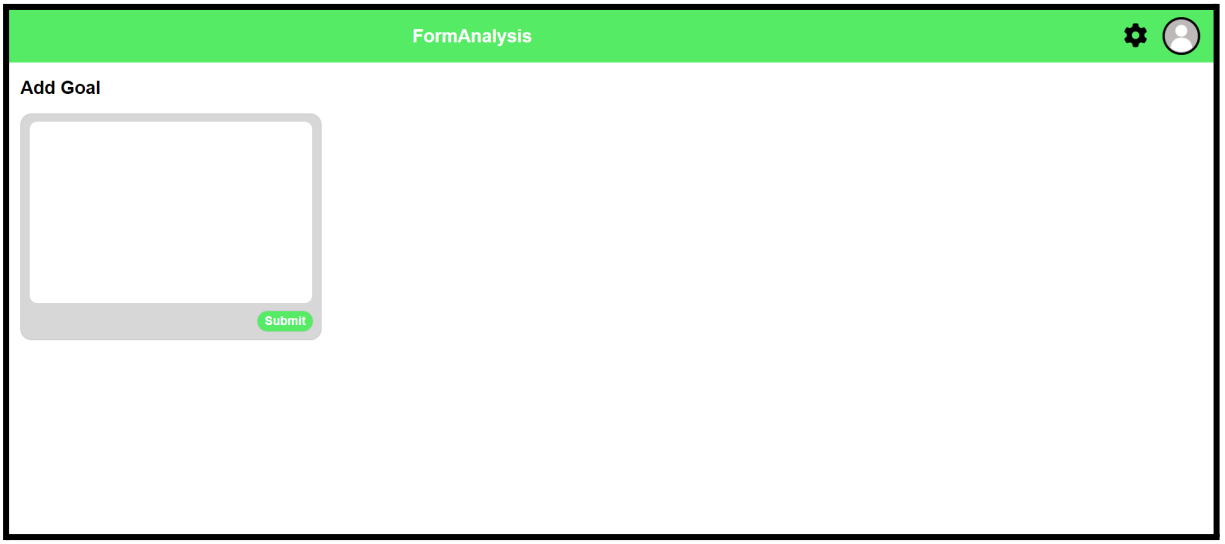
the system. On “Personalized Programs” section, users can see their personalized workout or nutrition plan by clicking on the buttons.

Add Personal Trainer Page



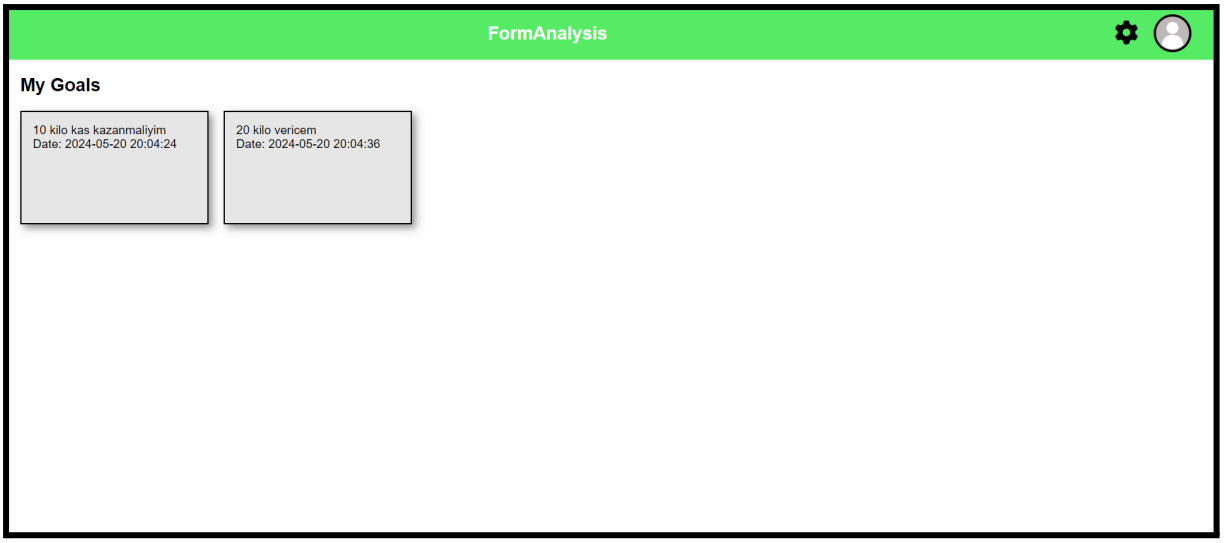
If a user does not have a trainer, they can send a coaching request to trainers by button. They can see the information related to each trainer.

Add Goal Page



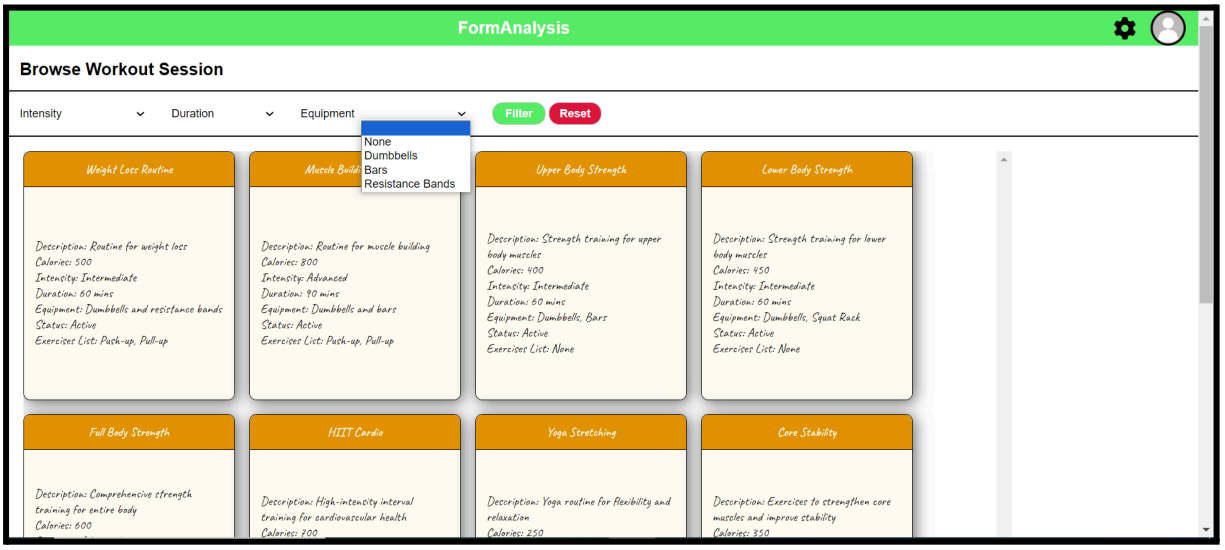
Trainees can write down a goal to themselves so that they will see them later to check their accomplishments.

My Goals Page



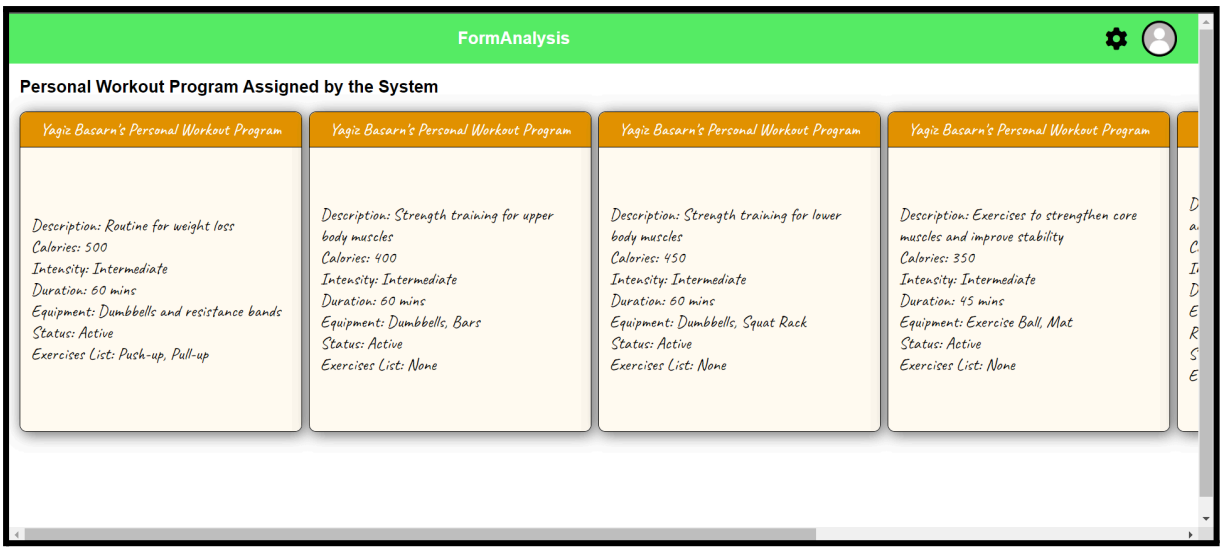
Trainees can see their past goals in this screen. This screen aims to help trainees track their fitness process.

Browse Workout Session Page



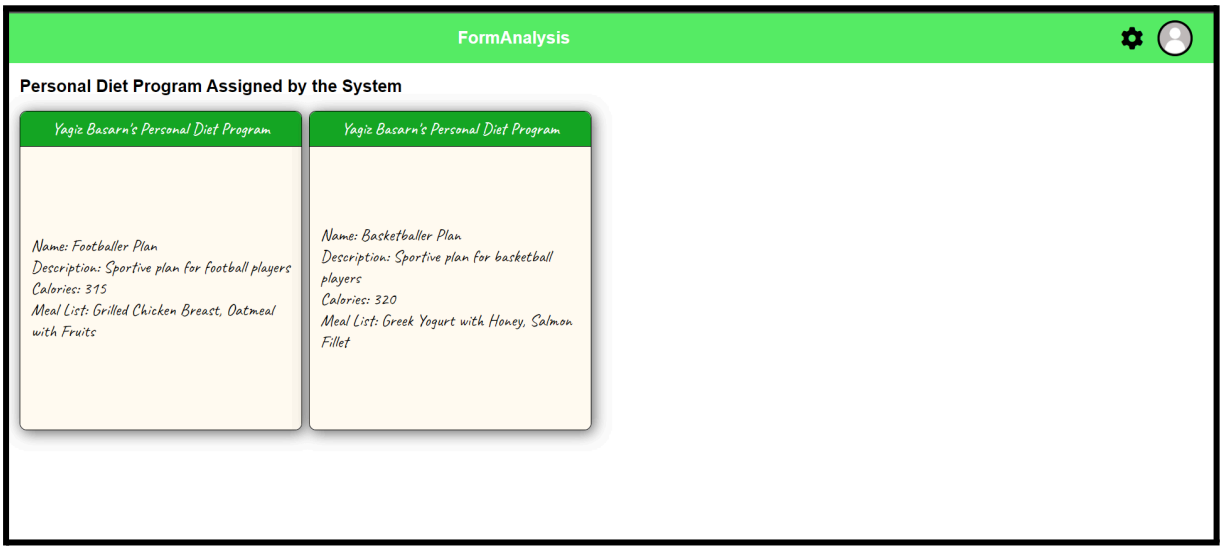
In this page, trainees can see the workout plans that are recommended by the system. By filters like intensity, duration and equipment, they can filter programs to choose one.

Personal Workout Program Page



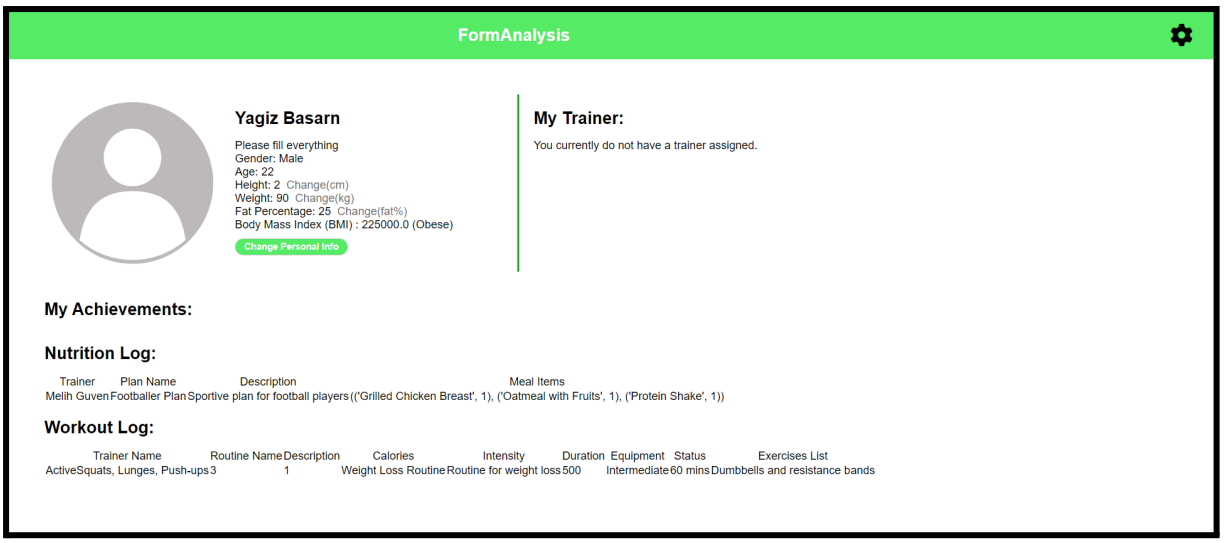
Trainees can see their workout programs automatically assigned by the system in this page. System assigns programs according to the users BMI (Body Mass Index), which is calculated by using the height and weight data of the user.

Personal Workout Program Page



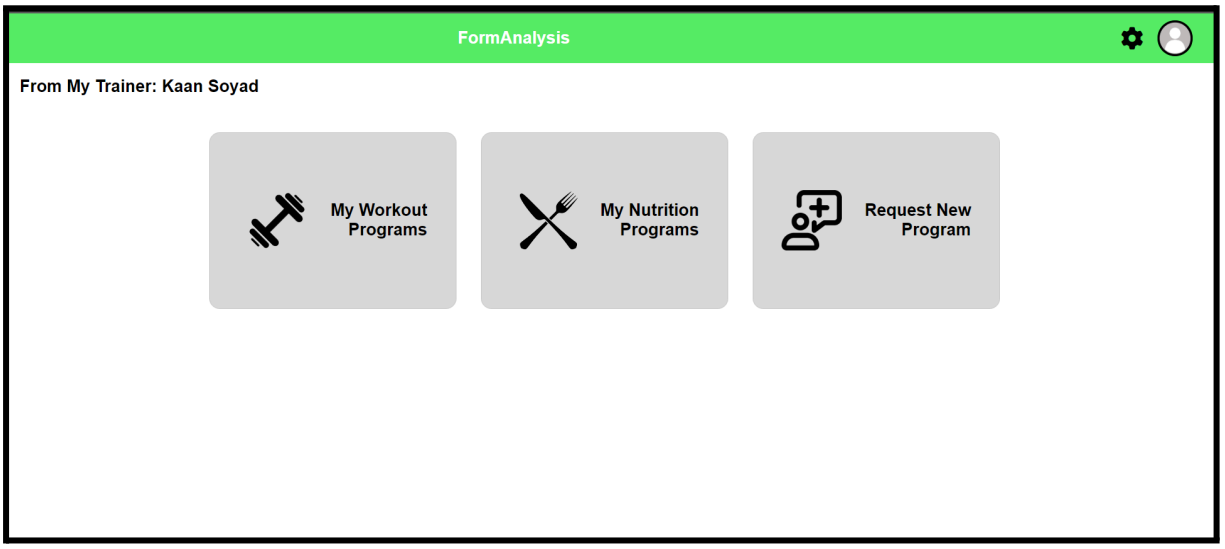
Trainees can see their nutrition programs automatically assigned by the system in this page. System assigns programs according to the users BMI (Body Mass Index), which is calculated by using the height and weight data of the user.

Trainee Profile Page



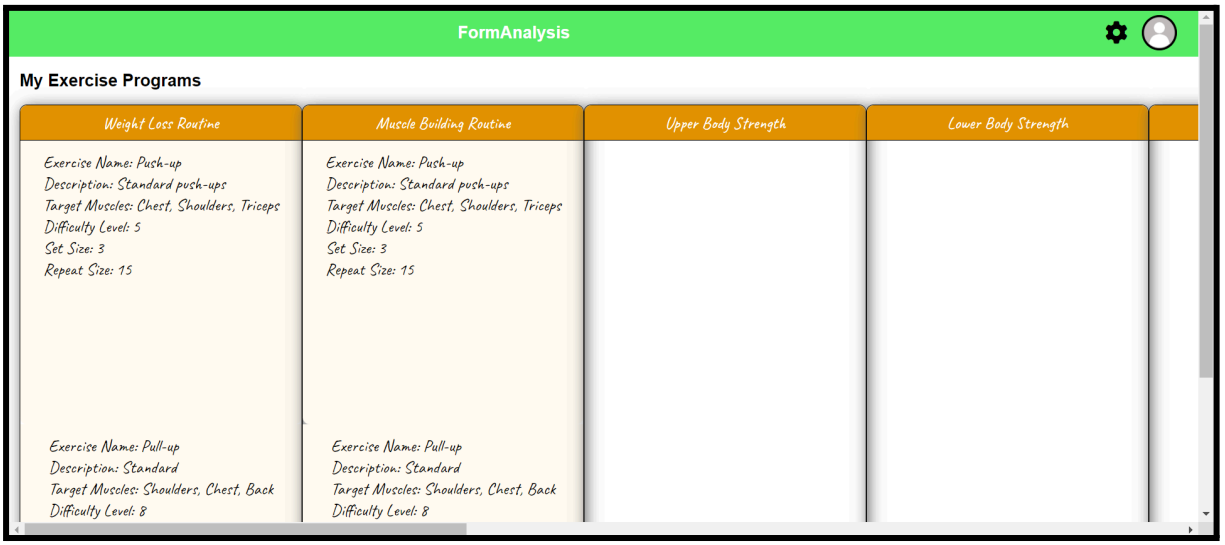
In the profile section of trainees, there is information that trainees can edit like height, weight, fat percentage. BMI is calculated using relevant data. In my trainer section, they can see their current trainer information. Also they can see their logs of workout and nutrition.

From My Trainer Page



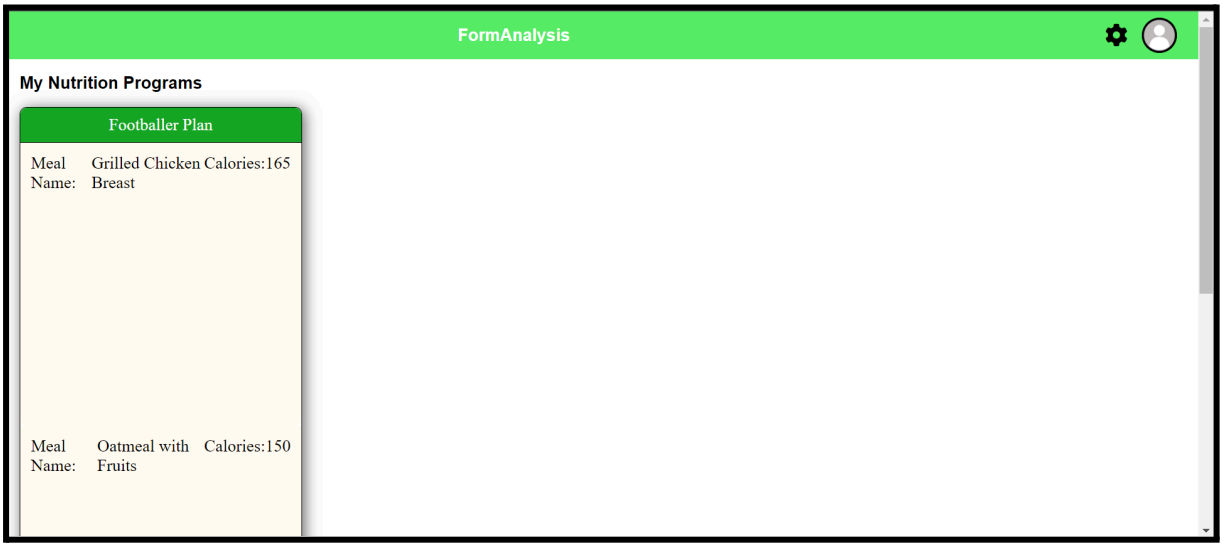
In this page, trainees can select a button corresponding to their needs. If a trainee wants to see the nutrition or workout programs that are assigned by their trainer, they can use two buttons on the left. If trainee needs a new program, it can use the request new program button.

My Exercise Programs Page



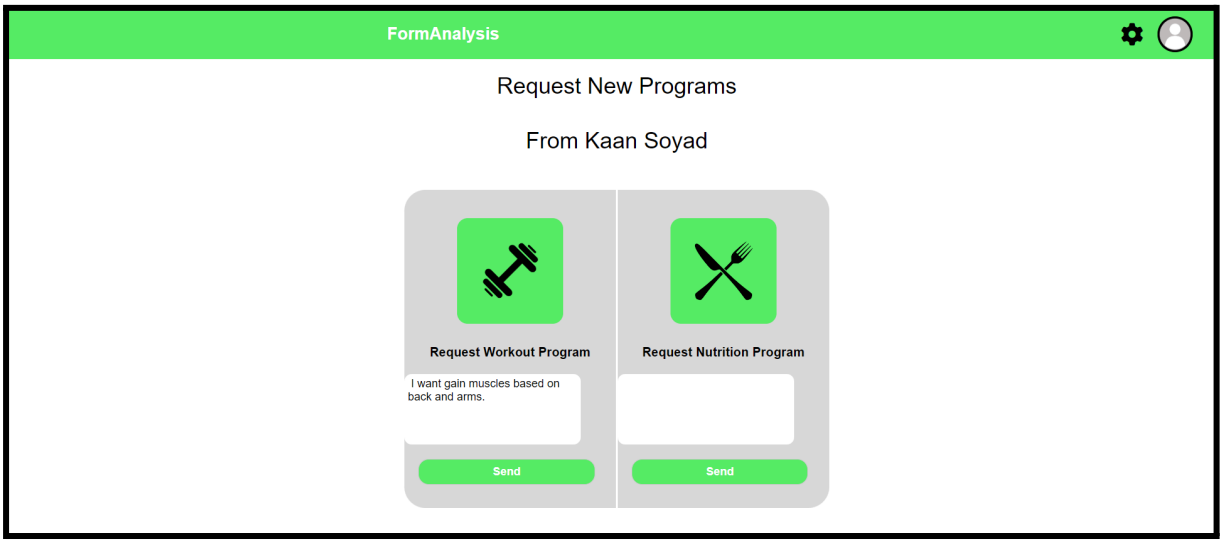
Here, trainees can see their exercise programs that are assigned by the trainer.

My Nutrition Programs Page



Here, trainees can see their nutrition programs that are assigned by their trainer.

Request New Programs Page

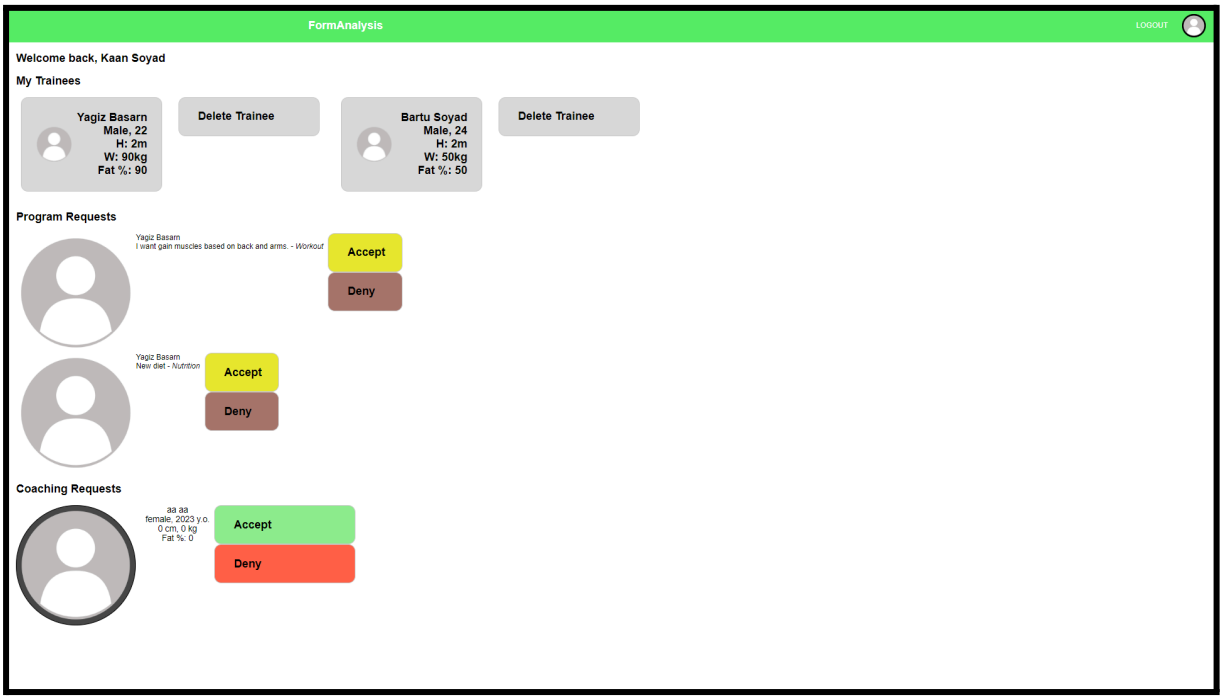


By specifying how they want their programs to be, trainees can send workout or nutrition program requests to their trainer.

Trainer Profile

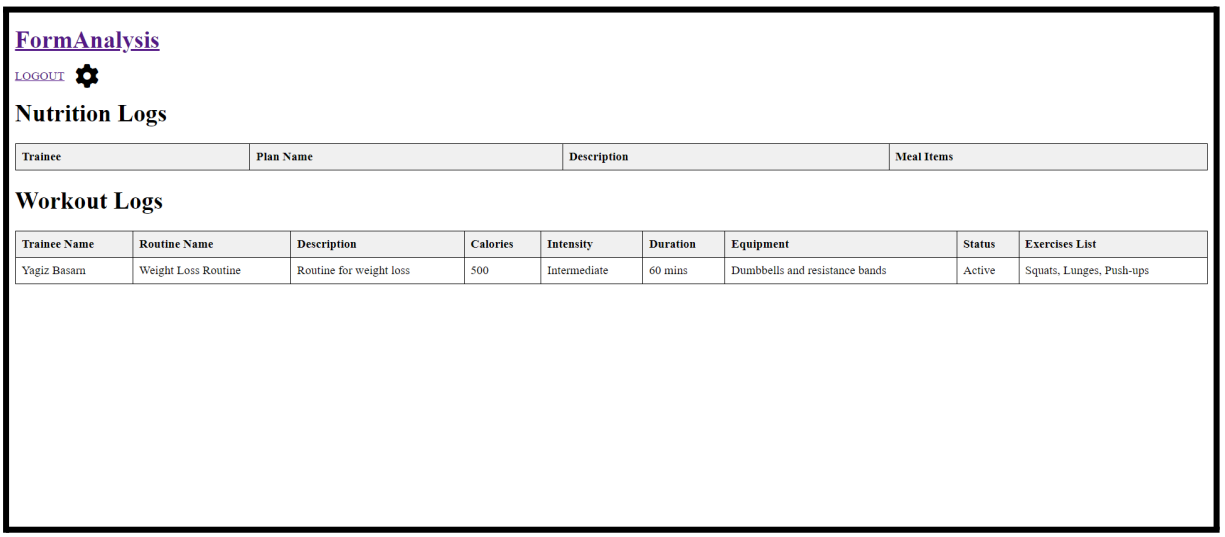
In the trainer profile page, trainers can change their height and weight to show the customers (trainees) and they can specify their speciality with the given certificate.

Trainer Homepage



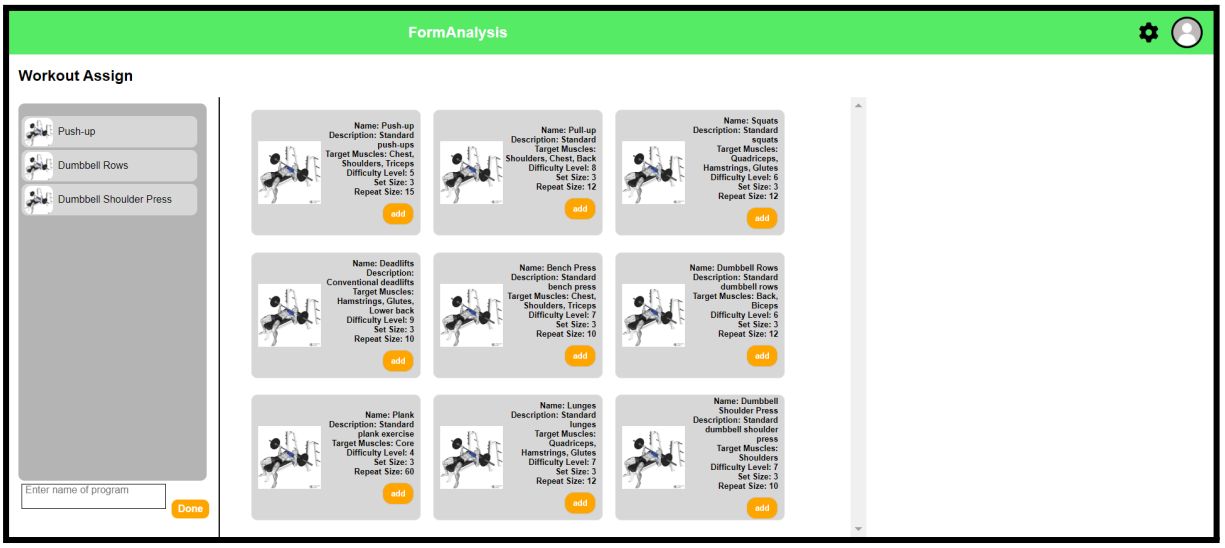
Trainers can see the new coaching requests on the coaching requests section. They can accept or deny them by using the buttons. Also program requests are displayed in the upper section. When they accept, the application will direct them to the new program assignment page. On my trainees section, trainers can delete their trainees or by clicking the names, they can see the logs of that trainee. By clicking the icon on top right, trainers can see their profile.

Trainee Log Page for Trainers:



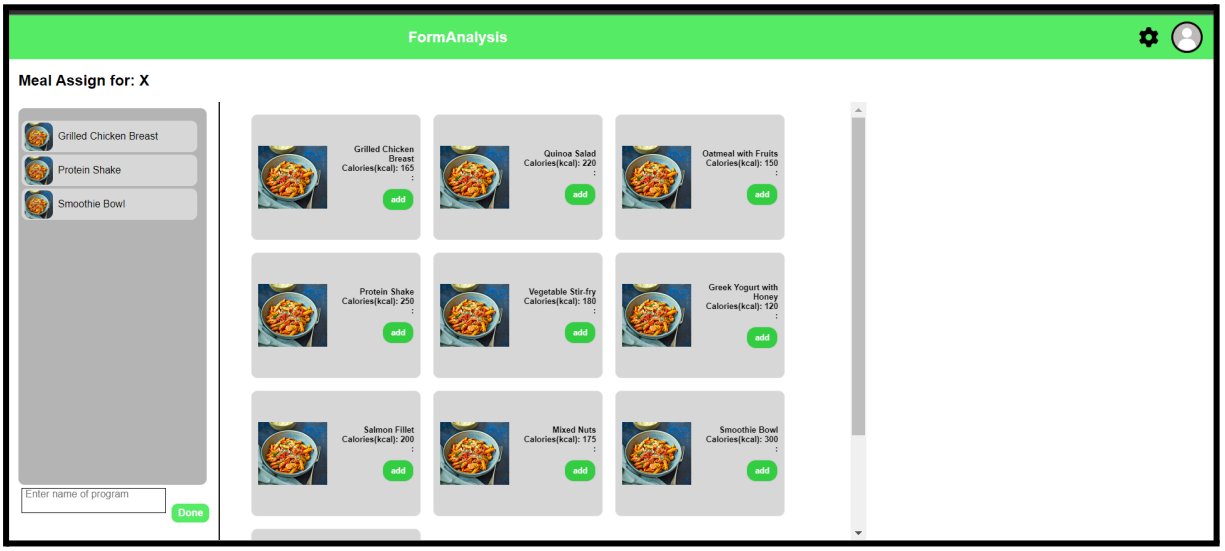
Trainers can see the logs of each trainee by clicking on their name. All workout and nutrition logs are displayed on this page.

Workout Assign Page for Trainers



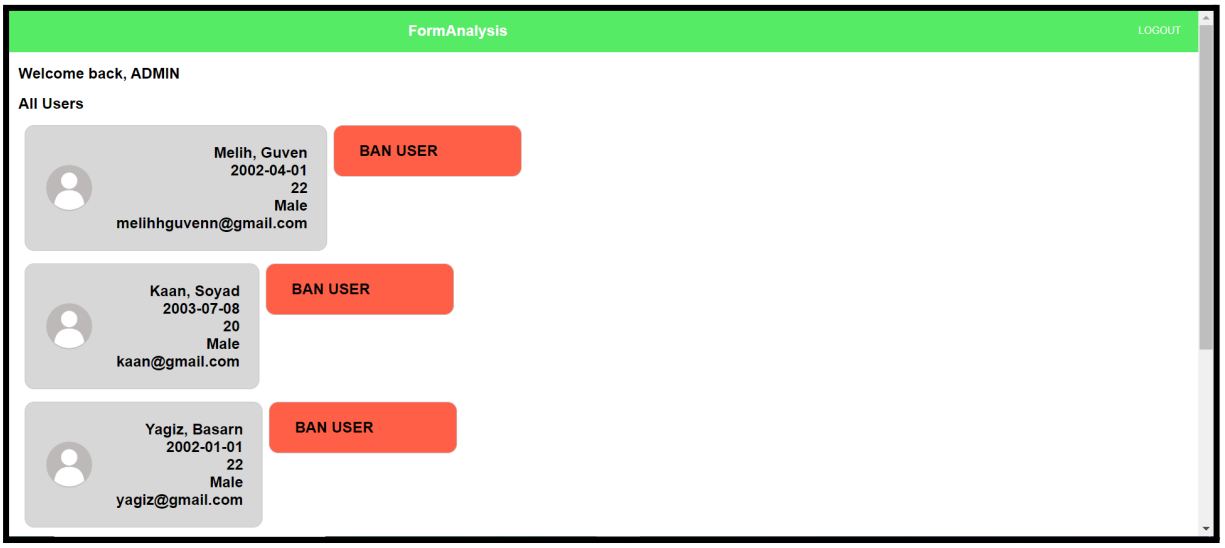
By clicking the exercises on the list and specifying the name of the program at the end, trainers can assign workout plans to the trainees.

Meal Assign Page for Trainers



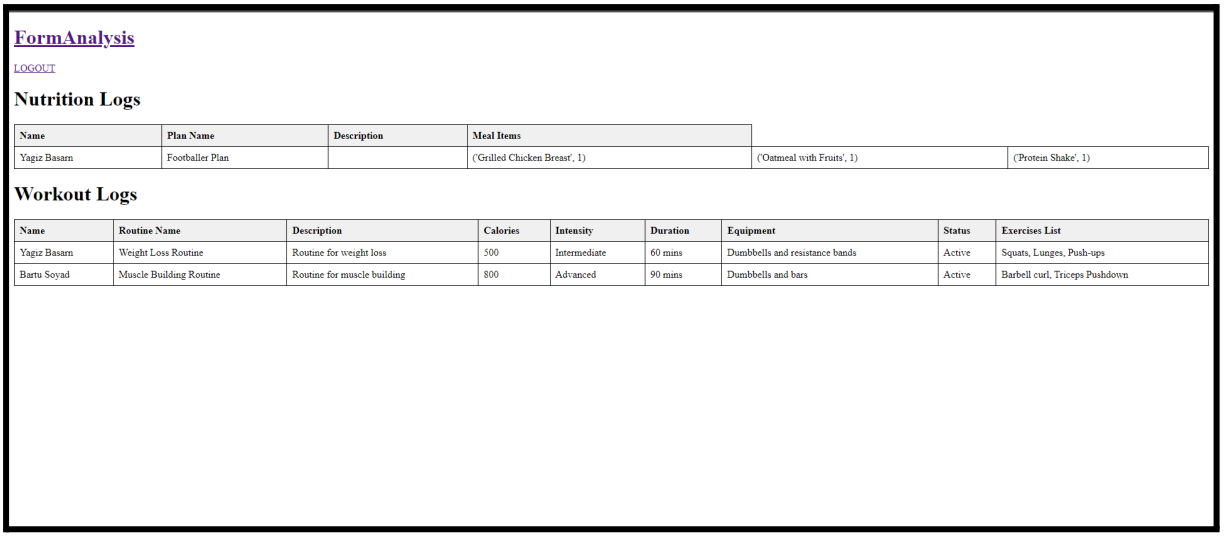
By clicking the meals on the list and specifying the name of the program at the end, trainers can assign nutrition plans to the trainees.

Admin Panel



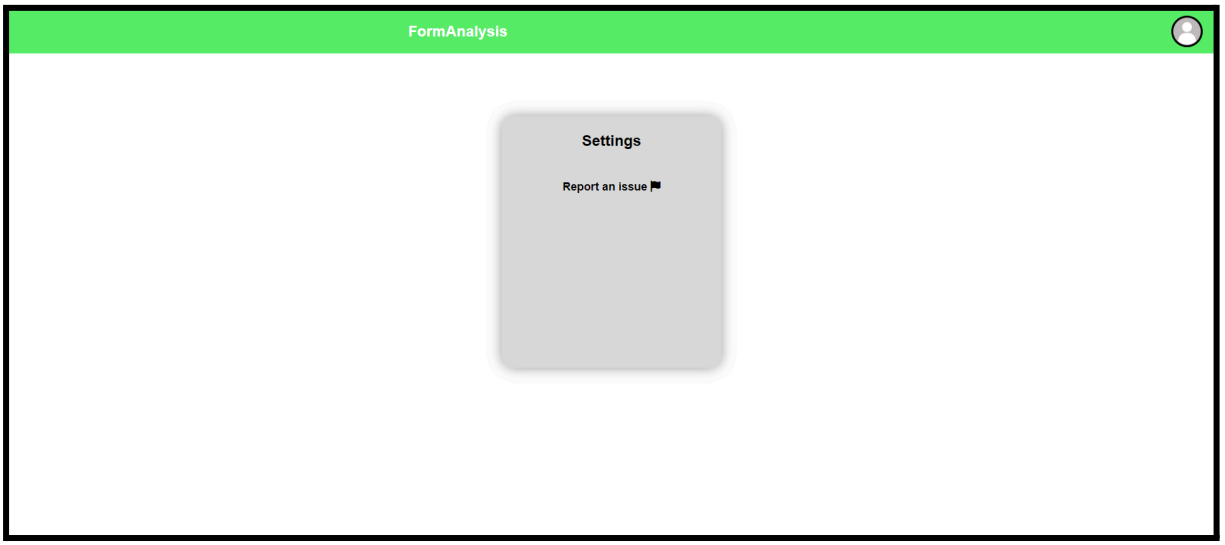
Admins can see all of the users in this homepage. They can ban users by clicking the corresponding button. Also admins can click on trainers to see the plans that they gave to trainees or click the trainees to see trainees' logs.

User Logs Screen for Admins



Admin can see all the plans given by all trainers to their trainees in this page. Logs of trainers can be displayed here.

Settings Screen



Users can report the issues/bugs they encounter via Email by using the “Report an issue” button on the settings page.