

**Faculty of Automation and Computer Science**

A picture containing font, graphics, logo, graphic design

Description automatically generated

Industrial Informatics Semester Project

2022 – 2023

Link: github.com/NTimea302/UpFit

**Team members:**

Nagy Timea | Team Leader

Nemes Raluca | Software Tester

Oprea Florin Octavian | Software developer

Szekely Rolland Iulian | Database developer

**Coordinating teacher:**

Eng. Teodora Sanislav

# Introduction (goal, objectives, functions)

UpFit is a fitness application designed to help individuals track their meals, monitor their calorie intake, and maintain a healthy lifestyle. The application caters to four different types of users: admin, coach, basic user, and premium user. Each user type has specific functionalities and privileges within the application.

The primary purpose of UpFit is to provide a platform where users can conveniently manage their nutrition and monitor their progress towards their fitness goals. The application offers features such as user registration, login functionality, meal tracking, meal history, and personalized calorie calculations based on user data such as weight, height, and activity level, UpFit calculates the recommended daily calorie intake.

Admins have full access and control over the application, including user management, food and food types management. Coaches users have the ability to post videos that can be viewed by premium users. Basic users have access to essential features such as meal tracking, meal history, and personalized calorie calculations. Premium users enjoy additional benefits and advanced features within the application such as viewing the vitamins they consumed.

UpFit enables users to add meals to their daily intake, allowing them to monitor their calorie consumption accurately. Users can select from a pre-defined list of foods from a datatable, making it easier to track the nutritional content of their meals.

# Application design

## UML diagrams – USE CASE

A picture containing text, diagram, line, font

Description automatically generated

## Database Design

A screenshot of a computer program

Description automatically generated with low confidence

The application has a well-structured database design to effectively manage data and support its various functionalities. The database consists of eight tables, each serving a specific purpose:

1. Admins Table:
   * Stores the credentials of administrators.
   * Contains columns such as ID (identifier), Username, and Password.
2. Coaches Table:
   * Stores the credentials of coaches.
   * Contains columns such as ID, Username, and Password.
3. Videos Table:
   * Stores information about uploaded videos.
   * Includes columns such as ID, Name (video title), Path (file path), and CoachFK (foreign key)
4. FoodTypes Table:
   * Stores different types of food categories.
   * Contains columns such as ID and Name (e.g., fruits, vegetables, meats, nuts, seeds).
5. Foods Table:
   * Stores detailed information about individual food items.
   * Includes columns such as ID, TypeFK (foreign key referencing the corresponding food type), Name, Calories, Proteins, Fats, Carbs, Fibers, and various vitamin fields (A, B, C, D, E).
6. MealTypes Table:
   * Stores different meal types, such as breakfast, lunch, and dinner.
   * Contains columns such as ID and Name.
7. Meals Table:
   * Represents individual meal instances recorded by users.
   * Includes columns such as ID, MealtypeFK (foreign key referencing the meal type), UserFK (foreign key referencing the associated user), FoodFK (foreign key referencing the consumed food), Quantity, and Date.

## Graphical user interface DEsign and functionalities

Upon launching the UpFit application, users are presented with an informative home screen, providing a general overview of the application's features and functionalities. The home screen serves as the starting point for users to access the login and registration functionalities.

Users have the option to log in to their existing accounts by entering their credentials, such as their username and password. Once logged in, users can view a table of their consumed nutrients for the day and have the option to add new meals by accessing the ‘Add Meal’ button and selecting a food type, a food from the database and entering the consumed amount.

For new users who have not yet registered, the application offers a registration process. Users can create a new account by choosing their preferred subscription type option and by providing necessary personal details such as their name, username, password and other details about their lifestyle and other relevant data that helps in determining personalized fitness goals and recommendations, such as height, weight and age. Upon successful registration, users can proceed to log in.

In addition to regular users, the application offers access to admins and coaches too. These type of users have their own login functionality. Admins have comprehensive access and control over the system, including user and food management. Coaches can upload videos for users.

##### Starting page of UpFit!

A screenshot of a website

Description automatically generated with low confidence

##### Users gui

The graphical user interface (GUI) for both basic users and premium users in the UpFit application shares a similar design and layout. However, premium users are granted additional options and features compared to basic users such as accessing videos related to exercises and tutorials, as well as being able to view detailed information about the vitamin content in their meals.

The differentiation between basic and premium users lies primarily in the availability of specific features and access to premium content, while the core GUI remains unchanged.

###### Log in and Register of User

A screenshot of a login page

Description automatically generated with medium confidenceA close-up of a login form

Description automatically generated with low confidence

###### PREMIUM and basic USER ACCOUNT

After logging in, the user is directed to either their premium or basic user account. In this section, the user is presented with essential information regarding their daily necessary intake for maintenance.

The user interface displays the following details:

* A message indicating that their daily necessary intake has been calculated based on the provided information.
* A goal recommendation for the user, indicating the target calorie intake per day, displayed as "@kcaldaily kcal".
* Information on the remaining calories for the user, presented as "You have left: @remainedKcal kcal".

Furthermore, the user has the option to add a new meal. By selecting this option, the user can input the details of the food and quantity they consumed. The added meal is then included in the user's daily nutrient list, which is presented in a tabular format.

The daily nutrient list contains the following columns:

* Nutrient: Displays the name of the nutrient, including "Calories consumed", "Protein", "Carbohydrates", "Fats", and "Fiber".
* Total Amount: Indicates the respective total amount of each nutrient consumed, retrieved from the "@macrodaily" variable.

The user can refer to this nutrient table to track their daily intake and monitor their nutritional values accordingly.

A picture containing text, screenshot, footwear, multimedia software

Description automatically generated A picture containing text, screenshot, footwear, multimedia software

Description automatically generated

##### Admin GUI

Upon successful login, the admin has access to four distinct management sections in the UpFit application to perform various administrative tasks and maintain the system efficiently. The four management sections available to the admin are as follows:

1. User Management:
   * In the "Manage Users" section, the admin can add new users to the system, view the existing list of users, view the details of each user, and delete user accounts as necessary.
2. Admin Management:
   * The "Manage Admins" section allows the admin to create new admin accounts or delete existing admin accounts as required.
3. Food Type Management:
   * In the "Manage Food Types" section, the admin has the ability to create new food types, edit existing food types, or remove food types from the system.
4. Food Management:
   * Within the "Manage Foods" section, the admin can create new food entries, edit the details of existing food items, or remove food items from the application.
   * This feature allows the admin to curate a database of foods that users can choose from when tracking their meals, ensuring accurate nutritional information and a diverse selection.

###### Log in and Starting Page of Admin

A screenshot of a login page

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

###### USER MANAGEMENT

A screenshot of a computer

Description automatically generated with medium confidence

###### User details and deletion interface for admin

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

The interface for managing admins (viewing admins list and deleting admins) and for managing users (viewing users list, user details, and deleting users) shares a unified design, making it easy for admins to perform tasks such as viewing lists and deleting accounts. The consistent interface ensures a seamless user experience and simplifies administrative management.

###### Food and food types management user interfaces

A screenshot of a menu

Description automatically generated with low confidence

A screenshot of a food menu

Description automatically generated with low confidence

##### COACH GRAPHICAL USER INTERFACE

The graphical user interface (GUI) designed for coaches in the UpFit application provides convenient access to uploaded videos and allows coaches to upload new videos directly from their devices.

Within the coach interface, coaches can view a list or gallery displaying the videos that have been uploaded so far. This provides coaches with a visual overview of the available video content for their users.

By offering these features within the coach GUI, UpFit empowers coaches to manage and expand the available video content, ensuring a rich and diverse library of resources for users.

A screenshot of a video file

Description automatically generated with low confidence

# Application Implementation

The implementation of the UpFit application involved transforming the design and the ideas into an actual code and integrating the mentioned functionalities. The application was developed in Visual Studio 2022 using the ASP.net Core Web App (Model-View-Controller) template. We started by defining the necessary classes/models. We used Entity Framework and code-first approach. After we defined the models, we generated a database schema based on these classes. We implemented the controllers to handle user requests and actions, and designed the views to provide a user-friendly interface.

## Structure of application

The UpFit application follows the Model-View-Controller (MVC) architectural pattern, which provides a structured approach to organizing and separating different components. The application consists of several folders, each serving a specific purpose:

1. **Controllers**: This folder contains the controllers that handle user requests, process data, and control the flow of the application. Controllers interact with the models and views to perform various actions.
2. **Migrations**: The Migrations folder stores database migration files generated by Entity Framework. These files track changes to the database schema over time, allowing for easy database updates and version control.
3. **Models**: The Models folder holds the model classes that represent the entities and data structures within the application. These classes define properties, relationships, and business logic related to the data.
4. **Views**: The Views folder contains the user interface templates that display information to the users. Views are responsible for rendering data and presenting it in a visually appealing and interactive manner.
5. **VideoFiles**: This folder is specifically dedicated to storing video files used within the application. It will contain videos related to exercises, tutorials, or any other multimedia content.

In the provided pictures, the model classes are illustarted. These classes represent the various entities and their relationships, providing a visual representation of the application's data structure.

By organizing the application into these distinct folders and adopting the MVC pattern, UpFit ensures modularity, separation of concerns, and a clear structure that facilitates maintenance and extensibility.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

## IMPLEMENTATION FOR daily CALORIE INTAKE CALCULATOR

The code snippet provided represents the implementation for the daily calorie intake calculator within the Register action of the Users Controller.

A group of people on a white background

Description automatically generated with low confidence

To enhance personalization, the application allows users to select their lifestyle level, which can be sedentary, medium, active, or very active at registration. These lifestyle levels are mapped to corresponding numeric values in the code, with sedentary being assigned a value of 1.2, medium as 1.375, active as 1.55, and very active as 1.725.

Based on the user's gender and selected lifestyle level, the code performs the necessary calculations using the Harris-Benedict equation. For male users (indicated by a Gender value of "M"), the equation for men is utilized. Conversely, for female users, the equation for women is applied. The resulting calorie intake is then assigned to the user's KcalDaily property.

By incorporating this code, the UpFit application accurately calculates each user's recommended daily calorie intake based on their gender and selected lifestyle level. This personalized approach ensures that users receive tailored guidance to support their health and fitness goals.

## CALCULATING NUTRIENTS FROM MEALS

A screenshot of a computer

Description automatically generated with low confidence

The below provided code snippet represents the implementation of the "MacroCalculations" action within the Users Controller. This action is accessed via an HTTP GET request and takes a user ID as a parameter.

Inside the action, the code retrieves the current date and initializes variables for storing the calculated nutrient values. It then queries the database to iterate through each meal entry associated with the specified user ID. For each meal, the code retrieves the corresponding food item from the "foods" table and calculates the nutrient values based on the meal's quantity.

The calculated nutrient values, such as calories, carbohydrates, proteins, fats, fibers, and various vitamins, are accumulated and stored in a list called "totalmacro." This list is then passed to the "LoggedIn" view using the ViewBag.

The "LoggedIn" view can access these nutrient values and display them accordingly using the code snippet "@using (Html.BeginForm("MacroCalculations"))".

In summary, the "MacroCalculations" action calculates the nutrient values from the user's meals and makes them available in the "LoggedIn" view, allowing users to view their daily nutrient intake.

A screenshot of a computer program

Description automatically generated with low confidence

## User Model

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace UpFit\_\_main.Models

{

public class User

{

[Key]

public int UserID { get; set; }

[Required(ErrorMessage = "Subscription Type is required.")]

[DisplayName("Subscription Type")]

public int SubscriptionTypeFK { get; set; }

[Required(ErrorMessage = "Username is required.")]

[RegularExpression(@"^[a-zA-Z0-9\_]+$", ErrorMessage = "Username can only contain letters, numbers, and underscores.")]

public string UserName { get; set; }

[DataType(DataType.Password)]

[Required(ErrorMessage = "Password is required.")]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

public string Password { get; set; }

[Required(ErrorMessage = "First Name is required.")]

[RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "First Name can only contain letters.")]

public string FirstName { get; set; }

[Required(ErrorMessage = "Last Name is required.")]

[RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "Last Name can only contain letters.")]

public string LastName { get; set; }

[RegularExpression("^[FM]$", ErrorMessage = "Gender must be 'F' or 'M'.")]

[Required(ErrorMessage = "Gender is required.")]

public string Gender { get; set; }

[Range(1, 150, ErrorMessage = "Age must be between 1 and 150.")]

public int Age { get; set; }

[Range(50, 300, ErrorMessage = "Height must be between 50 and 300.")]

public int Height { get; set; }

[Range(10, 300, ErrorMessage = "Weight must be between 10 and 300.")]

public int Weight { get; set; }

[Range(0, int.MaxValue, ErrorMessage = "Kcal Daily must be a positive value.")]

public int KcalDaily { get; set; }

public double Lifestyle { get; set; }

}

}

## Food model

namespace UpFit\_\_main.Models

{

public class Food

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int foodID { get; set; }

[Display(Name = "What type of food is?")]

[Required(ErrorMessage = "Type Required")]

public int type { get; set; }

[Display(Name = "Name")]

[Required(ErrorMessage = "Name Required")]

public string name { get; set; }

[Display(Name = "Calories")]

[Required(ErrorMessage = "Calories Required")]

public int calories { get; set; }

[Display(Name = "Proteins")]

[Required(ErrorMessage = "Proteins Required")]

public double? proteins { get; set; }

[Display(Name = "Fats")]

[Required(ErrorMessage = "Fats Required")]

public double? fats { get; set; }

[Display(Name = "Carbohydrates")]

[Required(ErrorMessage = "Carbs Required")]

public double? carbs { get; set; }

[Display(Name = "Fibers")]

[Required(ErrorMessage = "Fibers Required")]

public double? fibers { get; set; }

[Display(Name = "Vitamin A")]

[Required(ErrorMessage = "Vitamin A Required")]

public double? vitamin\_A { get; set; }

[Display(Name = "Vitamin B")]

[Required(ErrorMessage = "Vitamin B Required")]

public double? vitamin\_B { get; set; }

[Display(Name = "Vitamin C")]

[Required(ErrorMessage = "Vitamin C Required")]

public double? vitamin\_C { get; set; }

[Display(Name = "Vitamin D")]

[Required(ErrorMessage = "Vitamin D Required")]

public double? vitamin\_D { get; set; }

[Display(Name = "Vitamin E")]

[Required(ErrorMessage = "Vitamin E Required")]

public double? vitamin\_E { get; set; }

}

## Index page front-end

<main aria-labelledby="title">

<div class="d-flex justify-content-center align-items-center flex-column" style="background-color: rgba(255,255,255,0.3); ">

<div id="myCarousel" class="carousel slide" data-ride="carousel">

<!-- Indicators -->

<ol class="carousel-indicators">

<li data-target="#myCarousel" data-slide-to="0" class="active"></li>

<li data-target="#myCarousel" data-slide-to="1"></li>

<li data-target="#myCarousel" data-slide-to="2"></li>

</ol>

<!-- Wrapper for slides -->

<div class="carousel-inner">

<div class="item active">

<div class="slider">

<img src="~/Content/poza1.jpg" alt="Image 1">

<img src="~/Content/poza.jpg" alt="Image 2">

<img src="~/Content/poza2.jpg" alt="Image 3">

</div>

</div>

</div>

</div>

<div class="px-md-5" style="padding: 50px; text-align: center;">

UpFit! is a fitness application that helps users track their workouts, set fitness goals, and monitor their progress with its user-friendly interface and comprehensive tracking tools. UpFit! is a great choice for anyone looking to take control of their fitness journey.

</div>

<div class="login-register">

<h3 style="color: white;">Login or Register to use the application!</h3>

<div>

@Html.ActionLink("Log In", "Login", "Users", new { area = "" }, new { @class = "btn btn-primary" })

@Html.ActionLink("Register", "Register", "Users", new { area = "" }, new { @class = "btn btn-primary" })

</div>

<div style="padding: 50px; ">

<h3 style="color: white;">Are you a coach or admin?</h3>

<div>

@Html.ActionLink("Log In as Coach", "Login", "Coach", new { area = "" }, new { @class = "btn btn-primary" })

@Html.ActionLink("Log In as Admin", "Login", "Admins", new { area = "" }, new { @class = "btn btn-primary" })

</div>

</div>

</div>

</div>

</main>

## Posting videos

The provided code snippet showcases two methods within the **Coach Controller**: "AddVideo" and "VideoList". The "AddVideo" method is an HTTP POST action that handles the **uploading of a video file**. The "VideoList" method is an HTTP GET action that **retrieves a list of videos from the database**. It also creates a new instance of the "CodeFirstDb" context. These methods work together to handle video uploads and display the list of uploaded videos for the coach interface.

[HttpPost]

public ActionResult AddVideo(HttpPostedFileBase video)

{

using (CodeFirstDb db = new CodeFirstDb())

{

if (video.FileName != null)

{

video.SaveAs(Server.MapPath("/Videofiles/" + video.FileName));

Video uploadVideo = new Video();

{

uploadVideo.Vname = video.FileName.Split('.')[0];

uploadVideo.Vpath = "/Videofiles/" + video.FileName;

};

db.videos.Add(uploadVideo);

db.SaveChanges();

return RedirectToAction("VideoList");

}

}

return View("VideoList");

}

[HttpGet]

public ActionResult VideoList()

{

using (CodeFirstDb db = new CodeFirstDb())

{

List<Video> Videolist = new List<Video>();

foreach (Video video in db.videos)

{

Video dbVideo = new Video();

{

dbVideo.Vname = video.Vname;

dbVideo.Vpath = video.Vpath.ToString();

};

Videolist.Add(dbVideo);

}

return View(Videolist);

}

}

## important sections from admin controller

// POST: Admins/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "AdminID,UserName,Password")] Admin admin)

{

if (ModelState.IsValid)

{

db.admins.Add(admin);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(admin);

}

// GET: Admins/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Admin admin = db.admins.Find(id);

if (admin == null)

{

return HttpNotFound();

}

return View(admin);

}

// POST: Admins/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "AdminID, UserName, Password")] Admin admin)

{

if (ModelState.IsValid)

{

db.Entry(admin).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(admin);

}

// POST: Admins/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Admin admin = db.admins.Find(id);

db.admins.Remove(admin);

db.SaveChanges();

return RedirectToAction("Index");

}

// POST: Admins/AddUser

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult AddUser(User user)

{

if (ModelState.IsValid)

{

db.users.Add(user);

db.SaveChanges();

return RedirectToAction("ManageUsers");

}

return View(user);

}

// GET: Admins/DeleteUser/5

public ActionResult DeleteUser(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

User user = db.users.Find(id);

if (user == null)

{

return HttpNotFound();

}

return View(user);

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult CreateFoods(Food food)

{

if (ModelState.IsValid)

{

var foodType = db.foodTypes.FirstOrDefault(ft => ft.ID\_Type == food.type);

if (foodType != null)

{

food.type = foodType.ID\_Type;

db.foods.Add(food);

db.SaveChanges();

return RedirectToAction("ManageFoods");

}

else

{

ModelState.AddModelError("", "Invalid food type.");

}

}

ViewBag.FoodTypes = db.foodTypes.ToList();

return View(food);

}

## IMPORTANT SECTIONS FROM USERS CONTROLLER

[HttpPost]

public ActionResult Login(User user)

{

using (CodeFirstDb db = new CodeFirstDb())

{

var usr = db.users.SingleOrDefault(u => u.UserName == user.UserName && u.Password == user.Password);

if (usr != null)

{

Session["UserID"] = usr.UserID.ToString();

Session["UserName"] = usr.UserName.ToString();

Session["SubscriptionType"] = usr.SubscriptionTypeFK.ToString();

Session["KcalDaily"] = usr.KcalDaily;

return RedirectToAction("LoggedIn");

}

else

{

ModelState.AddModelError("", "Incorrect username or password.");

}

}

return View();

}

public ActionResult LoggedIn()

{

if (Session["UserID"] != null)

{

int userId = Convert.ToInt32(Session["UserID"]);

MacroCalculations(userId);

// Retrieve the user from the database based on the UserID

User user = db.users.Find(userId);

if (user != null)

{

return View(user);

}

}

// If the user is not logged in or not found in the database, redirect to the login

return RedirectToAction("Login");

}

// POST: Users/Register

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Register([Bind(Include = "UserID,SubscriptionTypeFK,UserName,Password,FirstName,LastName,Gender,Age,Height,Weight,KcalDaily,Lifestyle")] User user)

{

if (user.Gender == "M")

{

user.KcalDaily = (int)Math.Round(user.Lifestyle \* (66 + (13.7 \* user.Weight) + (5 \* user.Height) - (6.8 \* user.Age)));

}

else

{

user.KcalDaily = (int)Math.Round(user.Lifestyle \* (65 + (9.5 \* user.Weight) + (1.8 \* user.Height) - (4.7 \* user.Age)));

}

if (ModelState.IsValid)

{

db.users.Add(user);

db.SaveChanges();

return RedirectToAction("Login");

}

return View(user);

}

[HttpGet]

public ActionResult VideoList()

{

using (CodeFirstDb db = new CodeFirstDb())

{

List<Video> Videolist = new List<Video>();

foreach (Video video in db.videos)

{

Video dbVideo = new Video();

{

dbVideo.Vname = video.Vname;

dbVideo.Vpath = video.Vpath.ToString();

};

Videolist.Add(dbVideo);

}

return View(Videolist);

}

}

## Db Context

The provided code snippet represents the definition of a class named "CodeFirstDb" within the "UpFit\_\_main.Models" namespace. This class inherits from the "DbContext" class, which is a part of Entity Framework.

Within the "CodeFirstDb" class, there are several properties defined, each corresponding to a table in the database. These properties use the "DbSet<T>" class to represent database tables, where "T" is the corresponding model class for each table.

The properties include "users" for storing user information, "admins" for storing admin credentials, "foodTypes" for storing types of food, "foods" for storing detailed information about specific foods, "mealTypes" for storing types of meals (e.g., breakfast, lunch, dinner), "meals" for storing individual meal records, "coaches" for storing coach information, and "videos" for storing video details.

By defining these properties within the "CodeFirstDb" class, it allows the application to interact with the corresponding database tables using Entity Framework's capabilities, such as querying, inserting, updating, and deleting data.

Overall, the "CodeFirstDb" class acts as a bridge between the application and the underlying database, providing access to the necessary tables and facilitating data operations.

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace UpFit\_\_main.Models

{

public class CodeFirstDb : DbContext

{

public DbSet<User> users { get; set; }

public DbSet<Admin> admins { get; set; }

public DbSet<FoodType> foodTypes { get; set; }

public DbSet<Food> foods { get; set; }

public DbSet<MealType> mealTypes { get; set; }

public DbSet<Meal> meals { get; set; }

public DbSet<Coach> coaches { get; set; }

public DbSet<Video> videos { get; set; }

}

}

## Database implementation

The database was automatically generated using migrations, provided by **Entity Framework**. We set the “**Automatic Migrations**“ property to true. With automatic migrations enabled, whenever there are changes to the model classes, such as adding new entities, modifying properties, or updating relationships, Entity Framework automatically generates the necessary migration scripts. This process allows for seamless updates and changes to the database schema as the application evolves. To ensure the integrity and accuracy of the database, it was accessed and verified using **SQL Server Management Studio** (SSMS).

It's important to note that the **Connection string** used to establish a connection between the application and the database needs to be updated on each computer where the application is deployed or run. The connection string contains information such as the server name, authentication details, and database name. By updating the connection string, the application can establish the necessary connection to the correct database instance.

##### Foods table

A screenshot of a computer

Description automatically generated with medium confidence

##### Users table

A screenshot of a computer

Description automatically generated with medium confidence

# Application Testing

The testing phase of the UpFit application begins with thorough documentation to ensure a structured and organized approach. The testing objectives are clearly defined, encompassing the verification of functionality, performance, and reliability. The scope of testing outlines the specific areas of the application that will be tested, including key features, user roles, and critical workflows.

For this manual testing approach, a comprehensive set of test cases is prepared, covering various scenarios and functionalities. Each test case includes detailed steps, expected results, and any necessary test data or prerequisites. Additionally, comments are provided for reporting and tracking bugs or issues encountered during testing.

Throughout the testing phase, the assigned testers follow the defined test execution plan, adhering to the timeline and documenting any identified issues. The testing process aims to deliver a comprehensive test report summarizing the results, highlighting any issues discovered, and offering recommendations for improvement.







# Conslusion

In conclusion, UpFit is more than just a fitness application. It embodies a vision of empowerment, a commitment to supporting individuals on their journey towards a healthier lifestyle. With its user-friendly interface and comprehensive features, UpFit strives to inspire users to take charge of their well-being and make informed choices about their diet and fitness.

The application's ability to track meals, monitor calorie intake, and help users achieve their fitness goals goes beyond mere functionality. It resonates with the deep desire we all have to live healthier, more fulfilling lives. UpFit serves as a trusted companion, providing the tools and information needed to transform aspirations into tangible results.

Through its intuitive design and robust capabilities, UpFit aims to instill a sense of motivation, encouraging users to make positive changes and embrace a balanced approach to their health. It aspires to be the catalyst that empowers individuals to unlock their full potential and experience the transformative power of a healthy lifestyle.

UpFit is more than just an application; it's a partner on the journey towards personal well-being, offering support, guidance, and the belief that each individual has the capacity to achieve their goals and live their best life.