Software Engineering Department

Braude - College of Engineering

Capstone Project Phase B – 61999

Diagnosing early pediatric failure to thrive (FTT)

GitHub Link: https://github.com/EhsanSarboukh/FTTell-System.git

Project code: 24-1-D-3

Supervisor:

Dr. Natali Levi-Soskin

Authors:

Sabeel Hamood - 211581343

Sabeel.Hamood@e.braude.ac.il

Ehsan Sarboukh - 209002534

Ehsan.Sarboukh@e.braude.ac.il

Abstract

Failure to Thrive (FTT) is a prevalent issue among infants that is often overlooked or misdiagnosed. Traditional diagnostic systems for FTT are outdated, typically relying on factors such as weight or height below the 3rd percentile for age or a slowing growth rate. Our system, FTTell, aims to modernize and enhance the diagnosis and management of FTT through a two-part approach.

The first part of FTTell is accessible to all users and offers a demo test that uses scientifically validated formulas to assess a child's growth status. This test has demonstrated an accuracy of 88% and provides users with a preliminary indication of potential growth concerns.

The second part of the system is designed specifically for pediatricians, who must register and log into the platform. Beyond basic growth assessments, pediatricians have access to advanced tools with sensitivity analysis and a specialized emotion recognition test for children aged 6-25 months. This computer vision-based test evaluates facial expressions to further refine the diagnosis. Additionally, pediatricians are provided with a personalized nutritional plan generated by an AI system, it is the responsibility of the pediatrician to decide according to his judgment and the experience he has acquired over the years whether to refer to the nutritional plan and implement it in treatment.

This dual functionality not only empowers parents with early insights but also equips healthcare professionals with comprehensive tools to improve the accuracy and effectiveness of FTT diagnosis and management.

Table of Contents

1.Introduction

Failure to Thrive (FTT) is a significant health issue affecting infants and young children, often going unnoticed or being misdiagnosed due to reliance on outdated diagnostic methods. Traditionally, FTT diagnosis has focused primarily on growth metrics such as weight or height percentile, failing to capture the complexity of underlying issues. This lack of precision can delay intervention, leading to long-term developmental problems and affecting the child's overall well-being. Addressing this challenge is critical, as early detection and accurate diagnosis are crucial for providing timely and effective treatments.

The most used methods for diagnosing Failure to Thrive (FTT) focus on various aspects of a child's development. Growth monitoring is the primary tool, where a child's weight and height are regularly compared against standard growth charts. Medical history and physical examinations provide additional context by assessing any underlying health conditions that may contribute to poor growth. Nutritional assessments help identify dietary deficiencies, while psychosocial evaluations look into family dynamics and other environmental factors that could influence a child's growth. One widely recognized tool is the Rourke Baby Record (RBR), developed in Canada, which serves as a comprehensive tool for growth monitoring. However, the RBR is primarily a preventative health guide and is not specifically designed for diagnosing FTT, which limits its effectiveness in identifying the condition early. These methods, while helpful, often lack the depth and precision required for a thorough diagnosis of FTT, emphasizing the need for more advanced diagnostic systems like FTTell.

The system operates in two parts. For general users, it offers a demo test that applies scientifically validated formulas to provide an initial assessment of a child's growth status, giving parents early insights into potential concerns. For pediatricians, advanced features are available, including sensitivity analysis and a specialized computer vision-based emotion recognition test. This test evaluates facial expressions of children aged 6-25 months to enhance diagnostic accuracy. Additionally, an AI-driven module generates personalized nutritional plans to support FTT management. Together, these tools offer a more holistic approach, addressing the limitations of traditional diagnostic methods.

FTTell directly benefits both parents and healthcare professionals. Parents can gain early awareness of potential growth issues in their children through an accessible, user-friendly demo test. Pediatricians, on the other hand, are equipped with a comprehensive suite of tools, including emotion recognition and AI-powered nutrition recommendations, to make more informed diagnostic decisions. This system ultimately enhances the accuracy and timeliness of FTT diagnosis, benefiting children by ensuring

they receive early interventions that can significantly improve their long-term development and health outcomes.

The structure of this document is designed to guide readers through the development and innovation behind FTTell, starting with a **Review of the Literature**, which explores the foundational research on diagnosing Failure to Thrive (FTT) and current solutions. It then transitions into the **Expected Achievements**, detailing the system's unique capabilities, such as machine learning integration, comprehensive databases, and a user-friendly interface. The **Process** section covers the in-depth research into FTT, artificial intelligence, and computer vision, as well as the challenges faced in implementing these technologies. The **Product** section outlines the technical architecture, frontend and backend technologies, and design elements of the user interface, while the **Verification and Evaluation** section concludes with a thorough assessment of the system's functionality and impact.

2. Background

Pediatric Failure to Thrive (FTT), also known as weight faltering poses a substantial challenge for pediatricians, often leading to delayed diagnosis [4].

FTT is defined by weight-for-age below the 3rd percentile or weight deceleration across two major growth chart percentiles, with single indicators having limited predictive values.

Pediatricians need to evaluate the child from several angles, including psychosocial development, prenatal, birth weight, and postnatal development, to identify FTT [7][9].

The differential diagnosis for failure to thrive is extensive, historically categorized as organic and nonorganic. Organic failure to thrive involves major disease processes or organ dysfunction, while nonorganic failure to thrive suggests insufficient emotional or physical nurturing without clear pathophysiologic abnormalities, often associated with reactive attachment disorder.

Researchers team found that prevalence varies based on the studied population and recognition criteria [4].

In the U.S., around 10% of children in primary care and 5% of hospitalized children may experience FTT. An analysis showed varying rates of children crossing major percentiles in height-for-age and weight-for-age across different age groups. Weight-for-height had the highest crossing rates during preschool years, while weight-for-age had the lowest [2].

However, a downward shift doesn't necessarily indicate failure to thrive; a research showed that nearly 30% of normal babies experience a downward shift between 3 and 18 months, with the average of 13 months [9].

Low birth weight is a major predictor of the need for future referral for failure to thrive refer to height-for-age, weight-for-age, weight-for-height, and BMI-for-age.Risk factors may be classified as psychosocial or medical [4].

psychosocial: economic factors, psychosocial status considerations include marital stress, potential homelessness, domestic violence, parental employment, and substance abuse. Children born to mothers under 18 show poorer growth, especially in the first year, and are twice as likely to experience neglect compared to those born to older mothers [9]. medical: inadequate caloric intake, Gastroesophageal reflux, Inadequate breast milk supply, or ineffective latching, incorrect formula preparation, Mechanical feeding difficulties (e.g., cleft lip or palate) etc. [4].

FTT should be considered a clinical observation rather than a diagnosis. Nowadays recognition relies on consistent and accurate measurements of weight and height over time, documented on appropriate growth charts that use the "WHO charts (http://www.who.int/childgrowth, http://www.cdc.gov/growthcharts)". [4], when growth problems are suspected in infants, a team of doctors may recommend various laboratory tests to investigate further. These tests typically include: Blood count, Urinalysis, Electrolyte measurement, Thyroid function tests, Celiac disease testing, Testing for food allergies[4].

When medical evaluations fail to identify organic causes for the growth failure, a diagnosis of nonorganic FTT is made [1].

Facial recognition and emotional expressivity play vital roles in infants' social and cognitive development. Nonorganic FTT infants' facial expressions offer insights into their emotional display [1].

Currently, the "Rourke Baby Record" (RBR) from Canada is a comprehensive tool used for monitoring growth, although it is not employed for diagnosing FTT [8]. In modern diagnostic approaches, computer vision is being incorporated into pediatric healthcare through facial recognition, aiding in the identification and tracking of children. This integration enhances tailored care and boosts administrative efficiency within the medical sector. Furthermore, while AI tools can offer insights based on symptoms to assist in diagnosing Failure to Thrive (FTT) in children, they are not typically used by physicians for making medical diagnoses.

FTTell, a model-based tool for diagnosing postnatal FTT in children aged 0–5, has received ethical approval "permission number 0665-19-RMB". With a user-friendly interface and continuous updates, it achieved 87% accuracy in diagnosing FTT in 100 infants compared to expert diagnoses. Medical professionals can contribute to its refinement, aiding in accurate FTT diagnoses by pediatricians. This model is built on OPM models, prioritizing efficiency, and user-centric design. This approach ensures iterative improvement and optimal system performance [7].

To investigate the relationship between the facial expressions of children with non-organic FTT, the study included 12 nonorganic FTT infants (ages 6 to 25 months) and 12 matched control infants, they were videotaped in social and cognitive contexts to capture their facial expressions and emotional responses [1].

1. The first segment, lasting 4 minutes, was taken from an interview with the mother. During this time, the infant sat on the mother's lap with a few toys on a table in front, providing a relaxed environment for potential emotion expression.

2. The second segment comprised the first 3 minutes of administering the Bayley Scales of Infant Development, to reflect the infant's functional level.
    2.1. In the third segment of the testing procedure, the infant first engaged in a 1.5-minute face-to-face interaction with the experimenter.
    2.2. Following this, there was a 1.5-minute presentation of three toys - a mask, a jack-in-the-box, and balloon noise - selected to provoke emotional responses in the infant.
3. This was followed by a 2-minute face-to-face interaction between the infant and the mother, In the last dyadic interaction segment, mothers were given instructions to engage with their infants in a manner similar to how they would typically play with them at home.

To maintain consistency in infants' experiences, they first interacted with the experimenter, followed by their mothers, with an emotion-eliciting toy segment in between. Various age-appropriate toys were used to evoke emotions. Facial expressions were coded using the MAX system by Izard (1979), which categorizes emotion-relevant facial movements into upper, middle, and lower face actions [1].

FTT infants showed fewer expressions of positive emotions like joy, interest, and surprise compared to typically developing infants. In contrast, FTT infants displayed a higher frequency of negative emotions, including fear, sadness, anger, disgust, and displeasure. Furthermore, these negative emotional expressions lasted longer in FTT infants than in controls [1].

In addition to facial movements, this study also coded three other behaviors related to the regulation of emotional states: eye gaze, and self-comforting gestures. Four types of gazes were coded: averted attention, gazes toward the task, gazes toward the mother, and gazes toward the experimenter. Self-comforting behaviors encompassed actions such as playing with the hands, bringing the hands to the mouth or head, and placing objects in the mouth [1].

Results indicated that FTT infants exhibited a significantly higher frequency of expressions involving the upper face compared to controls. Conversely, the use of the lower face did not significantly differ between the two groups, FTT infants averted their gaze (from the toy or from the experimenter) more often than controls [1].

Treatment is given according to the source of the problem (medical or psychosocial)

Pharmacotherapy like cyproheptadine or megestrol (Megace) may be used for severe cases with underlying diseases or cancer treatment but isn't generally recommended for FTT. Catch-up growth involves children gaining at two to three times the average rate

for their age, with adjustments to breastfeeding or formula advised for infants with inadequate caloric intake [4].

Facial recognition in nonorganic FTT infants can provide clues about the environment of the infant and which can indicate the development of FTT following an environmental rather than genetic factor [1].

Psychosocial interventions include counseling parents on sick care management and promoting a balanced diet. Home nursing visits can supplement clinic evaluations, especially for families with transportation or work-related issues. Families experiencing food insecurity should receive social work support or access community resources like the Women, Infants, and Children program [4].

3.Architecture overview

- The architecture depicted in the diagram represents a 3-tier web development application hosted on a Digital Ocean Droplet. In this architecture, the client side (Client 1, Client 2, and Client 3) initiates HTTPS requests that are routed through a Domain Name System (DNS). The DNS resolves the domain name (FTTell.org) to the IP address of the server, ensuring that client requests reach the correct destination.
- Once the requests are resolved by DNS, they pass through NGINX, which acts as a reverse proxy server. NGINX efficiently manages incoming traffic by forwarding these requests to the appropriate service. In this case, the Node.js application. NGINX also handles load balancing, caching, and security features, improving the overall performance and security of the application.
- The requests then reach the presentation layer (Frontend), which handles the user interface and interacts directly with the users. These requests are passed to the business layer (Backend), which contains all the core functionality and logic of the application. The business layer processes the requests and forwards them to the data layer.
- The data layer interacts with the database (MongoDB) to retrieve or store data as needed and also communicates with an AI tool (third-party services) for advanced functionalities. The responses from the data layer are then sent back through the business layer to the presentation layer, which delivers the final output to the clients. This 3-tier architecture, supported by DNS and NGINX, ensures a clear separation of concerns, making the application scalable, secure, and maintainable.

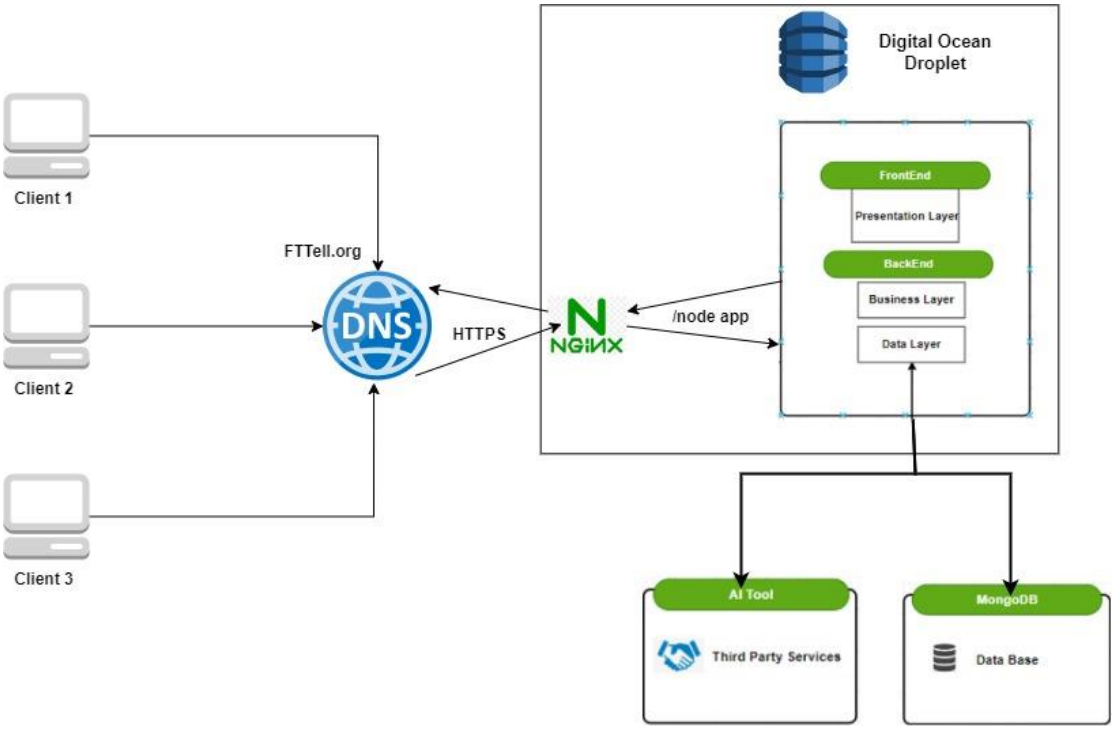

Figure 1: 3-Tier Architecture

4.Web Application

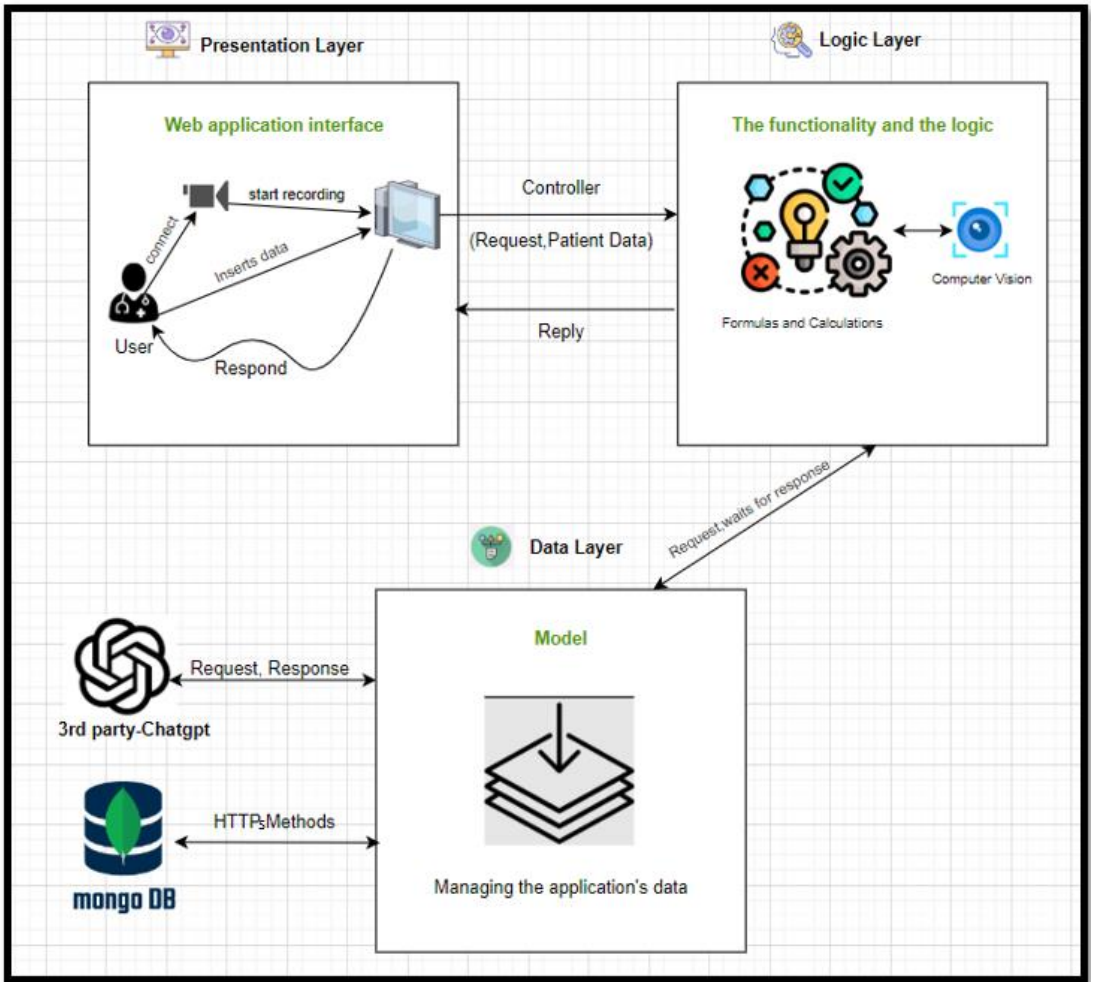

Figure 2: MVC architecture

We opted for the MVC architectural design pattern, given that our system is structured into three layers. MVC is recognized for its ability to manage these three layers effectively, which are the presentation layer, data layer, and business layer.

Presentation layer (view):

The presentation layer (also known as the view) is responsible for displaying user interfaces, forms, and integrating camera functionality. When a pediatrician accesses the system, they input patient data, and after entering all required information, they submit it. The system then creates a patient record. The presentation layer ensures a smooth interaction between users and the application with a secure way.

Logic layer:

The controller retrieves data from the form and queries the model.

The model houses the entire logic and functionality of the web application.

The received data is processed using the implemented functionality.

Additionally, the business layer communicates with the data layer to obtain responses from the AI tool or database.

The data layer: receives requests from the business layer and stores the data in the MongoDB database.

It also communicates with third-party services, including the AI tool.

The AI tool's response is relayed back to the data layer, which then forwards it to the business layer.

In the business layer, a comparison is made between results from formulas and the AI tool, and the controller responds accordingly.

Finally, the controller presents the model's reply that the view layer requested from the controller, The view layer presents on the screen for the pediatrician.

5.UML Diagrams

5.1 Activity Diagram:

- **Login Process:**

This diagram illustrates the user authentication process, where user credentials are validated, decrypted, and paired with stored passwords to grant access to the system or deny it if the credentials are incorrect.

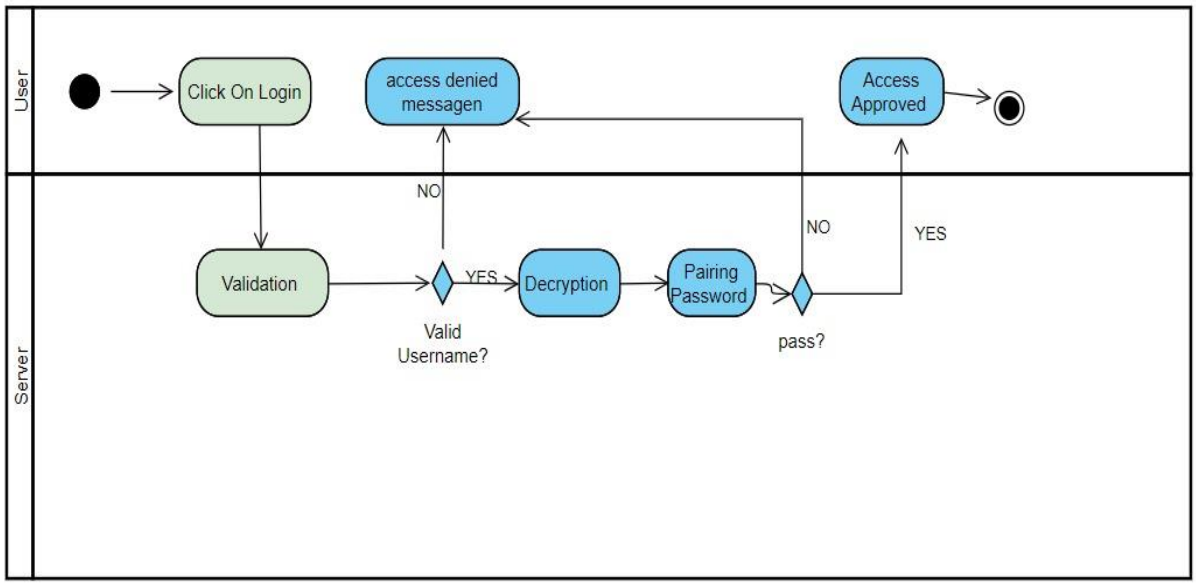

Figure 3: Login process -Activity diagram

- **Sign Up Process:**

SignUp Activity Diagram: This diagram represents the registration process, where a user submits an access code, which is validated and decrypted. Upon successful validation, the user can complete the registration process, provided the chosen username is unique.

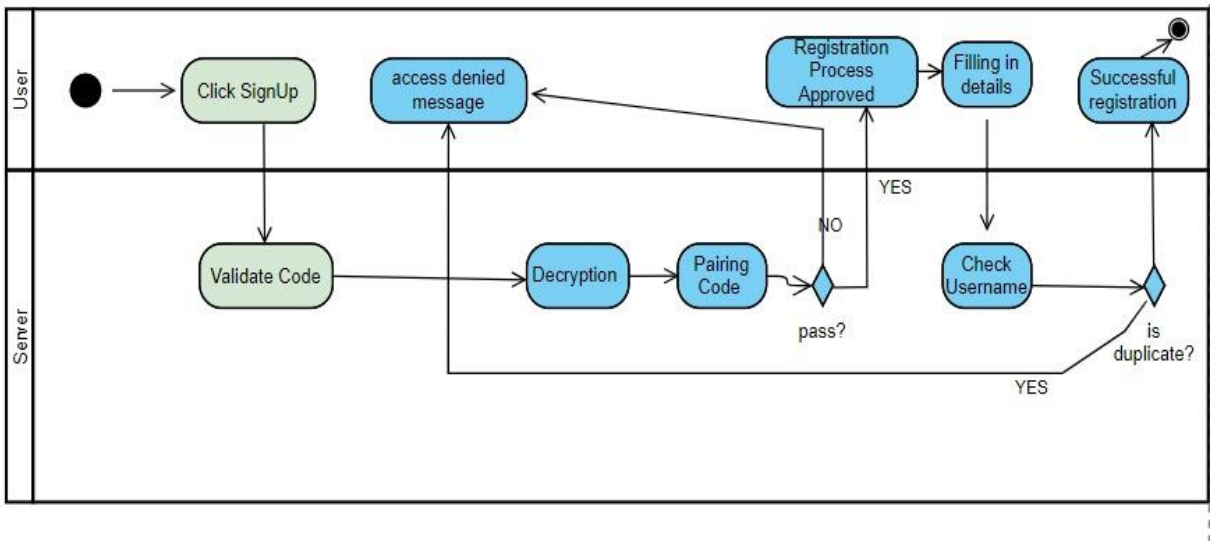

Figure 4: Sign up process -Activity diagram

- **Demo Diagnose:**

  This activity diagram represents the sequence of actions in "Demo Diagnose" process that will be accessible to all visitors to the site who are not a pediatrician and wish to diagnose their children.

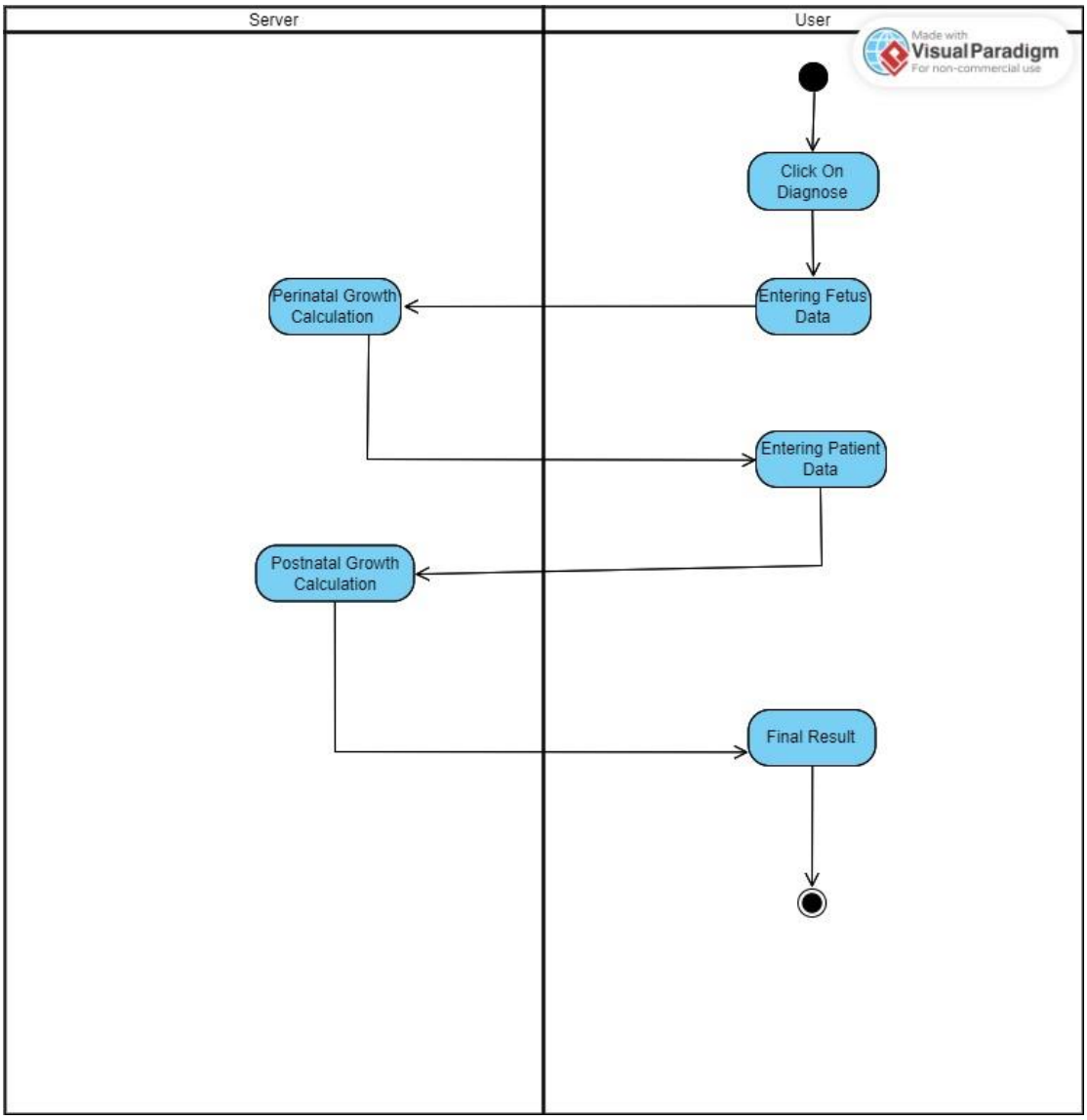

Figure 5: Demo diagnose -Activity diagram

- **Diagnose Patient For Pediatrician:**

  This activity diagram represents the sequence of actions in the "Patient Diagnosis" process, which is accessible only to pediatricians. The diagram assumes that the doctor is already logged into the system, following the registration and login process activity diagram.

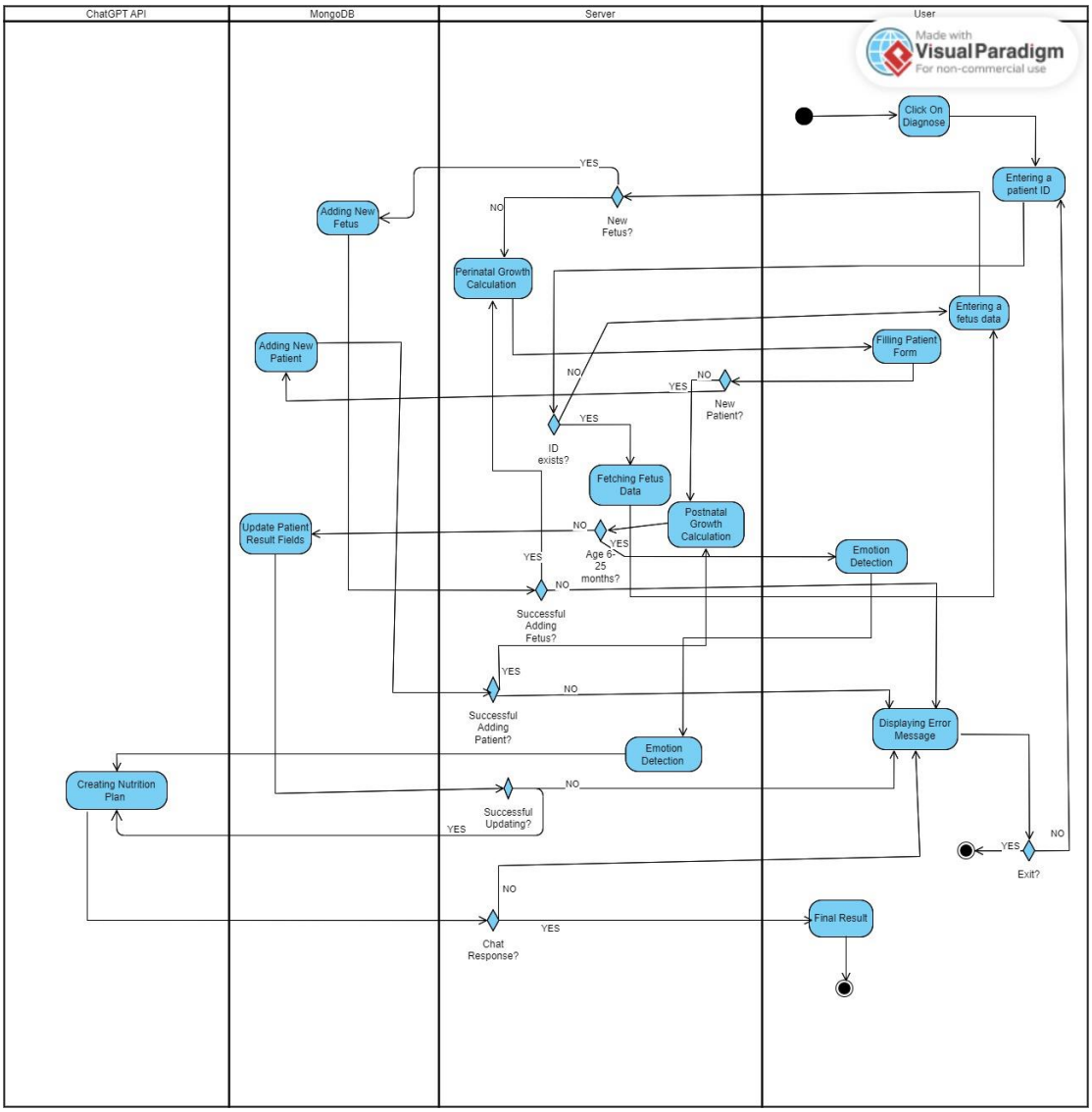

Figure 6: Diagnose Patient For Pediatrician -Activity diagram

5.2 Package diagram:

This package diagram represents the overall architecture of your application, connecting the frontend built with React.js, the backend developed in Node.js with emotion detection and routing, and the database hosted on MongoDB Atlas where different collections such as patients and pediatricians are managed.

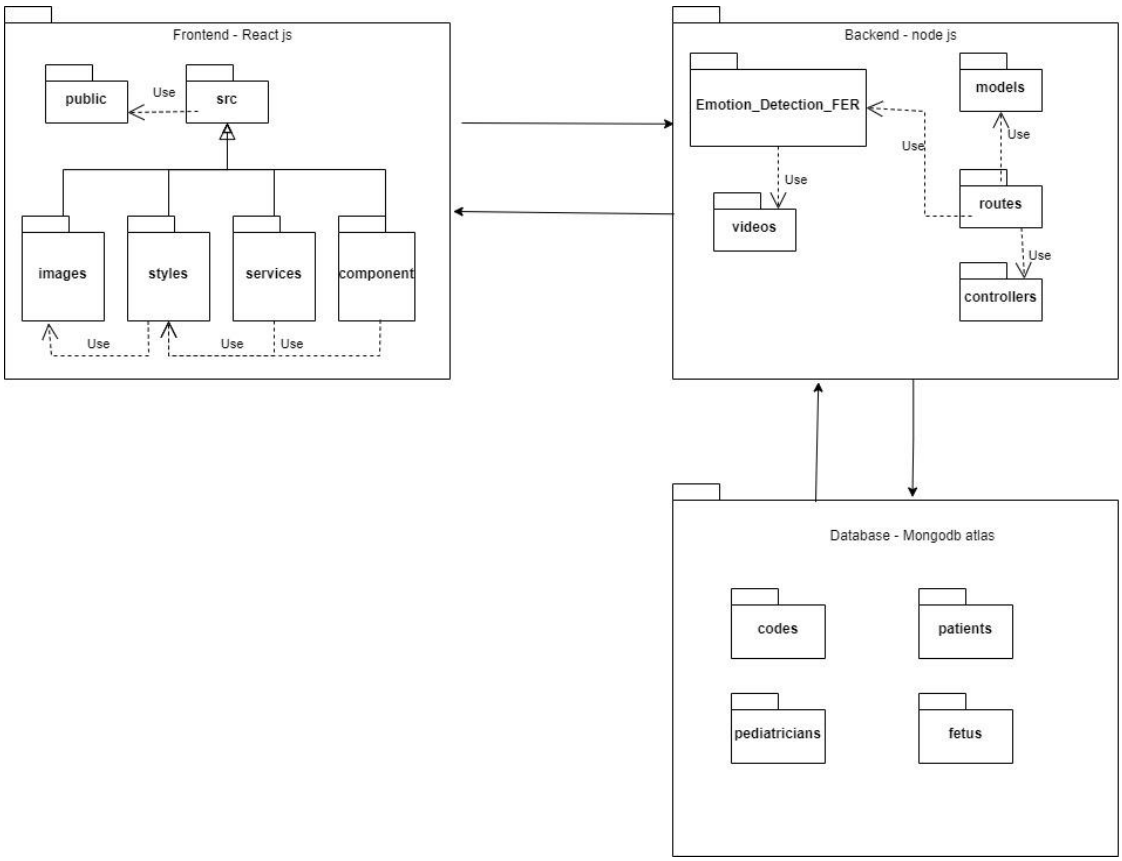

Figure 7: Package diagram

6. Project Work Division

Throughout the project, we adhered to agile principles to manage our work effectively. This approach involved breaking down the project into smaller, manageable tasks, each associated with a specific sprint and deadline. Regular check-ins, reviews, and adjustments were made to ensure the project remained on track and aligned with our goals. The use of sprints allowed us to focus on delivering incremental improvements, ensuring that each component was fully functional and tested before moving on to the next.

- Phase 1: Computer Vision Integration
    - Task 1: Implement Face Detection using FER Library
        - Sprint: Develop and test the computer vision algorithm using the FER (Facial Emotion Recognition) library in Python.
        - Details: The primary focus was to accurately detect and analyze facial emotions from images to aid in diagnosing FTTEL among children.
    - Task 2: Integration of Python Script with Node.js
        - Sprint: Research and implement a method to run the Python script within a Node.js environment.
        - Details: This task involved bridging the Python-based FER detection system with the Node.js backend, ensuring smooth communication between the two.
- Phase 2: Backend Development
    - Task 3: Develop Backend with Node.js and Express
        - Sprint: Set up and program the backend server using Node.js and Express to handle requests, manage data, and process results from the FER library, create a database connection via mongoose using atlas.
        - Details: The backend was designed to be robust and scalable, with endpoints for receiving and processing data from the frontend.
    - Task 4: Data Encryption for Database Security
        - Sprint: Implement encryption techniques to secure sensitive data stored in the database.
        - Details: This task focused on ensuring that all sensitive information, particularly patient data, was encrypted before being stored in the database to maintain data privacy and security.
- Phase 3: Frontend Development
    - Task 5: Frontend Integration using React.js

        - Sprint: Develop the frontend interface using React.js and implement communication with the backend using Axios for API requests.
        - Details: The frontend was designed to be intuitive and user-friendly, allowing easy navigation through the diagnosis process.
    - Task 6: Modernize UI Design
        - Sprint: Focus on enhancing the visual appeal of the frontend by modernizing the design, making it more accessible and appealing.
        - Details: Implemented modern UI/UX principles to ensure a clean and professional appearance of the application.

- Phase 4: Deployment
    - Task 7: Cloud Deployment and DNS Setup
        - Sprint: Deploy the application on a cloud platform (e.g., Digital Ocean) and set up a DNS name for easy access.
        - Details: The deployment process included configuring the server, uploading the application, and ensuring it was accessible online.
    - Task 8: Securing Communication with HTTPS
        - Sprint: Implement HTTPS for secure client-server communication using Certbot or similar tools.
        - Details: This task focused on securing the application by setting up SSL certificates, ensuring all communication was encrypted.

7. Challenges

During the project, we encountered several challenges that required innovative thinking to address effectively.

- **securing sensitive data:** During the development of our system, one of the key challenges we faced was securing sensitive data within our MongoDB database, particularly the identification of pediatricians, patients, and pediatricians' passwords. To address this, we employed two robust methods. First, for password security, we utilized the Node.js crypto library to hash the pediatricians' passwords. Second, for securing the identification information of both pediatricians and patients, we implemented the AES-256 symmetric encryption algorithm. AES-256, known for its strong encryption standard, uses the same key for both encryption and decryption, making it resistant to brute-force attacks. The security of this method relies on proper key management, including secure storage and periodic rotation of the encryption key. By encrypting this sensitive information, we ensure that even if the database is compromised, the data remains protected and unreadable without the correct decryption key, thereby safeguarding the privacy of both pediatricians and patients.
- **Restrict access:** Since our system includes both a demo version and a version exclusive to pediatricians, we need to ensure that regular users who are not certified pediatricians can only access the demo system. The entry point to the pediatrician-only system is a registration screen designed solely for certified pediatricians. This led us to consider how to prevent non-pediatricians from registering. One idea was to require users to upload a certified pediatrician license document, which we would then verify against a global database of pediatricians. If the license was validated, the user could proceed with registration; otherwise, they would be blocked. Although this seemed like an excellent solution, it proved impractical due to the extensive permissions required to access a global pediatrician database and the time constraints for implementation.

  Instead, we opted for a more feasible approach. We added a screen displaying an email contact for our team. Prospective users interested in registering for the pediatrician-only system must contact us via email and provide documents verifying their identity as a qualified pediatrician. After our team confirms their credentials and ensures they are not imposters, we will send them a secret code. This code will unlock the registration screen, allowing them to register for the system.

- **Using computer vision :** Given that this was our first project involving computer vision, our initial step was to thoroughly understand the technology and determine which libraries to use. We initially faced challenges in implementing the process of identifying and decoding a child's emotions using computer vision. To address this, we began searching for a suitable dataset online to train our model. However, after extensive research, we discovered pre-trained Python libraries that did not require additional training.

  We utilized the cv2 library, which is an interface to OpenCV, an open-source computer vision and machine learning software library. Additionally, we used the FER (Facial Emotion Recognition) library, which is specifically designed for emotion detection in images and videos. FER simplifies the recognition of emotions by providing pre-trained models that classify facial expressions into basic emotions such as happiness, sadness, anger, and surprise. By leveraging these advanced, existing technologies, we significantly reduced our implementation time while achieving a high level of accuracy.
- **Responsive website:** Adapting the design for all users across different devices is a crucial aspect of the project. The site needs to be responsive and compatible with various devices, including computers and smartphones. However, the system designed for pediatricians is intended exclusively for computer use, as we want to ensure that doctors capture the child's facial image only with a computer camera or an external camera connected to the computer, not with a smartphone.

  The challenge was to ensure that our CSS files would effectively accommodate all screen sizes, which required significant time and extensive testing. To address this, we designed the layout using percentage-based dimensions within the CSS files to maintain flexibility across different devices. Additionally, we incorporated "@media" queries to tailor the design to specific, universally recognized screen sizes, ensuring a responsive and user-friendly interface across all platforms.
- **Deployment**: another challenge we encountered a significant constraint due to the multiple technologies involved—Node.js, React.js, and a Python script with computer vision. Initially, we attempted to auto-deploy using Render, but this approach proved unsuccessful for our complex setup. Consequently, we began exploring other solutions and ultimately decided to deploy the application on DigitalOcean. DigitalOcean offered us greater flexibility and control, allowing us to create and configure Droplets (virtual private servers) tailored precisely to our application's needs.
- **Secure Client-Server Communication**: securing the data transferred between the client and server, a critical aspect of our application's security. To address

this, we implemented HTTPS using Certbot to secure client-server communication. HTTPS encrypts all data transmitted over the network, protecting sensitive information such as user credentials and API requests from potential eavesdropping or interception by unauthorized parties.

- **accuracy in AI result:** Creating suitable prompts for ChatGPT-3.5 Turbo to generate accurate and personalized nutrition plans for our patients. Initially, the responses lacked the specificity needed, so we iteratively refined our approach by testing various prompt structures. Through this process, we discovered that providing detailed patient information was crucial. This included specifics such as the patient's age in months, weight, height, gender, and any relevant medical conditions. We also found it necessary to mention any dietary restrictions or preferences, such as allergies.

8. Technologies of the development

8.1 Programming Languages:

- **Python:** Python is a versatile and widely-used programming language known for its simplicity and readability. Its clear syntax makes it a popular choice for both beginners and experienced developers. Python supports a broad range of applications, from web development to data analysis, artificial intelligence, and automation. With a rich ecosystem of libraries and frameworks, Python enables efficient development in various domains, including scientific computing, machine learning, and more. Its community-driven nature and extensive support make it an essential tool in the modern programming landscape.

  In the context of our work, we used Python in the backend to develop several key components of our project. Python was essential for implementing the emotion detection functionality using computer vision techniques. We also leveraged Python for analyzing the frequency of specific occurrences in our data.

- **JavaScript:** JavaScript is a dynamic and widely-used programming language primarily known for its role in web development. It enables developers to create interactive and responsive user interfaces by manipulating HTML and CSS in real time. Beyond the browser, JavaScript is also popular for backend development through environments like Node.js, allowing for full-stack development with a single language. With a vast ecosystem of libraries and frameworks such as React.

  JavaScript played a crucial role in our project, both on the frontend and backend. On the frontend, we used JavaScript with React to create a dynamic and responsive user interface, ensuring a smooth and intuitive experience for users. This included managing navigation, handling user input, and ensuring that data was accurately collected and processed. The frontend was designed to be user-friendly and accessible, allowing for seamless interaction with the system.

  On the backend, JavaScript, combined with Node.js and Express, was used to build the server-side functionality. This included setting up routes, managing API requests, and integrating with external services like the OpenAI API for generating nutrition plans. Additionally, JavaScript was employed to connect the backend with Python scripts,

enabling the processing and analysis of patient data. The backend also handled the secure storage of data, such as pediatrician information, ensuring the system's reliability and security. Overall, JavaScript's versatility and robust ecosystem were key to successfully implementing both the client-side and server-side aspects of the project.

8.2 Node.js Runtime Environment:

Node.js is an open-source, cross-platform runtime environment that allows JavaScript to be executed on the server-side, enabling full-stack JavaScript development. In our project, Node.js was crucial for building the backend, allowing us to run JavaScript outside of the browser and create a robust, scalable server. Its non-blocking, event-driven architecture was particularly beneficial for handling multiple requests efficiently, making it an ideal choice for our application's backend services. Node.js served as the foundation upon which we built our API, managed routing, and integrated various services like MongoDB and external APIs.

8.3 Computer Vision:

Computer vision is a field of artificial intelligence that enables machines to interpret and understand visual information from the world, much like humans do. By processing images and videos, computer vision systems can recognize objects, track movements, detect faces, and analyze visual data in various ways. This technology is widely used in applications such as facial recognition, autonomous vehicles, medical imaging, and more. It leverages techniques from machine learning and deep learning to enhance the accuracy and efficiency of visual data analysis, making it a powerful tool for automation and intelligent decision-making in a wide range of industries.

8.4 JavaScript Open-Source Libraries:

- **Express:** Express is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. It simplifies the process of handling HTTP/HTTPS requests, routing, middleware management, and server-side logic, making it a popular choice for developing web applications and APIs. Express's lightweight nature and extensive middleware ecosystem allow developers to create scalable and maintainable applications quickly.

  In our project, we used Express to build the backend API that handles various server-side operations. Express allowed us to manage routes efficiently,

enabling communication between the frontend and backend. We utilized it to handle HTTPS requests, Additionally, Express was crucial for integrating with other services, such as interacting with the OpenAI API and connecting to our MongoDB database.

- **Mongoose:** Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward, schema-based solution to model application data. Mongoose allows developers to define the structure of their data and interact with MongoDB using a simple and elegant syntax. It includes built-in data validation, type casting, query building, and business logic hooks, making it a powerful tool for managing MongoDB collections within a Node.js environment. we used Mongoose to manage and interact with the MongoDB database. Mongoose allowed us to define clear schemas for our data models, such as pediatricians and patients, ensuring consistency and structure in our database. Its built-in validation and middleware capabilities helped us enforce data integrity and execute custom logic before and after database operations, making it an essential part of our backend development.

- **Axios:** Axios is a promise-based HTTP/HTTPS client for JavaScript, widely used for making asynchronous requests to external APIs or servers. It functions in both browser and Node.js environments, providing a simple and consistent interface for sending GET, POST, PUT, DELETE, and other HTTP/HTTPS requests. Axios handles automatic JSON data transformation, request and response interception, and supports features like timeout control, request cancellation, and automatic retries. Its ability to seamlessly manage both HTTP and HTTPS requests makes it a reliable choice for secure and efficient communication in web applications. we used Axios to handle secure communication between the frontend and backend, as well as to interact with external APIs. After deploying the project to the cloud, all requests were made over HTTPS to ensure data security. Axios was particularly useful for sending HTTPS requests from the frontend to the backend server to submit form data, retrieve patient information, and handle user authentication securely. Additionally, we used Axios in the backend to make secure API requests to external services, such as interacting with the OpenAI API for generating nutrition plans

- **Multer:** Multer is a Node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is built on top of the busboy library and makes it easy to manage file uploads in Express and Node.js

applications. Multer allows developers to define storage options, file size limits, and file filters, providing control over where and how files are stored on the server. It is a popular choice for handling file uploads in web applications due to its simplicity and flexibility. we used Multer to manage the upload of video files required for the emotion detection functionality. Specifically, we configured Multer to store video files in a designated directory on the server, ensuring that files were properly organized and accessible for processing. Multer allowed us to easily integrate file uploads into our Express application, handling the multipart/form-data seamlessly. By using Multer, we were able to efficiently manage the storage and retrieval of video files.

- **dotenv:** Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env in Node.js applications. It is commonly used to manage configuration variables, such as API keys, database credentials, and other sensitive information, without hardcoding them directly into the application code. By using .dotenv, developers can keep sensitive data secure and separate from the codebase, making it easier to manage different configurations for development, testing, and production environments. In our project, we used .dotenv to securely manage sensitive configuration details, such as API keys for external services and MongoDB connection strings. By storing these values in a .env file, we ensured that our sensitive data remained secure and was not exposed in the source code. This approach also made it easier to switch between different environments, such as development and production, by simply updating the .env file. Using .dotenv allowed us to maintain a clean and secure codebase while providing flexibility in managing the project's configuration.

- **Cors:** CORS (Cross-Origin Resource Sharing) is an open-source Node.js library that provides a way to manage cross-origin requests. In web applications, the Same-Origin Policy restricts how resources on one domain can be requested from another domain, which can prevent a web application from interacting with APIs hosted on different domains. The CORS library helps by enabling developers to configure these cross-origin requests, specifying which domains are allowed to access server resources, what HTTP methods are permitted, and what headers can be exposed. we used the CORS library to allow secure communication between our frontend, which was hosted on one domain, and our backend API, which was hosted on

another domain. This was particularly important after deploying the project to the cloud, where different components might reside on separate domains. By configuring CORS, we were able to specify which domains could make requests to our API, ensuring that our application could function correctly while maintaining security.

- **React:** React is a popular open-source JavaScript library for building user interfaces, particularly for single-page applications (SPAs). It allows developers to create reusable UI components that manage their own state, leading to more organized and maintainable code. React's virtual DOM enables efficient updates and rendering of components, making web applications fast and responsive. It is widely used for creating dynamic and interactive user interfaces, and its component-based architecture encourages modular design. we used React to build the frontend user interface, focusing on creating a dynamic and responsive experience for users. React's component-based architecture allowed us to break down the UI into reusable components, such as forms, navigation elements, and data display sections, which made the development process more efficient and the code easier to maintain. The ability to manage state within components helped us handle user interactions and data updates smoothly. React also enabled us to implement client-side routing, allowing for seamless navigation between different parts of the application without requiring full page reloads. This made React an ideal choice for delivering a modern, interactive user experience in our project.

- **React Router:** React Router is a separate library that works with React to handle routing. Routing is the process of determining what content to display based on the URL. React Router allows developers to create and manage navigation between different views or pages in a React application, enabling single-page applications (SPAs) to have multiple routes (URLs) without requiring full page reloads. This was particularly useful in maintaining the state of the application and providing a fluid interaction flow for users. The ability to define routes dynamically based on user roles or actions made React Router an essential tool in organizing and controlling the application's structure.

8.5 Python Open-Source Libraries:

- **OpenCV (cv2):** OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a vast range of tools for image processing, video capture, and analysis, enabling developers to perform tasks such as object detection, facial recognition, image transformations, and much more. The cv2 module is the Python interface for OpenCV, making it accessible for Python developers to implement complex computer vision algorithms easily. OpenCV is widely used in industries ranging from robotics to healthcare due to its comprehensive capabilities and high performance. we used the cv2 module of OpenCV to implement the emotion detection functionality. Specifically, OpenCV was utilized for processing video frames captured from the camera and analyzing them to detect facial features. The powerful image processing tools provided by cv2 allowed us to accurately identify and track facial expressions, which were then analyzed to determine the emotional state of the child.

- **FER (**Facial Emotion Recognition): is an open-source Python library designed for detecting emotions from facial expressions in images and videos. It simplifies the process of recognizing and classifying basic emotions, such as happiness, sadness, anger, surprise, and more, by leveraging pre-trained deep learning models. FER is easy to integrate into computer vision projects, providing developers with a high-level interface to perform emotion recognition tasks without the need for extensive training data or complex model development. In our project, we used the FER library to automatically detect and analyze the emotions of children based on their facial expressions captured by the camera. By integrating FER with OpenCV, we were able to process video frames in real-time and apply FER's pre-trained models to classify the detected emotions accurately. This was a critical part of our project's functionality, as understanding the emotional state of the child was essential for our diagnostic process. The simplicity and effectiveness of FER allowed us to implement this feature quickly and with a high degree of accuracy, ensuring that our system could provide reliable emotional analysis as part of the overall diagnostic toolkit.
- **Numpy:** NumPy is a fundamental library in Python for numerical and scientific computing. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is the backbone of many other scientific libraries in Python, enabling operations such as linear algebra, Fourier transforms, and random number generation. One of the key features of NumPy is its ability to perform

vectorized operations, which means that operations on arrays are carried out element-wise, leading to significant performance improvements over standard Python loops. This makes NumPy essential for tasks involving large datasets or computational-heavy operations, such as data analysis, machine learning, and image processing.

8.6 MediaPipe in Computer Vision: MediaPipe is an open-source framework developed by Google that provides cross-platform, customizable, and efficient pipelines for building multimodal machine learning applications, particularly in the context of computer vision. It offers pre-built solutions for tasks such as hand tracking, face detection, and pose estimation, and is designed to work seamlessly across various platforms, including mobile devices and desktops. We integrated MediaPipe with OpenCV to enhance the accuracy and efficiency of emotion detection and facial recognition processes. By leveraging MediaPipe's high-performance pipelines, we were able to detect and analyze facial landmarks in real-time, allowing for more accurate tracking and assessment of children's facial expressions in our diagnostic system. MediaPipe's robust processing capabilities ensured smooth operation, even in real-time scenarios, making it a valuable asset to our project.

8.7 JSON: JSON (JavaScript Object Notation) is a lightweight data format used for data interchange between systems. In Python, the json module allows for easy serialization and deserialization of Python objects into JSON strings and vice versa. This is particularly useful when storing data in files or sending it over networks, as JSON is both human-readable and easy for machines to parse. In the provided code, JSON is used to store the results of emotion detection in a structured format, allowing the data to be saved to a file and later retrieved for analysis or other purposes. The json module simplifies this process, enabling seamless conversion between Python dictionaries or lists and JSON-formatted data.

8.8 Integration of python with Node.js using child_process: The child_process module in Node.js allows the spawning of new processes and the execution of system commands from within a Node.js application, making it ideal for running tasks that require separate execution environments. In our project, we used child_process to execute Python scripts directly from our Node.js server, which was crucial for integrating Python-based functionalities like emotion detection into the application. By spawning child processes, we could run these scripts independently,

capturing their output and errors, and seamlessly incorporating Python's capabilities into our Node.js environment. This approach enabled us to leverage complex algorithms without disrupting the main server operations.

8.9 Handling File Operation with Node.js Using File System: The fs (File System) module in Node.js provides an API for interacting with the file system, allowing for operations such as reading, writing, creating, and deleting files and directories. we used the fs module to manage video files related to the emotion detection process. Specifically, it allowed us to create directories, check for their existence, and store video files securely. By using fs, we ensured that our application could handle file storage efficiently, organizing and managing the necessary resources for processing video data without manual intervention.

8.10 Data encryption in the backend: We ensured the security of sensitive data for both patients and pediatricians by encrypting their IDs. To encrypt the pediatrician's ID, we use the bcrypt library to securely encrypt the IDs of pediatricians during the registration process. Bcrypt is a robust hashing algorithm specifically designed to protect sensitive information, such as passwords and IDs, by generating a unique, one-way hash that is extremely difficult to reverse. When a new pediatrician registers, their ID is encrypted using bcrypt, which applies a hashing process with a configurable number of "salt rounds" to enhance security. This hashed ID is then stored in the database instead of the plain ID. The use of bcrypt ensures that even if the database were compromised, the original IDs remain protected, as the hash cannot be easily deciphered or reverse-engineered. This approach helps maintain the confidentiality and security of sensitive data within our system. To encrypt the fetus's and the patient's ID, we implemented the AES-256-CBC encryption algorithm. This method involves creating a cipher with a secret key and an initialization vector (IV), both of which are securely stored and managed. When an ID is provided, it is encrypted into a hexadecimal format before being stored in the database. This encryption process transforms the plain text ID into an unreadable format, making it highly secure against unauthorized access. Later, when we need to retrieve and verify the ID, the encrypted data is decrypted back into its original form using the same key and IV. This approach ensures that sensitive IDs remain confidential and protected, both at rest and in transit, while allowing our system to perform necessary operations securely and efficiently.

8.11 **MongoDB:** is a NoSQL database known for its flexibility, scalability, and ease of use. Unlike traditional relational databases, MongoDB stores data in a document-oriented format using BSON (Binary JSON), which allows for a more flexible and dynamic schema. This makes MongoDB particularly well-suited for applications that require the storage and retrieval of large volumes of unstructured or semi-structured data. Its ability to handle horizontal scaling and its rich querying capabilities make it a popular choice for modern web and mobile applications.

8.12 **Atlas**: MongoDB Atlas is a cloud-based database service that provides an easy and scalable way to host and manage MongoDB databases. It offers a fully managed environment, handling the infrastructure, updates, backups, and scalability, allowing developers to focus on building applications without worrying about database management. Atlas provides a variety of deployment options across multiple cloud providers, with built-in security features like encryption, authentication, and automated backups. It integrates seamlessly with MongoDB, making it an ideal solution for developers who want to leverage MongoDB's flexibility and scalability while taking advantage of a managed service.

8.13 **How Our Database is Built:** In our project, we used MongoDB, hosted on Atlas, to manage and store various types of data critical to our system. Our database is organized into multiple collections, each serving a specific purpose:

- Pediatricians: This collection stores the encrypted of the identification, usernames, passwords, and the medical clinics they are associated with.
- Patients: This collection holds detailed information about patients, such as their birth date, weight at different stages, encrypted identification, and other medical details. This data is crucial for tracking the growth and development of children over time.
- Fetus: This collection stores data related to fetal growth, including measurements at various stages of pregnancy, encrypted identification. The information is used to analyze and monitor the health of the fetus throughout its development.
- Codes: This collection is used to store encrypted special codes, possibly related to access or authentication processes within the system.

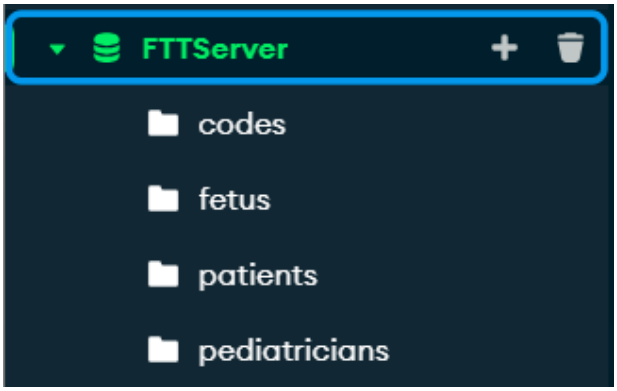

Figure 8: MongoDB database

By structuring our database in this way, we ensure that the application can efficiently manage and retrieve the necessary data for providing accurate diagnostics and maintaining a secure and scalable system. MongoDB's flexibility and Atlas's managed environment have allowed us to build a reliable backend that supports the complex needs of our project.

9. Technologies for deployment

- **DigitalOcean- Virtual Server Hosting:** DigitalOcean is a cloud infrastructure provider that offers virtual private servers, known as Droplets, which function as virtual machines. We used DigitalOcean to purchase and manage our server, providing us with the flexibility to configure it exactly as needed. Through SSH access, we were able to connect to this server and use command-line tools like nano to edit configuration files and manage the server environment directly. This approach allows us to treat the server like a virtual computer, giving us full control over the operating system and software installed, making it ideal for deploying complex applications.

  Nano is a simple, command-line text editor commonly used in Unix-based systems like Linux. It's ideal for making quick edits to configuration files directly on a server, which is crucial during server management and deployment tasks. With its user-friendly interface, Nano allows users to perform essential text editing functions through easy-to-remember keyboard shortcuts, making it accessible even for those less familiar with command-line tools. In deployment scenarios, such as configuring Nginx or adjusting environment variables on a DigitalOcean server, Nano provides a straightforward way to make real-time changes efficiently.

- **Nginx: Efficient Web Server and Reverse Proxy:** Nginx is a powerful and widely used web server that also functions as a reverse proxy server. We used Nginx on our DigitalOcean Droplet to efficiently handle incoming HTTP requests and direct them to the appropriate application services running on the server. Nginx is known for its high performance and low resource usage, making it an excellent choice for serving static content, load balancing, and managing multiple application processes. By configuring Nginx, we ensured that our application can handle a high number of concurrent connections smoothly, improving the overall efficiency and responsiveness of the application.
- **Certbot: Securing the Application with HTTPS:** Certbot is a tool provided by Let's Encrypt that automates the process of obtaining and renewing SSL/TLS certificates for your domain. We used Certbot to secure our application by enabling HTTPS, which encrypts the data transferred between clients and our server. This encryption is critical for protecting sensitive information, such as user credentials and personal data, from being intercepted by unauthorized parties. By integrating Certbot with Nginx, we

ensured that all communication with our application is encrypted and secure, providing users with confidence in the safety of their data.

- **PM2: Process Management and Server Uptime:** PM2 is a process manager for Node.js applications that keeps the server running continuously, even in the event of crashes or restarts. We used PM2 on our DigitalOcean server to manage the application processes, ensuring that they are always active and available to handle user requests. PM2 also provides useful monitoring and logging features, allowing us to keep track of the application's performance and quickly respond to any issues. By using PM2, we ensured that our application remains online and functional 24/7, providing a reliable user experience.
- **Cloudflare: Domain Management and Security:** Cloudflare is a content delivery network (CDN) and domain management service that we use to purchase our domain and manage DNS settings. Cloudflare provides additional security features, such as DDoS protection and a web application firewall, to safeguard our server and application from malicious attacks. By configuring our domain through Cloudflare, we were able to keep our server's IP address hidden and route traffic through Cloudflare's network, ensuring that our application is both accessible and protected under a single, secure domain name.
- **How They Connect Together:** Each of these components plays a crucial role in the deployment and operation of our application. DigitalOcean provides the foundational virtual server where everything is hosted. Within this server, Nginx efficiently manages web traffic, while Certbot ensures that all communication is encrypted. PM2 keeps our application processes running smoothly and continuously, while Cloudflare manages our domain and provides additional security. Together, these tools create a robust, secure, and reliable environment for our application, ensuring that it performs well under various conditions and always remains accessible to users.

10. Interface with the client throughout development

Throughout the frontend development process, we consistently engaged with pediatricians and regular users, such as parents or other non-pediatricians, to create a user-friendly interface tailored to all users. Given that this is a medical system, we selected calming colors that are easy on the eyes, such as white, light blue, dark blue, and soft green. We reserved the color red for specific elements, like the logout button or for displaying important messages to the user.

Understanding that pediatricians spend many hours in the clinic in front of a computer while treating children, we designed the entire user interface to be straightforward and free of unnecessary effects that could disrupt their work. The demo system was designed with the same color scheme and layout as the pediatrician system to familiarize users with the healthcare environment.

Before deployment, we allowed users to test the site locally on our computers and gathered their feedback to refine the system interface. One parent suggested including instructions in the training video on how to obtain the quantitative data needed for diagnosis, which we promptly added.

After completing the deployment process, we met with Prof. Shaul Ron, a specialist in pediatric gastroenterology at Rambam Health Care Campus. The professor tested the system, he noted that the user interface is intuitive, easy to use, and well-suited for pediatricians. Regarding the system's functionality and accuracy, he affirmed that the part of the diagnosing based on formulas it reliably diagnoses conditions and, in most cases, provides correct diagnoses. Additionally, he mentioned that the nutrition plan suggested by the AI tool was helpful in recommending certain foods to parents that could aid in their child's weight gain.

We took all user comments and feedback seriously and made improvements to the system's user interface accordingly, ultimately creating a system that is highly adapted to all users and maximizes the overall user experience.

11. User Experimentation and Testing Methodology

To ensure that the navigation and user authentication processes on the FTTell website function as intended, we conducted a comprehensive testing process focusing on user interaction, system responses, and security protocols.

- **Navigation Test:** We began by evaluating the navigation bar, which includes links to the Home, About, FAQ, and Tutorial sections. The primary objective was to ensure that all navigation links worked seamlessly across different devices and browsers. We used a combination of manual testing and automated tools to simulate various user interactions. Each link was clicked multiple times to check for broken links, proper page rendering, and responsive design across screen sizes.
- **User Authentication and Access Control:** Given that only pediatricians are allowed access to the system, we placed particular emphasis on testing the authentication mechanisms. We tested the registration and login processes by attempting to create accounts with both valid and invalid credentials. During the sign-up process, we ensured that all password criteria were correctly enforced and that error messages appeared for non-compliant passwords. We also tested the email-based access code system by attempting to log in with and without the required access code. The system was tested to verify that only users who had received an access code via email could proceed past the login screen.
- **Private Routes and Security Testing:** To verify the security of private routes, we tested unauthorized access attempts. We tried to manually navigate to restricted sections, such as the home or dashboard pages, by directly entering the URL in the browser. The system successfully redirected unauthorized users back to the login page, confirming that non-pediatrician users could not bypass the authentication process.

  In addition, we used Postman to test the communication between the frontend and backend. We ensured that all POST requests related to user authentication were handled correctly, with proper responses being sent back to the client.

  Through this rigorous testing process, we validated that the navigation and authentication systems work as intended, providing a secure and user-friendly experience for pediatricians using the FTTell system.

12. Testing formulas and diagnostic accuracy for Non-Pediatrician users

In the next phase of testing, we focused on validating the accuracy and reliability of the diagnostic formulas used in the FTTell system, specifically for non-pediatrician users who utilize the demo diagnosis feature. This testing involved inputting various data sets into the system and verifying that the results were both accurate and consistent.

- **Data input and formula validation:** We started by entering data for two patients: one exhibiting symptoms of Failure to Thrive (FTT) and another with normal growth patterns. The input fields included critical metrics such as birth weight, weight at various milestones (6, 12, 24 months, etc.), and maternal information such as height, weight, and age.
  We thoroughly tested the system's ability to calculate percentile ranks and growth indicators based on this input data. The formulas were tested for their ability to handle different combinations of data, including extreme cases (e.g., very low or very high weights) to ensure the robustness of the algorithm.
- **Diagnose result accuracy:** For the patient exhibiting FTT symptoms, the system correctly identified a percentile drop and provided a severity score indicative of potential growth issues. The final report generated by the system included a clear indication of the problem, with specific notes on fetal growth and birth weight. The results were cross-verified against expected outcomes based on the input data, confirming that the system's diagnostic capabilities were functioning as intended.
  For the patient without FTT, the system accurately reflected normal growth patterns, showing no percentile drop and generating a severity score that aligned with typical developmental milestones.
- **Non-Pediatrician access and demo mode testing:** We also conducted testing on the demo mode, which allows non-pediatrician users to simulate a diagnosis without actual access to the full system. The purpose of this testing was to ensure that demo users received accurate and representative results based on the data they entered.
  The system was tested to confirm that the demo results did not differ in accuracy from those obtained by pediatrician users. All diagnostic outputs, whether in demo mode or full access, were consistent and reliable, ensuring that the system provides valuable insights regardless of the user's access level.

- **Data persistence testing**: We simulated multiple demo diagnoses using non-pediatrician accounts. After each demo, we checked the database to confirm that no user data from these sessions had been stored. This was verified by running queries on the database to search for any records matching the inputs provided during the demo sessions.

  The system behaved as expected, with no demo data being saved to the database. This confirms that the system properly isolates demo users' interactions from the production data, ensuring that only data from verified pediatrician users is stored and processed.

  This data handling mechanism is crucial for maintaining the security and privacy of the system, preventing unauthorized users from impacting the integrity of the database while still allowing them to explore the functionality of the system through the demo mode.

13. Testing the diagnose process for pediatricians

The final and most critical part of the FTTell system testing involved evaluating the complete diagnosis process for pediatricians. This phase was focused on ensuring that pediatricians could successfully perform a diagnosis using the system, including data input, emotion detection, and generating final diagnostic reports and nutrition plans.

- **Diagnosis process testing:**

1. **Starting the Diagnosis:** Pediatricians begin the diagnosis process by entering the patient's ID, which initiates the workflow. We tested this functionality by inputting various patient IDs to ensure that the system correctly pulls up the corresponding patient's data or prompts the user to enter new data if the patient is new.
2. **Fetal and Child Data Entry:** Pediatricians are required to input comprehensive data, including fetal mass and length at different stages, the child's weight at various age milestones, and maternal information. Each input field was tested rigorously for validation checks, ensuring that all required fields were properly filled and that the data was saved accurately to the database.

   We entered both typical and edge-case data (e.g., extremely low or high weights) to verify that the system's calculations and diagnostics remained accurate under all conditions.

- **Emotion Recognition Test:** For children aged 6-25 months, an emotion recognition test is integrated into the diagnosis process. Pediatricians are instructed to use a connected camera to capture the child's facial expressions during various interactions. We tested this feature by running the system with different camera setups, ensuring that the video feed was correctly processed and that the emotion recognition software functioned as intended.

  The system was verified to correctly interpret and record facial movements, providing meaningful data for the final diagnostic report. We tested the emotion detection algorithm by presenting various emotional stimuli to the child and comparing the system's results with expected outcomes.

- **Final Diagnostic Report:** After completing the input and emotion detection phases, the system generates a comprehensive diagnostic report. This report includes percentile rankings, severity scores, and potential growth issues

based on the input data. The report also provides insights into the child's emotional responses, including the most frequently detected emotions.

We ensured that the final reports were accurate and reflective of the input data. Each element of the report was cross-verified against known standards for pediatric health assessments.

- **Nutrition Plan:** For children diagnosed with potential growth issues, the system provides a personalized nutrition plan generated by chat gpt 3.5 turbo model. We tested the generation of these plans, ensuring they catered to specific dietary needs based on allergies and other input data (e.g., lactose intolerance).
- **Final outcome:** The testing confirmed that the FTTell system provides pediatricians with a reliable, comprehensive tool for diagnosing FTT and other growth-related issues in children. The integration of emotion detection adds an additional layer of insight, helping to capture subtle emotional cues that might correlate with physical growth challenges. The system is robust, accurate, and user-friendly, allowing pediatricians to focus on providing the best care possible while leveraging advanced technological support.

14. Networking and API testing with Postman

To ensure seamless communication between the client and server and verify the correct operation of the MongoDB database through Mongoose, we conducted comprehensive API testing using Postman.

- **Testing process: API Endpoint Verification:** Each API endpoint was tested to ensure it responded correctly to various HTTP requests (GET, POST, PUT, DELETE). We started by verifying the connection to the server, checking that all endpoints were reachable and returned the expected status codes.
- **Data Retrieval (GET Requests):** We tested all GET endpoints by requesting data from the database. This included retrieving patient records, fetching diagnostic results, and accessing predefined lists like the nutrition plans. The responses were checked for accuracy and completeness, ensuring that the data returned matched what was expected based on the test inputs.
- **Data Submission (POST Requests):** The POST endpoints were tested to ensure that data submitted through the frontend was correctly processed and stored in the MongoDB database. This involved submitting new patient data, diagnostic inputs, and other related information through Postman. We checked the database after each test to confirm that the data was correctly saved and that no unintended data overwrites or errors occurred.
- **Data Updates (PUT Requests):** We tested the PUT endpoints to confirm that existing records could be updated correctly. This included modifying patient information, updating diagnostic results, and revising saved records. The system's response was verified to ensure that only the intended records were updated without affecting other data.
- **Data Deletion (DELETE Requests):** The DELETE requests were tested to ensure that records could be removed from the database when required. We verified that the correct records were deleted and that the system handled any dependencies or related data appropriately, preventing orphaned records or broken references.
- **Error Handling:** We deliberately tested error scenarios, such as submitting incomplete data, using incorrect endpoints, or attempting unauthorized access. The system's error messages and status codes were checked to ensure they were clear, accurate, and helpful for debugging and user guidance.
- **Results:**
  The API testing confirmed that the networking layer of the FTTell system functions correctly, with all requests being accurately processed and responded

to by the server. Data integrity was maintained throughout, with MongoDB and Mongoose handling the database operations efficiently and reliably. The use of Postman allowed for a detailed examination of each request and response, ensuring that the system is robust and ready for real-world use.

This detailed testing process helps to ensure that pediatricians and other authorized users can interact with the system confidently, knowing that the data they input and retrieve is handled correctly by both the client and server sides of the application.

15. Results and Conclusion

- **achieving the project goals:**

  At the outset of our project development journey, we set a central goal: to create a system that would be exemplary in every aspect—algorithm accuracy, security, and user experience. The primary objective was to develop an innovative system for the early diagnosis and treatment of growth problems in children aged 0-60 months, specifically targeting Failure to Thrive (FTT). We aimed to build a user-friendly system tailored to all types of users, particularly doctors, ensuring that it not only performed effectively but also met the highest standards of security and usability.

  We are now pleased to report that we have largely achieved these goals. The algorithm, as detailed in the evaluation and testing chapters, consistently delivers an accuracy rate exceeding 88%, providing reliable identification of children at risk of FTT and delivering personalized treatment plans based on evidence-based recommendations. This level of precision is crucial for effective diagnosis and underscores the robustness of our approach.

  In terms of security, we have implemented stringent measures to protect all sensitive information, such as patient and doctor IDs and passwords. These are encrypted using advanced encryption techniques, ensuring that the data is secure both at rest and in transit. Furthermore, all communication between the client and server is safeguarded by HTTPS, providing an additional layer of protection against potential threats.

  From a design perspective, we prioritized creating an intuitive and accessible user interface. After allowing real users to test the system, we received positive feedback affirming that the interface is both convenient and easy to navigate. This validation from actual users reinforces our belief that we have succeeded in designing a system that meets the practical needs of healthcare professionals while maintaining simplicity and ease of use.

  Overall, we have developed a system that not only meets but exceeds our initial aspirations, offering a powerful, secure, and user-friendly tool for pediatric diagnostics.

- **Considerations in Decision-Making:**

  Throughout our project, our approach to decision-making was always guided by thorough research and careful consideration of all available options. Before making any significant decision, we meticulously explored the advantages and disadvantages of each potential path. This systematic evaluation ensured that we could choose the most effective solutions that aligned with our project goals. Additionally, we always considered the remaining time available to us, carefully assessing whether the proposed options could be realistically implemented within our time constraints. In instances where we faced difficult decisions or when there were multiple viable options, we sought guidance from our supervisor, Dr. Natalie Levy, whose expertise was invaluable in helping us navigate these challenges. Her input was particularly crucial in areas where we needed to balance competing priorities, such as algorithm accuracy, security, and user experience. Our primary considerations included ensuring that the system provided highly accurate and reliable diagnoses for Failure to Thrive (FTT) in children, which led us to rigorously test and select algorithms that could achieve an accuracy rate of over 88%. Security was another paramount concern, given the sensitive nature of the data being handled. We prioritized the use of advanced encryption techniques and secure communication protocols, such as HTTPS, to protect data both in transit and at rest. Additionally, we focused on creating a user-friendly system that would be intuitive and accessible for pediatricians, which influenced our design and interface decisions. Feedback from real users confirmed that our design choices were effective in making the system convenient to use. Scalability and maintainability were also key factors, driving our selection of technologies that would allow the system to be easily updated and expanded as needed. Overall, our decision-making process was characterized by a deliberate and informed approach, supported by expert consultation, time management considerations, and driven by the goal of developing a robust, secure, and user-friendly system for pediatric healthcare.

- **Reflections on Our Development Process:**

  We can say that, overall, we worked correctly and effectively, successfully meeting all the deadlines we set for ourselves. However, there are certainly a few things we could have changed or considered adding in hindsight. For example, we worked on access control only after implementing all the pages, leaving it for the end of the development process. We did not anticipate that this would cause delays, but it ended up affecting our timeline. Furthermore, if we had more development time, we might have added additional features for the diagnosis,

such as a check for the frequency of a child's hand-to-mouth gestures, as we identified that this could also be an indicator of growth issues. Moreover, we postponed the cloud deployment to the final stage of development, assuming it wouldn't pose many challenges. However, this decision led to unexpected issues that added an extra week to our development timeline. From this experience, we've learned that for future projects, we should begin with the deployment process and then proceed with the rest of the development tasks.

However, it is important to emphasize that, in general, we worked well, followed a solid process, and met all our goals within the set timeframes.

16. Evaluation

For the purpose of evaluating the model and ensuring the correctness of the algorithm, we conducted tests on a sample of 100 children. The data for this evaluation was provided by our supervisor, Dr. Natalie Levy, who developed the underlying formulas for our model.

It is important to highlight that these formulas were rigorously tested in collaboration with Professor Ron Shaul, a specialist in pediatric gastroenterology, and they demonstrated an accuracy of 88%. Our evaluation of the model confirmed that the component based on these formulas also achieved an accuracy of 88%.

For the second part of the system, which is exclusively accessible to pediatricians, we conducted an emotion recognition test on a group of children aged 6-25 months, including both healthy children and those suffering from Failure to Thrive (FTT). The model demonstrated an impressive accuracy of 90% in this test.

- **Performance Metrics Analysis of FTTell Model:**

1. **Accuracy** – the ratio of the correctly labeled subjects to the whole pool of subjects.

$$\boldsymbol{accuracy} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{TP} + \boldsymbol{TN} + \boldsymbol{FP} + \boldsymbol{FN}} = \frac{\mathbf{59 + 28}}{\mathbf{59 + 28 + 9 + 4}} = \mathbf{0.87}$$

2. **Precision** – the ratio of the correctly labeled by our model to all labeled.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}} = \frac{\mathbf{59}}{\mathbf{59 + 4}} = \mathbf{0.936}$$

3. **Sensitivity** – the ratio between the number of infants that were correctly labeled by FTTell as having FTT and all the infants that actually had FTT.

$$\boldsymbol{Sensitivity} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}} = \frac{\mathbf{59}}{\mathbf{59 + 9}} = \mathbf{0.867}$$

4. **F1 score** – the harmonic mean (average) of the precision and sensitivity.

$$\boldsymbol{F1} = \frac{\mathbf{2} * \boldsymbol{precision} * \text{sensitivity}}{\boldsymbol{precision} + \text{sensitivity}} = \frac{\mathbf{2 * 0.936 * 0.867}}{\mathbf{0.936 + 0.867}} = \mathbf{0.9001}$$

5. **Specificity-** the ratio between the number of infants that were correctly labeled by FTTell as not having FTT and all the infants that actually did not have FTT.

$$\boldsymbol{Specificity} = \frac{\boldsymbol{TN}}{\boldsymbol{TN + FP}} = \frac{\boldsymbol{28}}{\boldsymbol{28 + 4}} = \boldsymbol{0.875}$$

17.User documentation

Welcome to the FTTell user documentation. FTTell is an advanced diagnostic system designed to assist in identifying and managing Failure to Thrive (FTT) in children aged 0-60 months. The system integrates clinical data analysis with computer vision to evaluate growth patterns and facial expressions, providing accurate diagnoses and tailored treatment suggestions. This documentation will guide you through the features and functionalities of FTTell, offering step-by-step instructions to help you make the most of the system. Whether you're entering patient data, performing emotion recognition tests, or reviewing diagnostic outcomes, this guide has you covered.

Please note that this documentation is divided into two main sections. The first section is intended for site visitors and any user who wishes to get an indication of their child's growth status, without including the emotion recognition test. The second section is specifically designed for qualified pediatricians

17.1 Documentation for users

To visit our system and benefit from the services offered, please click the following link: https://www.fttell.org/

- **Landing Page:**

Figure 9: Landing page

On the landing page, you will find a general and concise description of Failure to Thrive. If you are not familiar with this medical condition, it is recommended that you read the description.

- **Nav bar:**
  - **Home:**

Figure 10: Home navigator

By clicking on "HOME" you will return to the landing page.

- **FAQ:**

Figure 11: FAQ navigator

By clicking on "FAQ" you will render to the FAQ page.

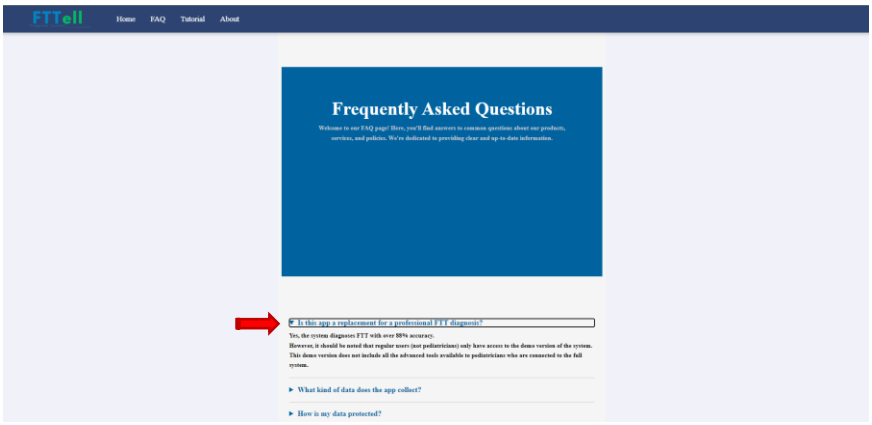

Figure 12: FAQ page

This page shows the most frequently asked questions, in order to see the answer for each question that has been asked click on the arrow on the left side of the question.

- **Tutorial:**

Figure 13: Tutorial navigator

By clicking on "Tutorial" you will render to the tutorial page.

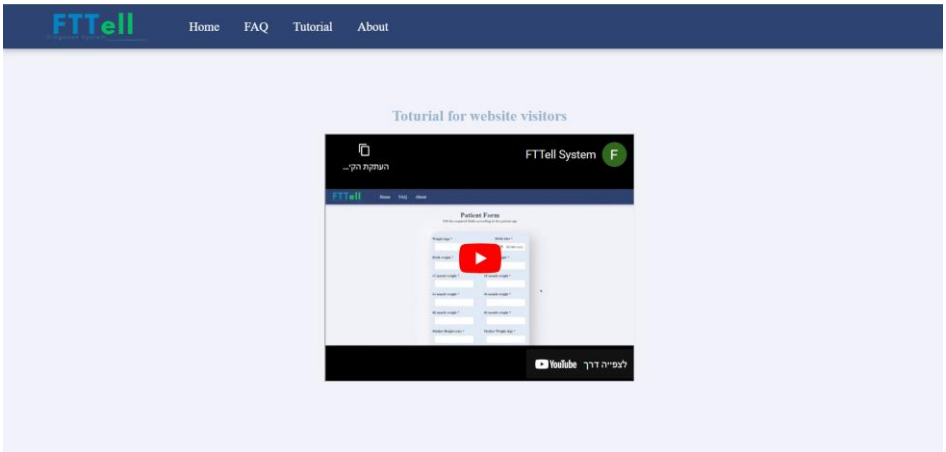

Figure 14: Tutorial page

On this screen there are two videos, the first explains how a regular user without an account can diagnose his child and get an indication of his growth status.
The second video shows how pediatricians can register and log into the system, diagnose a patient and perform an emotion recognition test if necessary.

- **About:**

Figure 15: About navigator

By clicking on "About" you will render to the about page.

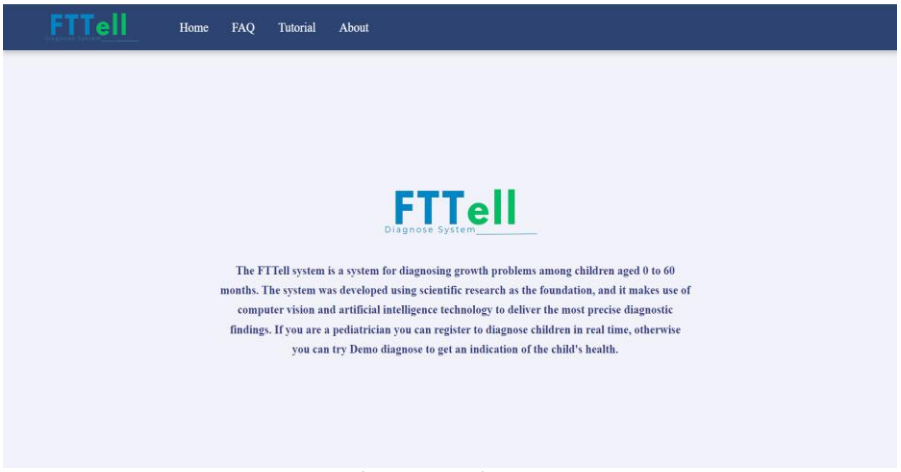

Figure 16: About page

Figure 17: About page

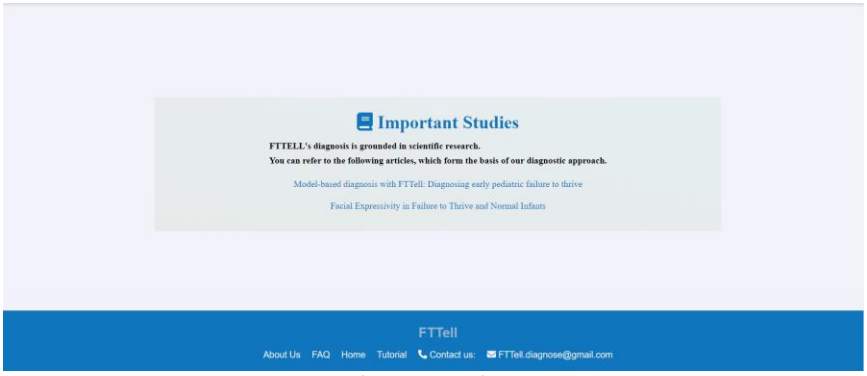

Figure 18: About page

On the "About" page you can find a general description of the FTTell system, an explanation of the system's development team and links to the studies on which the diagnosis is based.

- **Diagnosing Process**

Figure 19: Landing page

In order to start the diagnosing you must press on "Diagnose Patient" button.

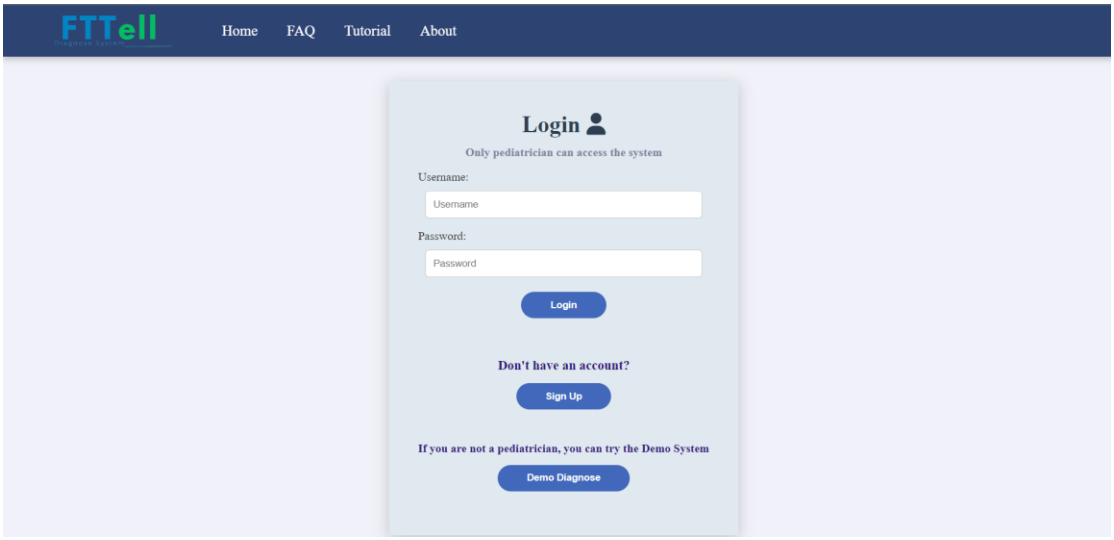

Figure 20: Login page

Once you clicked on "Diagnose Patient" button you will be navigated to the login page.

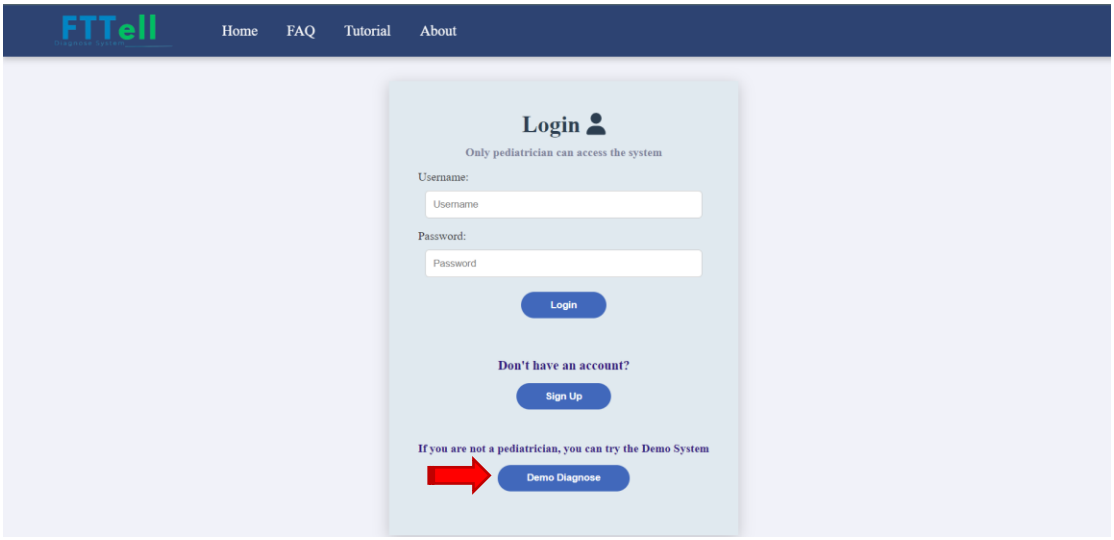

Figure 21: Login page

By clicking on the "Demo Diagnose" button you will proceed to the demo diagnose system.

- **Demo diagnose system**

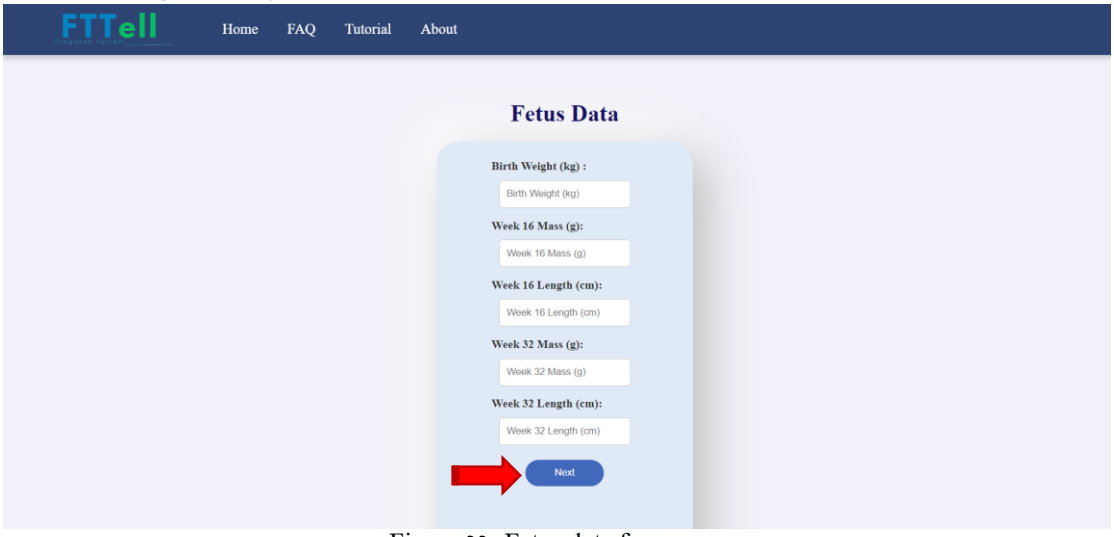

Figure 22: Fetus data form page

On this screen you must fill in all the fields that appear in the form, the fetal data should be kept by every mother during the pregnancy period, they must be filled in carefully and make sure to enter exact values.
Then press the "Next" button to continue the diagnosis, you will proceed to the patient form.

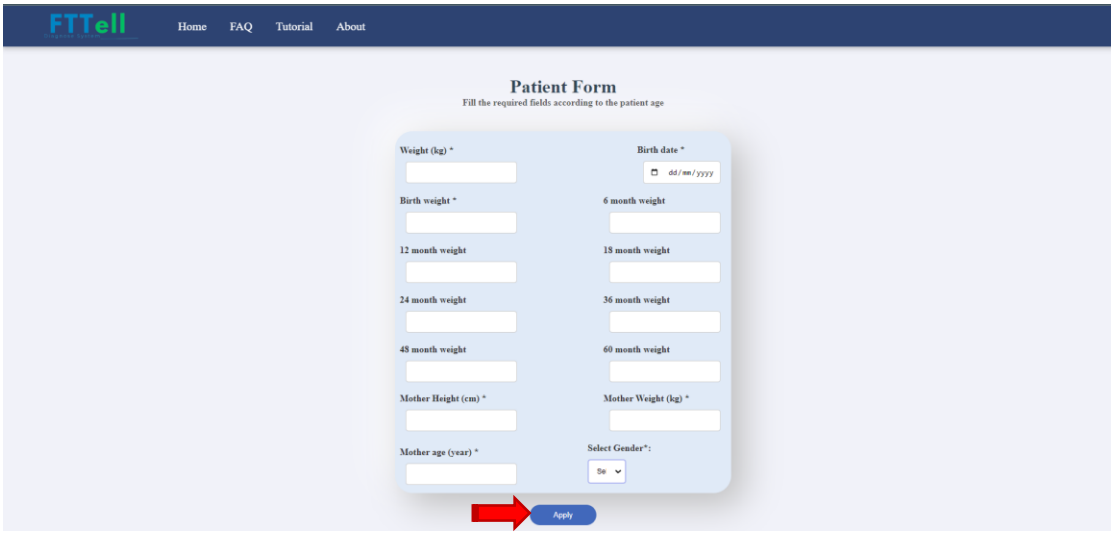

Figure 23: Patient form page

In the patient form you must fill in all the required fields marked with a start.
In the weight field, enter the child's most recent weight.
In the birth weight field, you must enter the same value you typed in the previous screen.
As it appears in the form, there are weight fields for ages 6-60 months, these fields are not required but it is recommended to fill them in, if you do not have these data you can leave them blank.
To finish the diagnostic process, press on the "Apply" button in order to reach the final result test.

- **Final Result:**

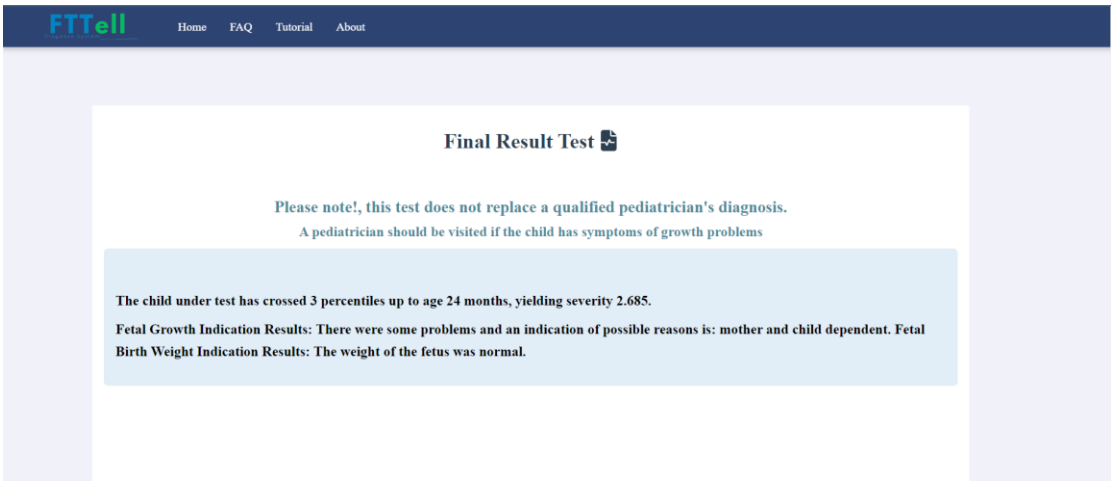

Figure 24: Final result test page

The screen in front of you shows the diagnosis result of a healthy child who is not suffers from FTT, The last two lines show a diagnostic result of the fetal test which can give direction and highlight the cause of the problem and this can sometimes be useful when the child looks healthy according to the diagnosis but the parent suspect that their child has growth problems, in this case the fetal results show that the symptoms that the parent recognizes in his child can be due to the fact that the genetics The child's is born in this way

that reflects symptoms of growth problems but is not a dangerous situation because it is born with the child and is natural.
In any case, the user must note that this test is not a substitute for the diagnosis of a qualified pediatrician, therefore, as soon as there are concerns about the development of growth problems in the child, a pediatrician should be consulted as soon as possible.

17.2 Documentation for pediatricians

To visit our system and benefit from the services offered, please click the following link: https://www.fttell.org/

- **Landing page:**

Figure 25: Landing page

In order to start the diagnosing you must press on "Diagnose Patient" button.

- **Sign up process:**

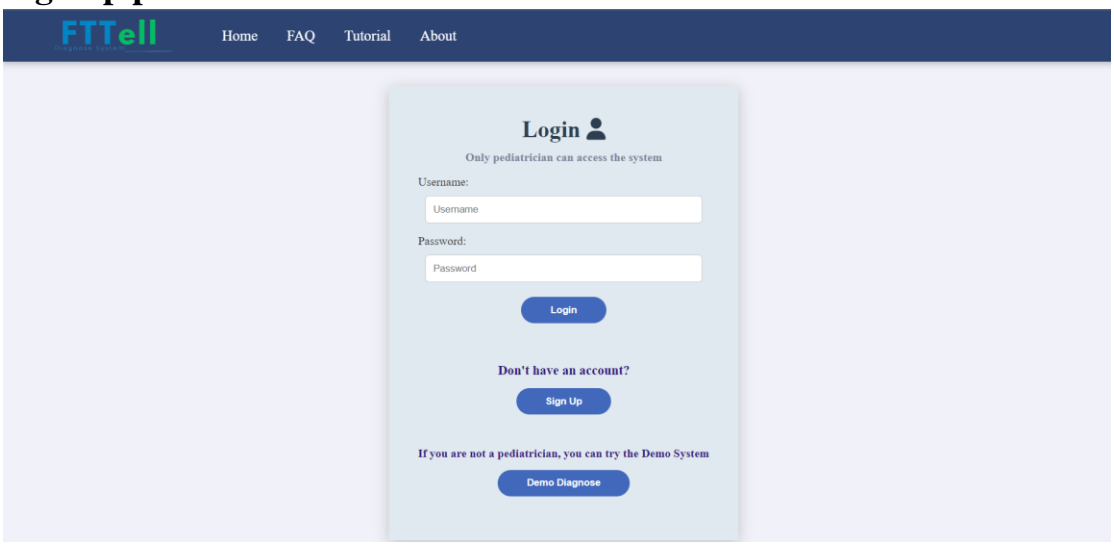

Figure 26: Login page

Once you clicked on "Diagnose Patient" button you will be navigated to the login page.

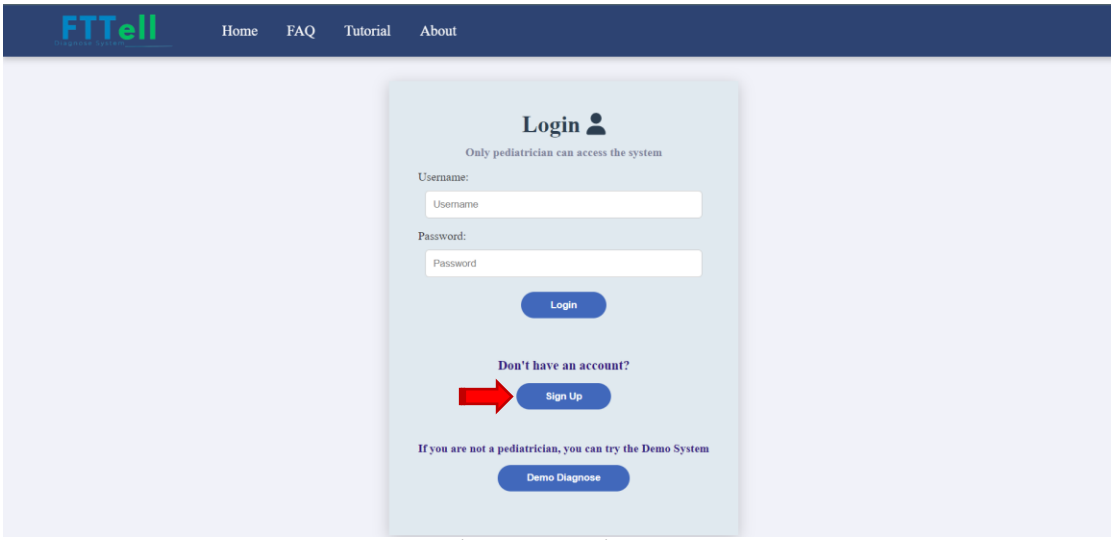

Figure 27: Login page

In order to register for the system, click the "Sign Up" button you will render to the insert code page.

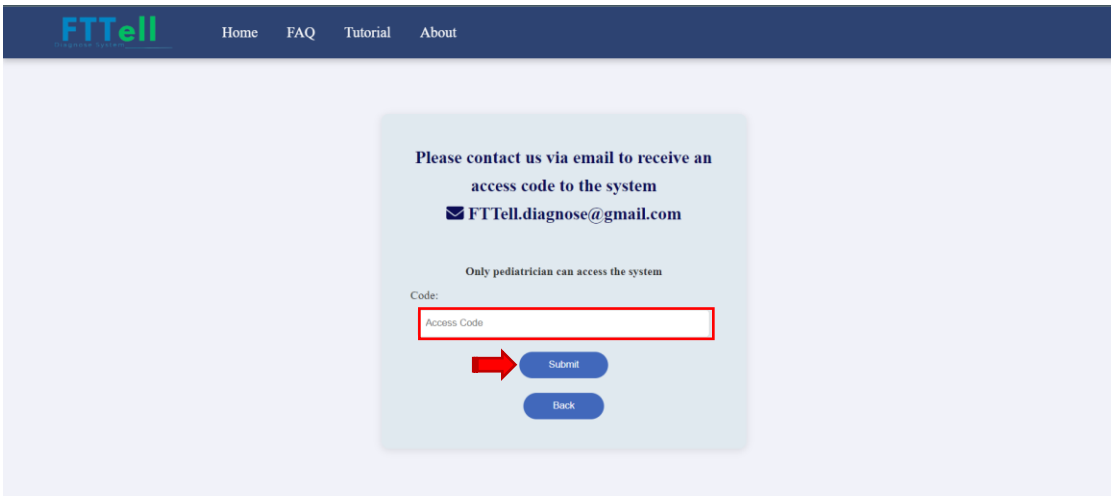

Figure 28: Code for registration

Any pediatrician interested in registering for FTTell and diagnosing patients should contact us via the email on this page: FTTell.diagnose@gmail.com. Documents that verify the doctor's identity, such as a doctor's license and a photocopy of an ID card, must be sent by email.
In a short time our team will verify the identity of the pediatrician, the doctor will receive a secret code back by email, you must enter this code in the field that appears on the screen and press "Submit", then you will be transferred to the registration page to create an account.

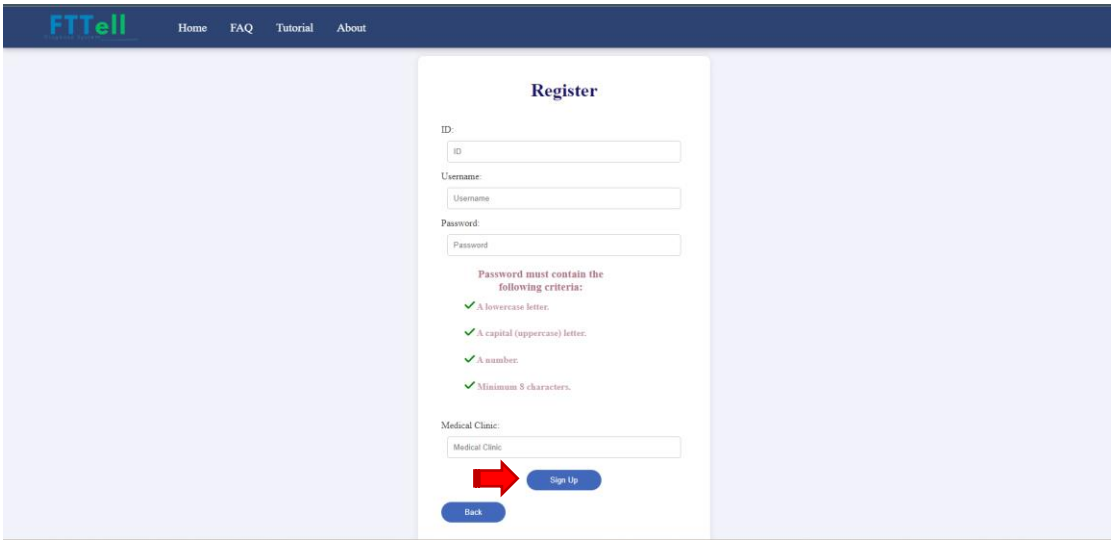

Figure 29: Registration page

In the registration page are required, the password must answer all the requirements for a secure password.
After filling the form click on "Sign Up" button, a pop-up message will appear in order to approve your registration.

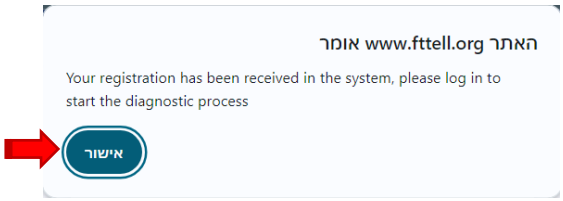

Figure 30: Pop-up message

Once confirming the message, you will render to the login page and you have successfully signed to the system.

- **Login page:**

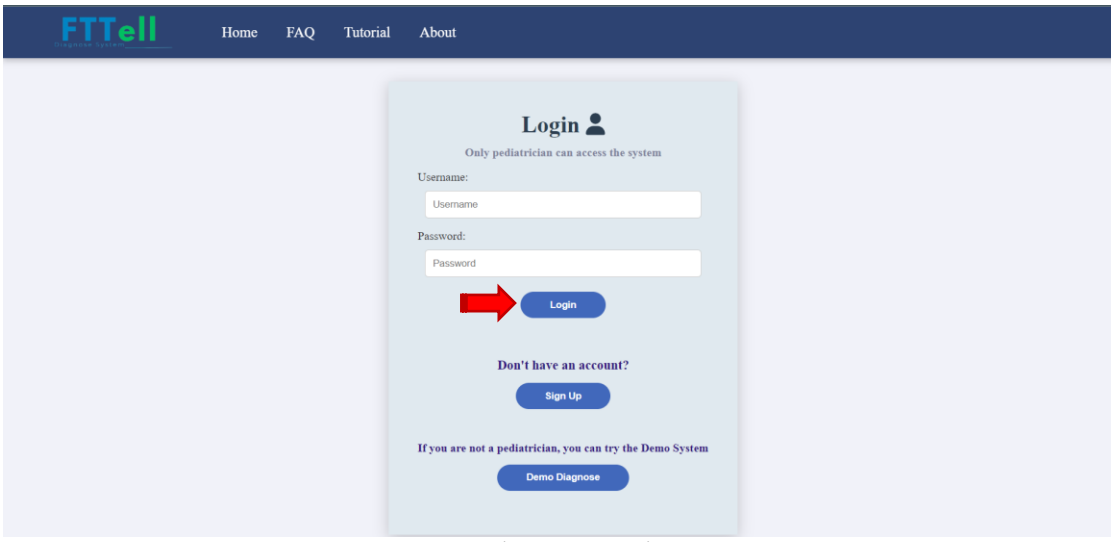

Figure 31: Login page

To log in to the system, enter a valid username and password you created on the registration page and click the "Login" button to enter the pediatrician system home page.

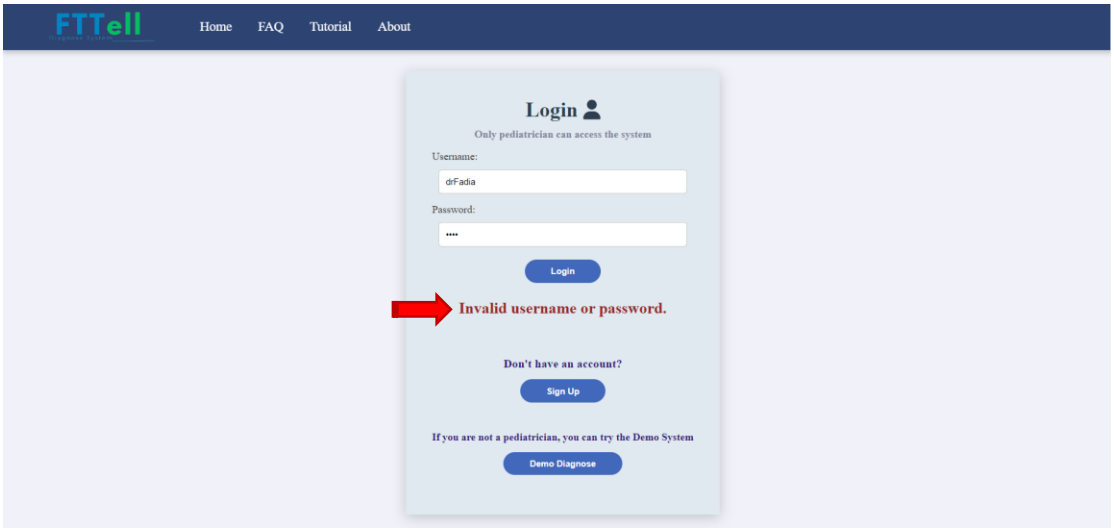

Figure 32: Invalid credential

In case one of the details is incorrect, an appropriate message will be displayed.

- **Pediatrician home page**

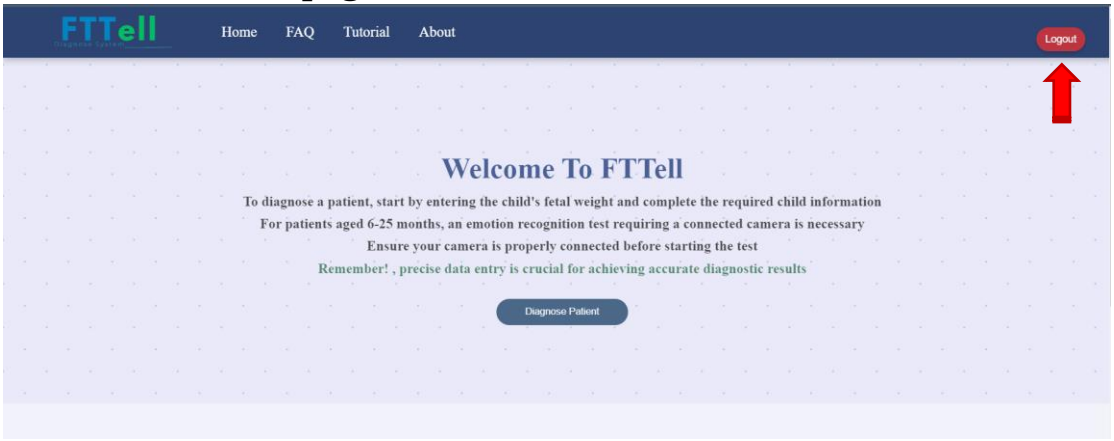

Figure 33: Home page for pediatrician

In order to exit the system, click on the "Logout" button on the right top corner of the page.

- **Diagnosing process**

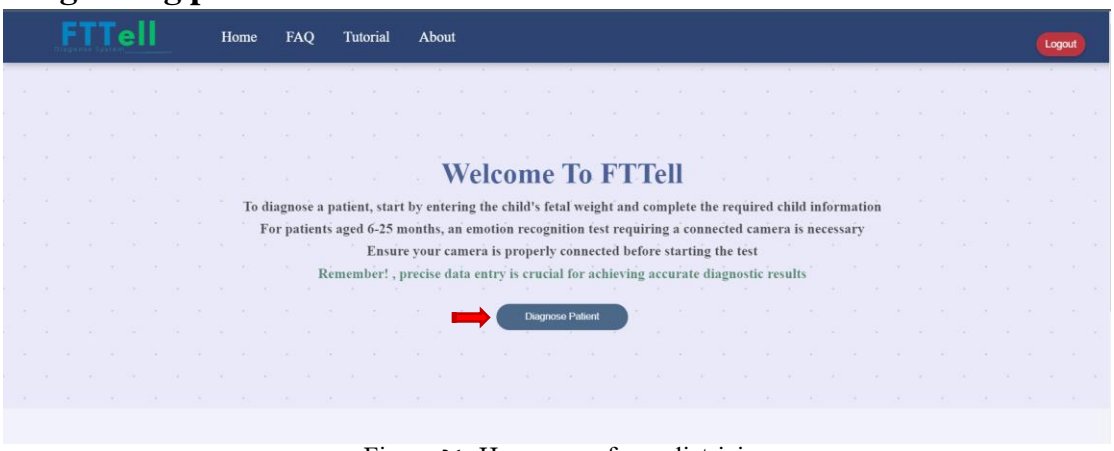

Figure 34: Home page for pediatrician

Click on "Diagnose Patient" button to start the diagnose process.

Figure 35: Patient ID page

The first step of the diagnosing process is entering the patient's ID, then press on "Diagnose" button.

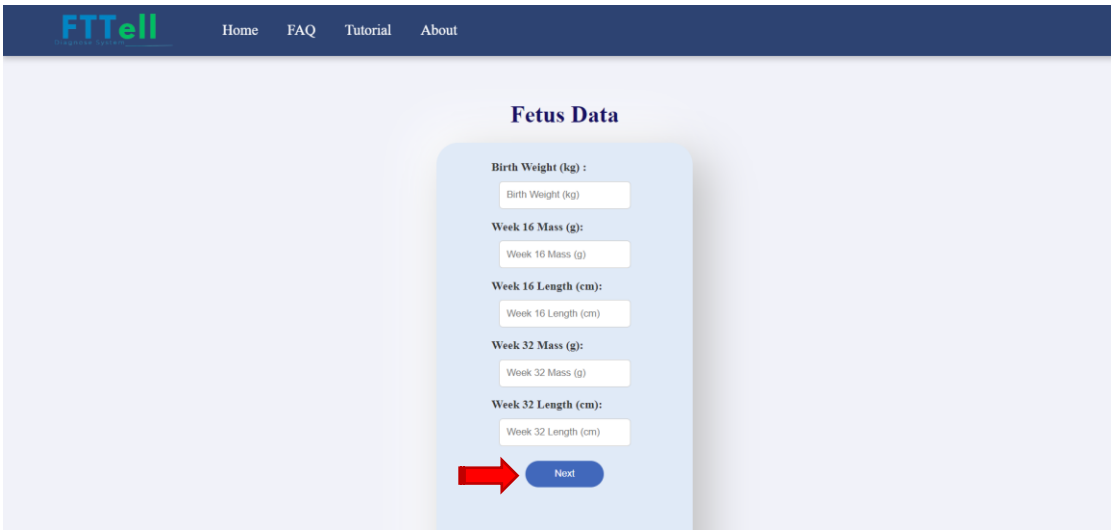

Figure 36: Fetus form

On this screen you must fill in all the missing fields that appear in the form, the fetal data should be kept by every mother during the pregnancy period, they must be filled in carefully and make sure to enter exact values.
Then press the "Next" button to continue the diagnosis, you will proceed to the patient form.

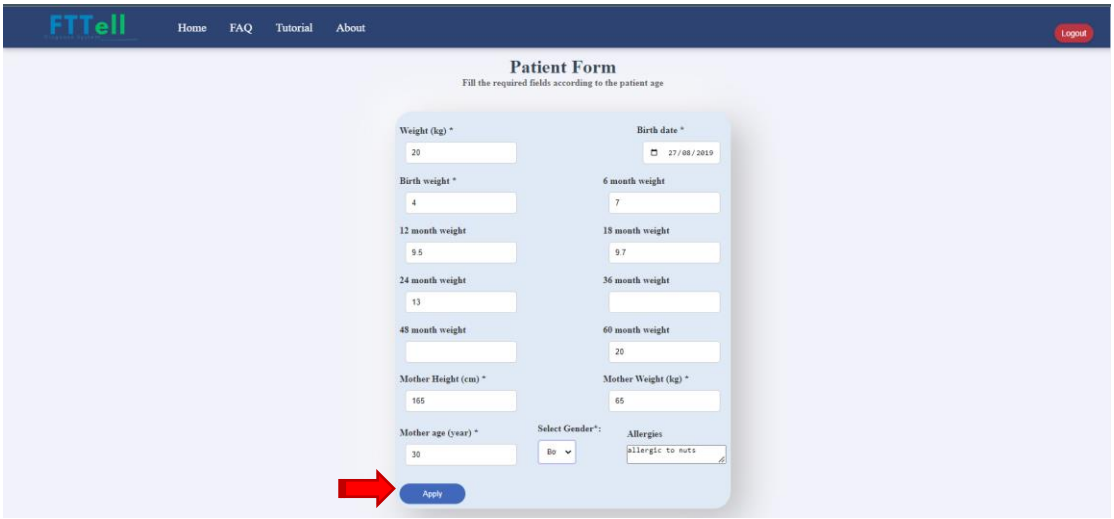

Figure 37: Patient form page

In the patient form you must fill in all the missing required fields marked with a start.
In the weight field, enter the child's most recent weight.
In the birth weight field, you must enter the same value you typed in the previous screen.
As it appears in the form, there are weight fields for ages 6-60 months, these fields are not required but it is recommended to fill them in, if you do not have these data you can leave them blank.
To finish the diagnostic process, press on the "Apply" button in order to reach the final result test.

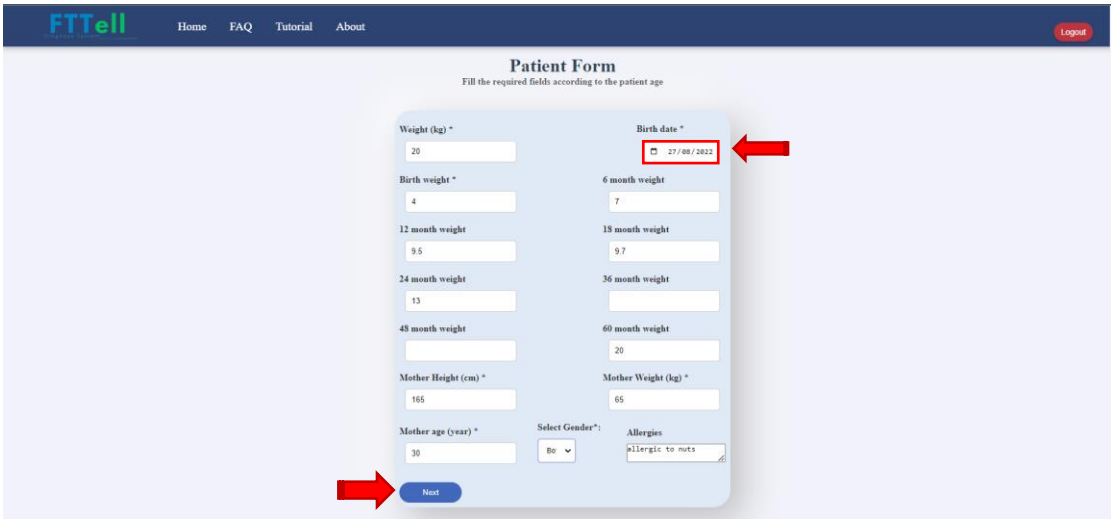

Figure 38: Patient form page

Please note, if the patient is between the ages of 6-25 months, the "Next" button will appear in order to perform an emotion recognition test.
Click the "Next" button to continue the test to the end.

- **Emotional recognition test**

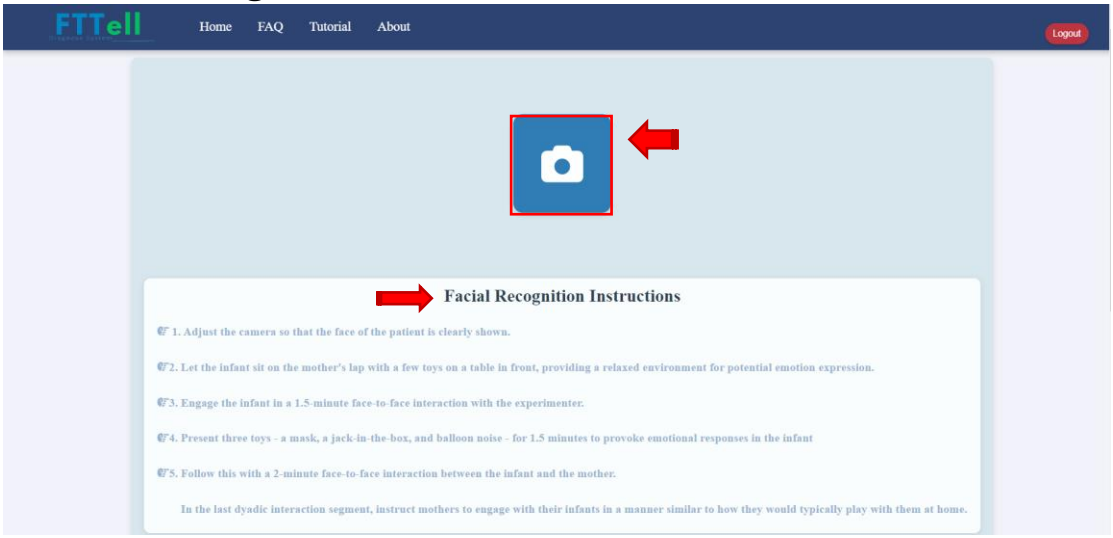

Figure 39: emotion recognition test page

This screen is only relevant for patients in the age range of 6-25 months, as soon as you click on the "Next" button on the patient form page, you will go to the emotion recognition test instructions screen.
On this page we have a detailed instruction for performing the test, they must be read carefully before starting to record the child's face, the doctor must ensure that the camera is turned on and connected to the computer.
To check that the camera is working, click on the camera icon that appears on the page and set the camera at an angle that captures and shows the child's entire face clearly.

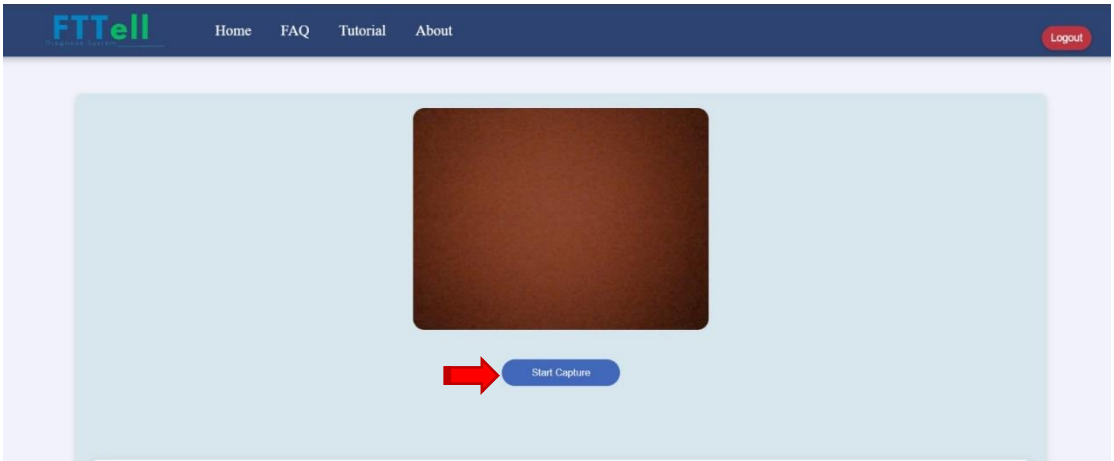

Figure 40: emotion recognition test page

After clicking on the camera icon, you should give a permission in order to use the camera and the camera will be opened automatically.
Before clicking on "Start Capture", adjust the camera in the appropriate place that will capture only the child and especially his face.

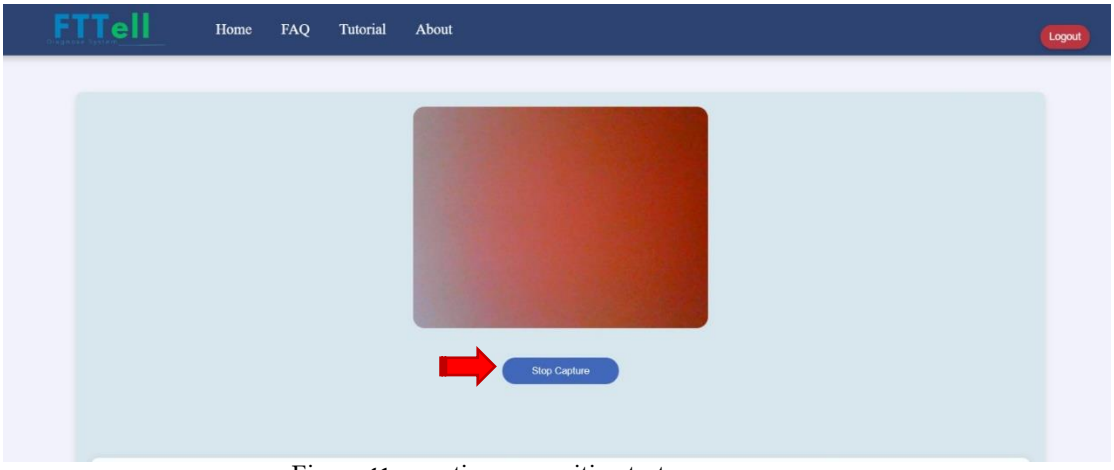

Figure 41: emotion recognition test page

Please record the child and follow the steps explained in the instructions, at any moment you can scroll down and read the instructions and follow them, during the recording you must not press "Stop Capture" before completing all the steps in the instructions.
After completing the test steps, you must click on "Stop Capture" to start the process of decoding the result of the emotion recognition test, this process can take from 5-20 minutes, so you should wait patiently and do not exit the current screen.

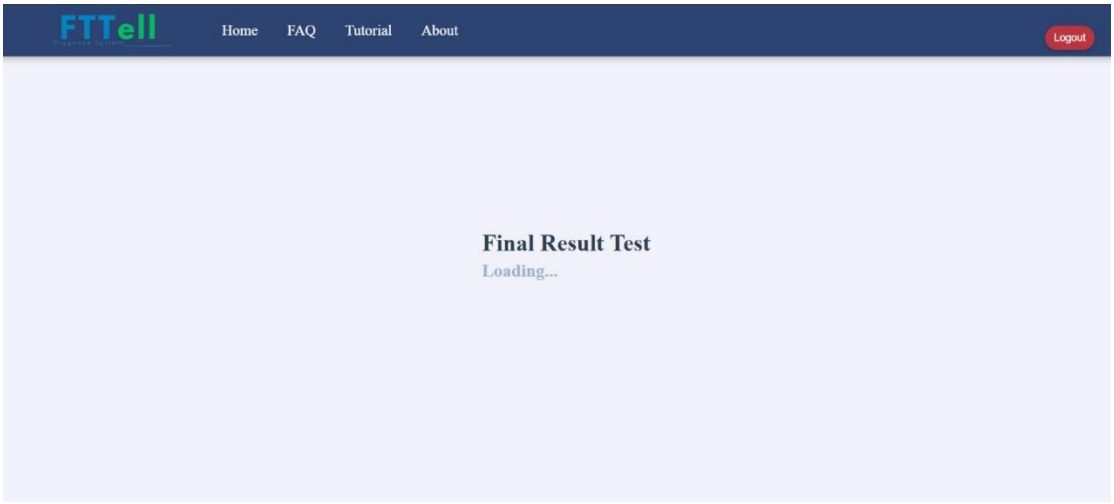

Figure 42: Calculating and processing the data

This page shows that the video is being analyzed, once it is done the final result will be shown on the screen.

- **The final result test**

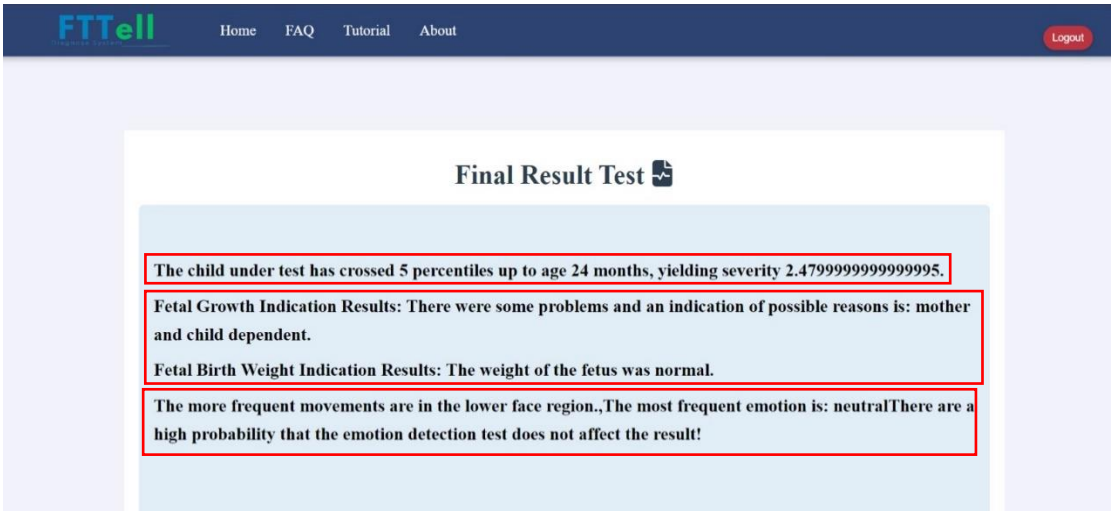

Figure 43: Final result test page

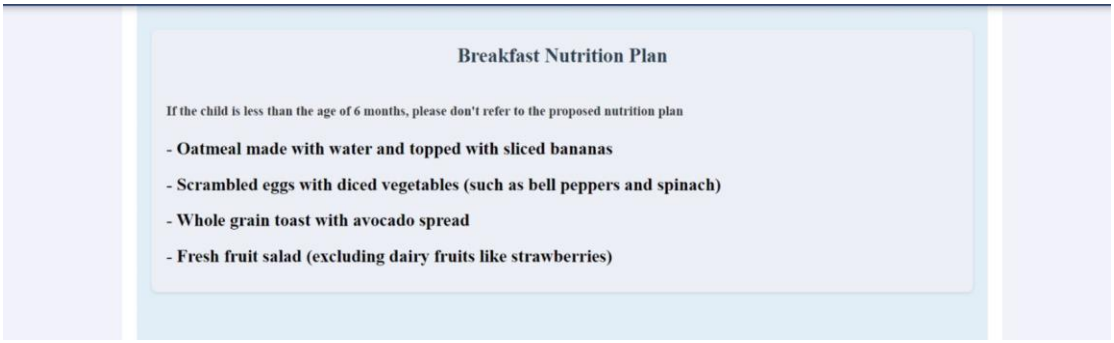

Figure 44: Breakfast nutrition plan

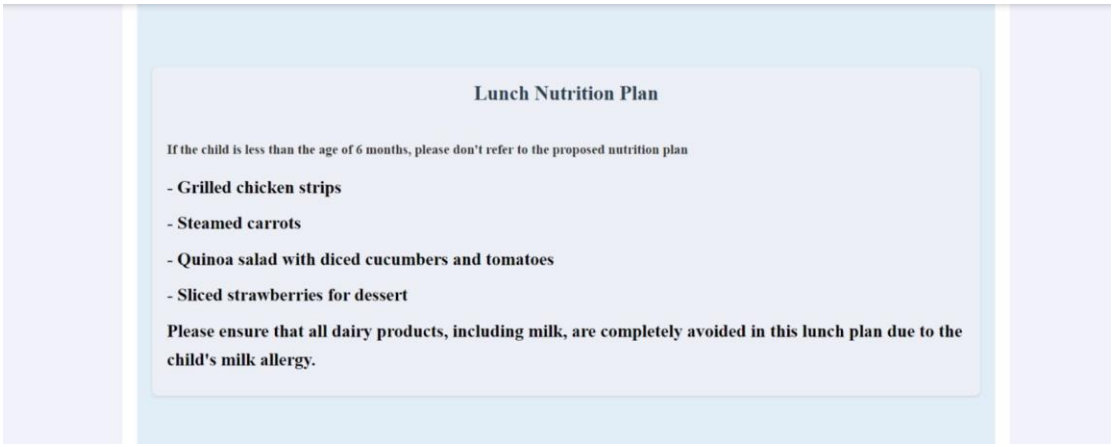

Figure 45: Lunch nutrition plan

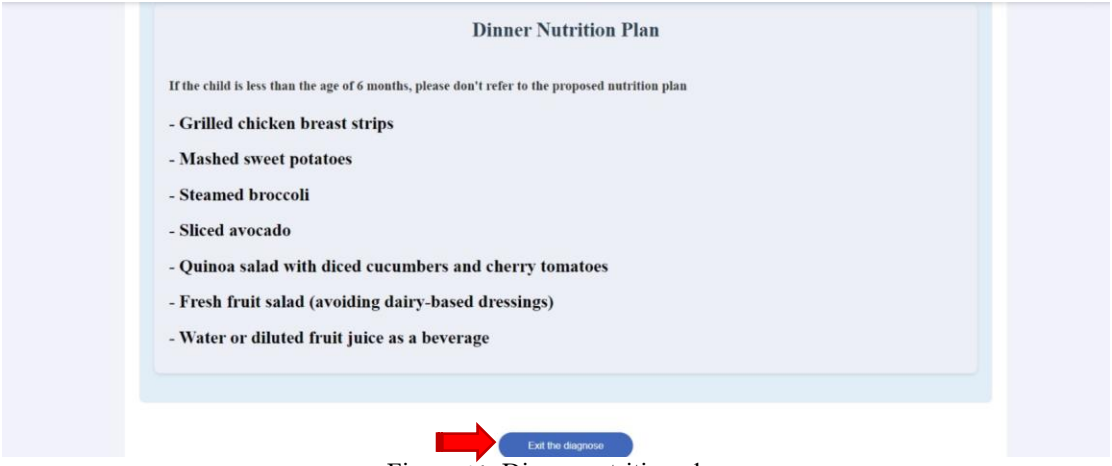

Figure 46: Dinner nutrition plan

At the beginning of the page appears the final result which is divided into 3 parts.
First part: In the first line it is written whether the child suffers from FTT or is healthy, in the attached picture a sick child suffering from FTT with a severity of 2.5 was examined.

Second part: presents fetal growth indication and fetal birth weight indication result, these results are associated with the examination of the child during the fetal period.

Third part: presents the result of the analysis of the child's emotion recognition, in the result it is emphasized which emotion appeared most frequently during the recording and which part of the face (upper/lower) the child used the most frequently, finally it is decided whether the result of the emotion recognition should be taken into consideration or if it is not possible to draw conclusions about the state of the child's growth.

Below the final result, appears a nutrition plan offered by an artificial intelligence tool (ChatGPT), the plan refers to the child's allergies if they exist, it is important to note that if the child is under 6 months old, this nutrition plan is not intended for him.
In all cases, the doctor must consider whether to use the proposed nutrition plan according to his informed considerations.
After reading the diagnosis result, click on "Exit diagnose" in order to secure the data for each patient, this will navigate you back to the home page.

18.Developer documentation

18.1 How to install the libraries from npm

clone from github repository https://github.com/EhsanSarboukh/FTTell-System

git clone https://github.com/EhsanSarboukh/FTTell-System.git

navigate using shell to the frontend folder using command cd

For reading and setting the environment and handling packages use this command in shell: Apt-get update && apt-get install npm.

navigate using shell to the backend folder using command cd

For reading and setting the environment and handling packages use this command in shell: Apt-get update && apt-get install npm.

in the backend folder navigate to folder called Emotion_Detection_FER

in this folder that contains python script that require libraries installations:

npm i python3

python get-pip.py

pip install opencv-python

pip install fer

pip install numpy

pip install mediapipe

pip install tensorflow

navigate to the backend folder:

start the server on localhost:

node server.js

navigate to the frontend folder:

npm start

18.2 package diagram

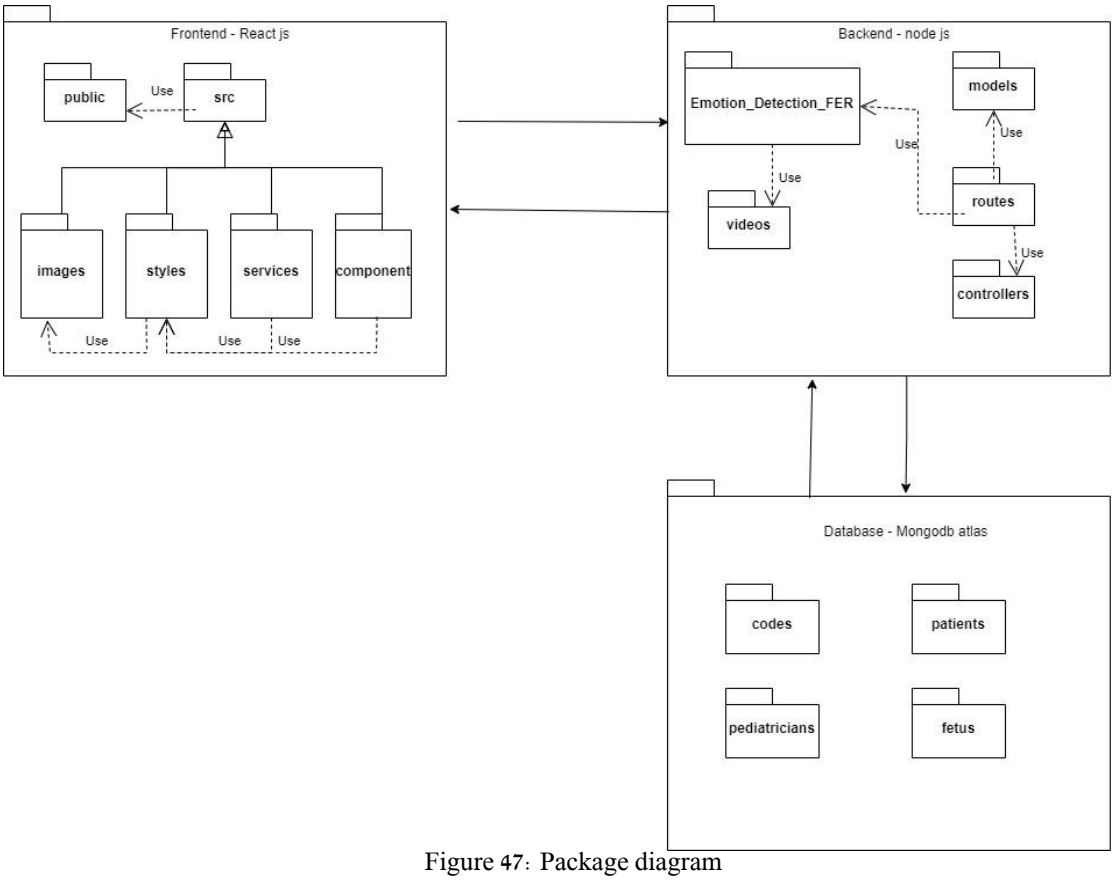

Figure 47: Package diagram

18.3 Project folders hierarchies:

The project is divided into two main folders:

- **Frontend folder:**

Public folder: contains all the static elements of the react web-application

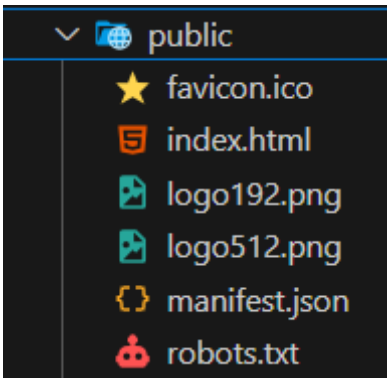

Figure 48: Frontend folder – public folder

src folder: contains multiple elements and folders we will go through each one of them same as the package diagram above

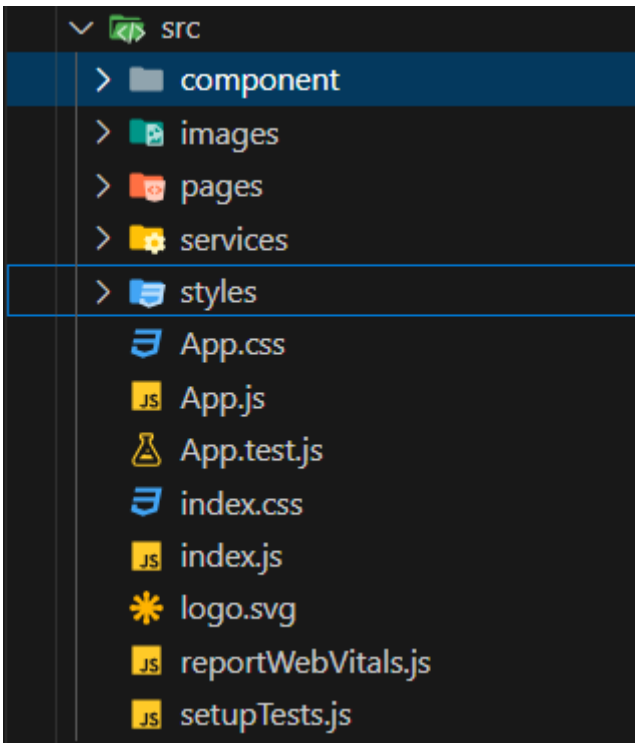

Figure 49: Frontend folder – src folder

Component folder:

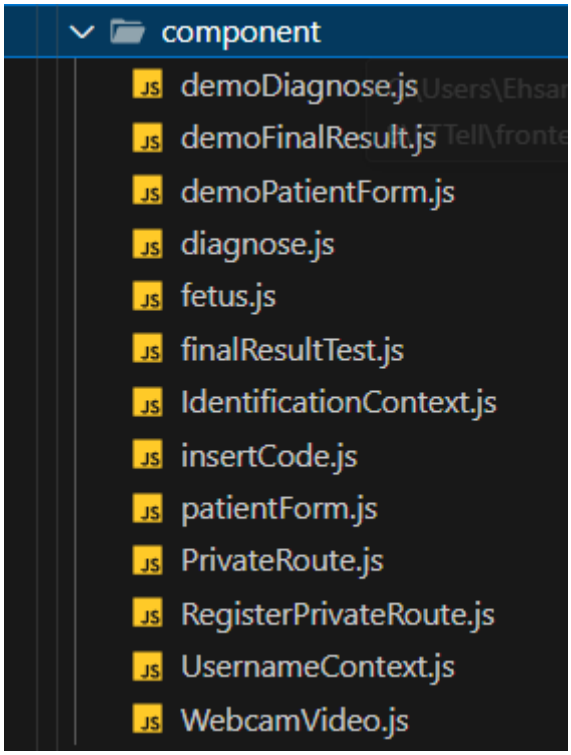

Figure 50: Frontend folder – Component folder

images folder:

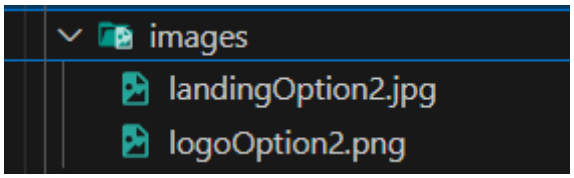

Figure 51: Frontend folder – images folder

pages folder:

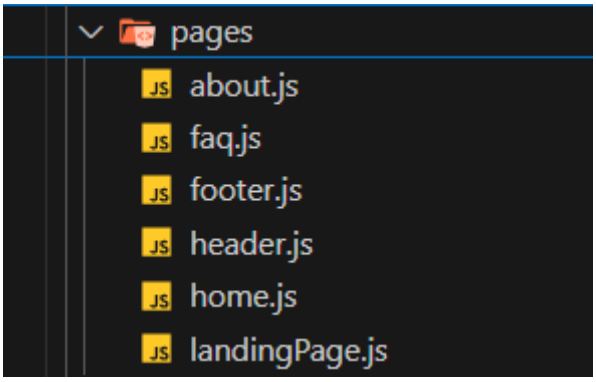

Figure 52: Frontend folder – pages folder

services folder:

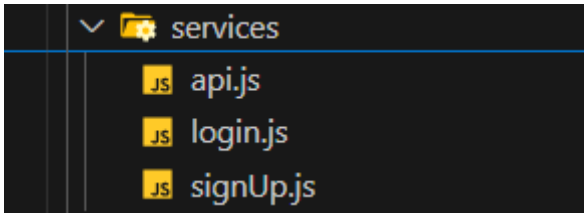

Figure 53: Frontend folder – services folder

styles folder:

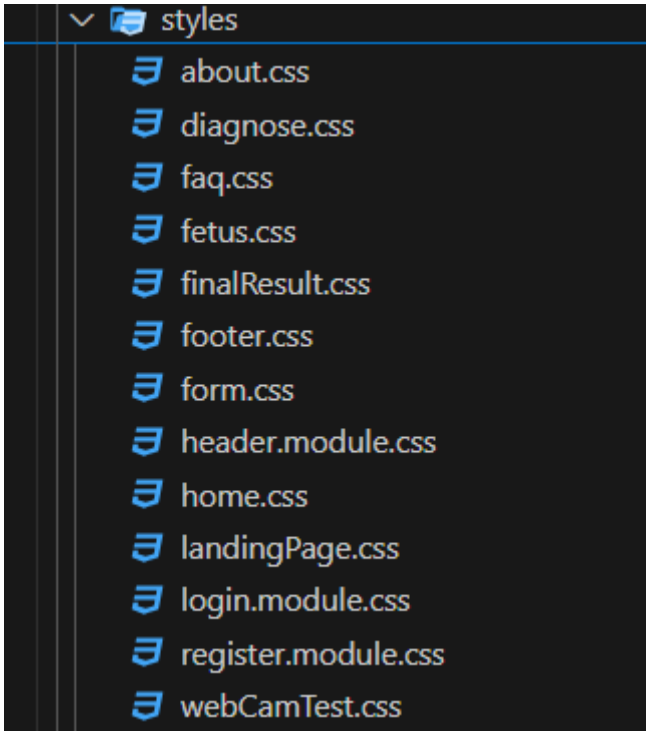

Figure 54: Frontend folder – styles folder

- **backend folder:**

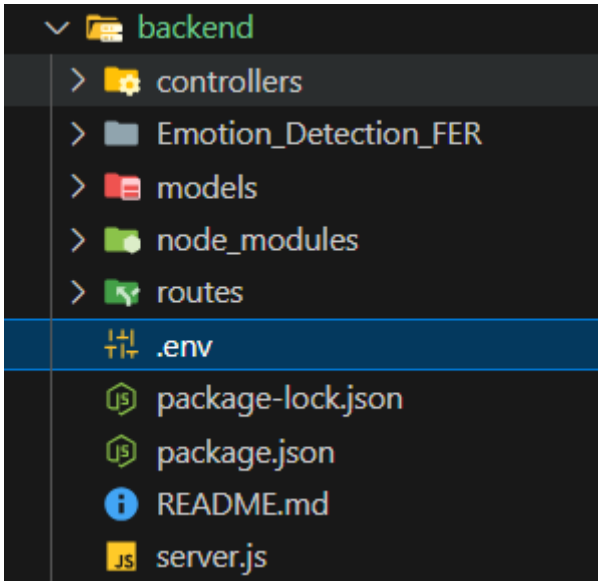

Figure 55: Backend folder

Controllers folder:

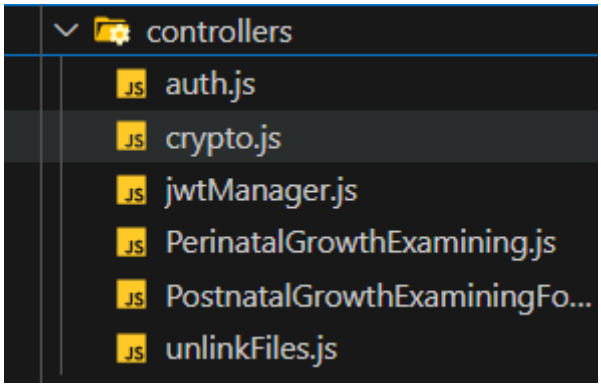

Figure 56: Backend folder – controllers folder

Emotion_Detection_FER folder:

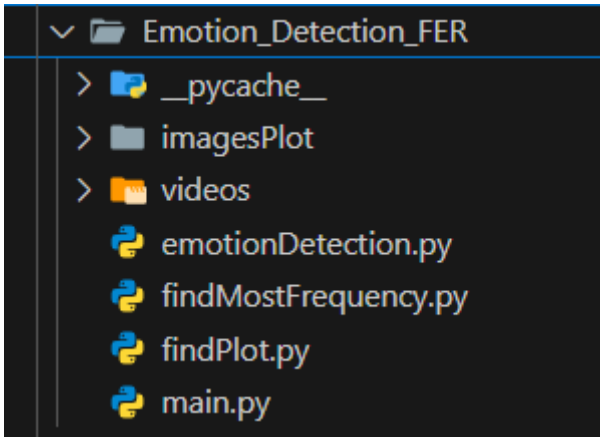

Figure 57: Backend folder – Emotion detection FER folder

Models folder:

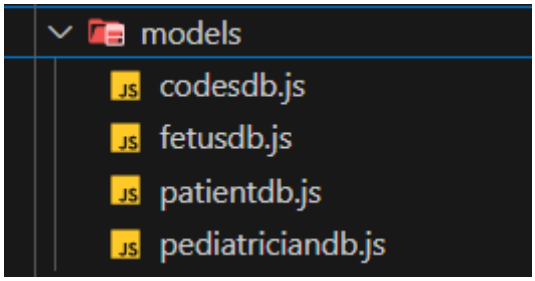

Figure 58: Backend folder – models

Routes folder:

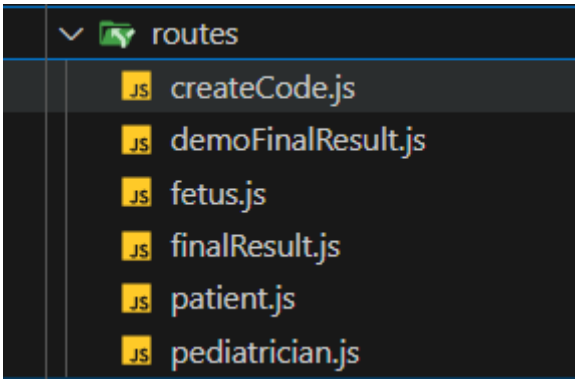

Figure 59: Backend folder – routes

18.4 Functional explanation about the backend

Now we will go through each file in the backend folder and explain the input and output of each function if they exist otherwise the purpose of each script.

Controllers/auth.js

This middleware function verifies the JWT token from the Authorization header and attaches the decoded payload to req.user if valid. If the token is invalid, it returns a 401 Unauthorized response.

Controllers/Crypto.js

encrypt(text)

- Encrypts a plain text using AES-256-CBC with a static IV and key.
- Input: A string text to be encrypted.
- Output: A hexadecimal string representing encrypted data.

decrypt(encryptedText)

- Decrypts an encrypted hexadecimal string using AES-256-CBC with a static IV and key.
- Input: A hexadecimal string encryptedText to be decrypted.
- Output: The decrypted original text (e.g., "Hello World").

Controllers/jwtManager.js

jwtManager(user)

- Generates a JWT token for the provided user containing the user's ID and username.
- Input: An object user with at least two properties: id (user's unique identifier) and username (user's name).
- Output: A JWT token string signed using the secret stored in process.env.jwt_salt.

Controllers/unlinkFiles.js

unLinkFiles(filePath)

- Deletes a file from the filesystem if it exists.
- Input: A string filePath representing the path to the file (e.g., "./path/to/file.txt").
- Output: No return value, but logs either an error or "deleted" to the console depending on whether the deletion was successful.

Controllers /PerinatalGrowthExamining.js:

This file contains all the calculations and formulas required for analyzing patient data during the fetal period. Several functions are defined and called within the main function, which is exported for external use.

function firstTrimesterPi(Mft, Lft) :

- input: the fetus mass in gram at week 16 (Mft) and the length in centimeters at week 16 (Lft).
- output: first Trimester Pi that is the input for fetalGrowthIndication.

function lastTrimesterPi(Mlt,Llt) :

- input: the fetus mass at week 32 (Mlt) and the length at week 32 (Llt).
- output: last Trimester Pi that is the input for fetalGrowthIndication.

function fetalGrowthIndication(PIft,PIlt):

- input: the two calculated values from the functions firstTrimesterPi and lastTrimesterPi
- output: the function returns the result about the fetus fetal growth indication that is shown in the final result test

function fetalBirthWeightIndication(bw):

the function checks if the patient weight indication is normal

- input: the patient weight in kg.
- output: if There were some problems and the birth weight indication is not normal the function will return "low", otherwise the function will return "ok".

main: function (id, bw, week16Mass, week16Length, week32Mass, week32Length):

The main function evaluates the fetal growth and birth weight based on input parameters collected during pregnancy (fetal mass and length at 16 and 32 weeks) and birth weight. It calculates specific fetal growth indicators (Ponderal Index, or Pi) for the first and last trimesters and uses these to assess whether any issues occurred during the pregnancy, such as mother-dependent or child-dependent factors affecting growth. Additionally, it evaluates whether the birth weight was within a normal range. The function then returns a comprehensive result summarizing these evaluations.

- input:
  - id (number/string): The ID of the patient (fetus), although this is not actively used in the calculations within this function.
  - bw (number): The birth weight of the fetus in kilograms.
  - week16Mass (number): The mass of the fetus at 16 weeks (presumably in grams or kilograms).
  - week16Length (number): The length of the fetus at 16 weeks (in centimeters or meters).
  - week32Mass (number): The mass of the fetus at 32 weeks (presumably in grams or kilograms).
  - week32Length (number): The length of the fetus at 32 in cm
- Output: finalIndicationResults (string) A detailed textual report.

Controllers /PostnatalGrowthExaminingFormulas.js :

function birthPercentileCalc(gender , bw):

calculate the birth percentile based on the child's gender and birth weight.

- Input:
  - gender (string): The gender of the child ("boy" or "girl").
  - bw (number): Birth weight of the child (in kilograms).
- Output:
  - A string representing the birth percentile (e.g., "0-0", "1-5", etc.). that is input for the function firstPeriodPercentilesCrossCalc.

function month6PercentileCalc(gender , w6):

calculate the weight percentile of the child at 6 months.

- Input:
  - gender (string): The gender of the child ("boy" or "girl").
  - w6 (number): The weight of the child at 6 months (in kilograms).
- Output: A string representing the percentile for weight at 6 months. that is input for the function firstPeriodPercentilesCrossCalc.

function month12PercentileCalc(g, w12):

calculate the weight percentile of the child at 12 months.

- Input:
  - g (string): The gender of the child ("boy" or "girl").
  - w12 (number): The weight of the child at 12 months (in kilograms).
- Output: A string representing the percentile for weight at 12 months. that is input for the function secondPeriodPercentilesCrossCalcu.

function month18PercentileCalc(g, w18):

The function calculates the weight percentile of the child at 18 months.

- Input:
    - g (string): The gender of the child ("boy" or "girl").
    - w18 (number): The weight of the child at 18 months (in kilograms).
- Output: A string representing the percentile for weight at 18 months.

function month24PercentileCalc(gender , w24):

The function A string representing the percentile for weight at 24 months

- Input:
    - gender (string): The gender of the child ("boy" or "girl").
    - w24 (number): The weight of the child at 24 months (in kilograms).
- Output: A string representing the percentile for weight at 24 months.

function month36PercentileCalc(gender , w36):

The function calculates the weight percentile of the child at 36 months.

- Input:
    - gender (string): The gender of the child ("boy" or "girl").
    - w36 (number): The weight of the child at 36 months (in kilograms).
- Output: A string representing the percentile for weight at 36 months.

function month48PercentileCalc(gender , w48):

The function calculates the weight percentile of the child at 48 months.

- Input:
    - gender (string): The gender of the child ("boy" or "girl").
    - w48 (number): The weight of the child at 48 months (in kilograms).
- Output: A string representing the percentile for weight at 48 months.

function month60PercentileCalc (gender, w60):

The function calculates the weight percentile of the child at 60 months.

- Input:
    - gender (string): The gender of the child ("boy" or "girl").
    - w60 (number): The weight of the child at 60 months (in kilograms).
- Output: A string representing the percentile for weight at 60 months.

function severity (pc1,pc2,pc3):

The function calculates the severity of weight gain deceleration based on the percentiles crossed in different periods.

- Input:
    - pc1 (number): The percentile crossing value for period 1.
    - pc2 (number): The percentile crossing value for period 2.
    - pc3 (number): The percentile crossing value for period 3.
- Output: A weighted sum of the percentiles.

function totalPercentilesCrossedCalcu(pc1, pc2, pc3):

The function calculates the total number of percentiles crossed from birth to 60 months

- Input:
    - pc1, pc2, pc3 (numbers): Percentile crossing values for periods 1, 2, and 3.
- Output: The total number of percentiles crossed across all periods.

function firstPeriodPercentilesCrossCalc(p0, p6):

The function calculates the number of percentiles crossed during the first period (birth to 6 months).

- Input:
    - p0 (string): The birth percentile.
    - p6 (string): The percentile at 6 months.

- Output: The difference in percentiles between birth and 6 months.

function lastPercentiledatagetting1(p0, p6):

The determine the final percentile and age for the first period (birth to 6 months). that is input for the function secondPeriodPercentilesCrossCalcuecond as lastAgePerPc1.

- Input:
    - p0 (string): The birth percentile.
    - p6 (string): The percentile at 6 months.
- Output: The last percentile and age for the first period.

function secondPeriodPercentilesCrossCalcuecond (lastAgePerPc1, p12, p18, p24):

The function calculates the number of percentiles crossed during the second period.

- Input:
    - lastAgePerPc1 (string): The final percentile and age from the first period.
    - p12, p18, p24 (strings): Percentiles at 12, 18, and 24 months.
- Output: The difference in percentiles for the second period (6 months to 24 months).

function lastPercentiledatagetting2(maxPc1, p12, p18, p24):

The function determines the final percentile and age for the second period (6 to 24 months). that is input for the function thirdPeriodPercentilesCrossCalcu as lastAgePerPc2.

- Input:
    - maxPc1 (string): The final percentile from period 1.
    - p12, p18, p24 (strings): Percentiles at 12, 18, and 24 months.
- Output: The final percentile and age for period 2.

function thirdPeriodPercentilesCrossCalcu(lastAgePerPc2, p36, p48, p60):

The function calculates the number of percentiles crossed during the third period.

- Input:
    - lastAgePerPc2 (string): The final percentile and age from the second period.
    - p36, p48, p60 (strings): Percentiles at 36, 48, and 60 months.
- Output: The difference in percentiles for the third period (24 to 60 months).

function lastPercentiledatagetting3(maxPc1 , p36, p48, p60):

The function determines the final percentile and age for the third period (24 to 60 months).

- Input:
    - maxPc1 (string): The final percentile from previous periods.
    - p36, p48, p60 (strings): Percentiles at 36, 48, and 60 months.
- Output: The final percentile and age for the third period.

function weightGainDecelerationAnalyzing (gender, bw, weight6M, weight12M, weight18M, weight24M, weight36M, weight48M, weight60M):

The function analyzes the deceleration in weight gain and assess the overall growth pattern of the child over time.

- Input:
    - gender (string): The gender of the child ("boy" or "girl").
    - bw (number): Birth weight.
    - weight6M, weight12M, weight18M, weight24M, weight36M, weight48M, weight60M (numbers): Weights at 6, 12, 18, 24, 36, 48, and 60 months.
- Output: An array containing the severity, total percentiles crossed, and the final percentile and age.

function postnatalGrowthEvaluating (Id, lastAgePer, totalCross, evalSeverity):

The function evaluates the child's growth based on percentile crossing and provide feedback on potential concerns related to Failure to Thrive (FTT).

- Input:

- Id (number): Child's ID.
- lastAgePer (string): The final percentile and age.
- totalCross (number): Total percentiles crossed.
- evalSeverity (number): Severity value.

- Output: A string summarizing the postnatal growth evaluation.

function main (Id, gender, bw, weight6M, weight12M, weight18M, weight24M, weight36M, weight48M, weight60M):

The function gathers input data, calculate weight gain deceleration, and provide a comprehensive evaluation of the child's postnatal growth, focusing on potential issues like Failure to Thrive (FTT).

- Input:
  - Id (number): Child's ID.
  - gender (string): Child's gender.
  - bw, weight6M, weight12M, weight18M, weight24M, weight36M, weight48M, weight60M (numbers): Child's birth weight and weights at various age intervals.
- Output: A final string summarizing whether the child suffers from FTT or is healthy.

Models/Codedb.js

codesSchema

- Defines a Mongoose schema for storing codes in the database.
- Input: Each document in the Code collection will have a code field, which is an Object.
- Output: A Mongoose model named Code that allows interaction with the MongoDB collection for storing and retrieving code objects.

Models/Fetusdb.js

fetusSchema

- Defines a Mongoose schema for storing fetal development data at specific weeks (16 and 32) in the database.
- Input: Each document will have the following fields:
    - id: An Object representing a unique identifier.
    - week16Mass: A Number representing the mass at 16 weeks.
    - week16Length: A Number representing the length at 16 weeks.
    - week32Mass: A Number representing the mass at 32 weeks.
    - week32Length: A Number representing the length at 32 weeks.
- Output: A Mongoose model named Fetus for storing and retrieving fetal development data.

Models/Patientdb.js

patientSchema

- Defines a Mongoose schema for storing patient information, including birth data, weight at different months, and other relevant medical data.
- Input: Each document in the Patient collection will have the following fields:
    - id: An Object representing a unique identifier for the patient.
    - birthDate: A Date object representing the patient's birth date.

- ageInMonth: A Number representing the patient's age in months.
- alergics: A String containing details about allergies.
- motherHeight: A Number representing the height of the mother.
- motherWeight: A Number representing the weight of the mother.
- motherAge: A Number representing the age of the mother.
- gender: A String representing the gender of the patient.
- birthWeight through Month60Weight: Number fields representing the patient's weight at birth and at intervals up to 60 months, with default values set to 0.
- result: A String storing the final diagnosis result.

- Output: A Mongoose model named Patient, used for creating and retrieving patient records.

Models/Pediatriciandb.js

PediatricianSchema

- Purpose: Defines a Mongoose schema for storing pediatrician information in the database.
- Input: Each document in the Pediatrician collection will have the following fields:
  - id: A String representing a unique identifier for the pediatrician.
  - username: A String representing the pediatrician's username.
  - password: A String representing the pediatrician's hashed password.
  - medicalClinic: A String representing the name of the medical clinic where the pediatrician works.
- Output: A Mongoose model named Pediatrician, used for storing and retrieving pediatrician records.

routes/demoFinalResult.js

This JavaScript file defines two **POST routes** for demo purposes that process **fetus growth data** and **patient growth data** using custom formula logic from external controllers (PostnatalGrowthExaminingFormulas and PerinatalGrowthExamining). The goal is to compute results based on growth data and return them to the client.

- Dependencies:
  - **express**: Manages HTTP requests and routing.
  - **main1 (PostnatalGrowthExaminingFormulas)**: A custom module that provides growth examination formulas for postnatal growth.
  - **main2 (PerinatalGrowthExamining)**: A custom module that handles growth examination for fetuses during the perinatal period.
- Routes Explanation:
  1. **POST /demoFetusResult**: Processing Fetus Growth Data
     - Input: A JSON object with fetus growth data fields like birthWeightFetus, week16Mass, week16Length, etc.
     - Process:
       - The route checks if all required fields (birthWeightFetus, week16Mass, week16Length, week32Mass, week32Length) are present in the request body. If any field is missing, it returns a 400 status with an error message.
       - If all fields are present, it calls the main2.main() function from the **PerinatalGrowthExamining** module, passing the fetus data.
       - The result from main2.main() is stored in fetusResult.
     - Output:
       - On success return a 200 status with the growth calculation results (fetusResult or patientResult).
       - On failure (missing fields or server error), the route returns appropriate error messages with either 400 or 500 status codes.
  2. **POST /demoPatientResult**: Processing Patient Growth Data.
     - Input: A JSON object with patient growth data fields like gender, birthWeight, Month6Weight, Month12Weight, etc.
     - Process:

- This route logs and checks if the necessary fields are present in the request body, particularly the weight data for various months.
- After confirming that all required fields are present, it calls the main1.main() function from **PostnatalGrowthExaminingFormulas**, passing the patient growth data.
- The result from main1.main() is stored in patientResult.

- Output
  - On success, return a 200 status with the growth calculation results (fetusResult or patientResult).
  - On failure (missing fields or server error), the route return appropriate error messages with either 400 or 500 status codes.

routes/ finalResult.js

This JavaScript file is part of an **Express.js** application and defines routes for handling requests related to **patient and fetus data processing**, including calling a Python script for additional emotion detection analysis. The main functionality revolves around retrieving data, executing analysis using custom formulas, and running a Python script for a webcam test.

- Dependencies:
  - **express**: Handles routing for HTTP requests.
  - **spawn** from **child_process**: Used to execute the Python script that handles emotion detection from a video file.
  - **Patient and Fetus Models**: Mongoose models for interacting with patient and fetus data stored in MongoDB.
  - **PostnatalGrowthExaminingFormulas (main1)**: Custom controller for postnatal growth examination formulas.
  - **PerinatalGrowthExamining (main2)**: Custom controller for perinatal growth formulas.

- **unLinkFiles**: Custom module that deletes files (likely for cleaning up the video and JSON files after processing).
- **cryptData**: Custom encryption module used to encrypt and decrypt identification data for patient and fetus records.

- Routes Explanation:

1. **GET /deleteFiles/: identification:** File Cleanup Route

    - Input: The route expects an **identification** parameter to locate and delete associated files.
    - Process: Calls the unLinkFiles controller to delete the video (.mp4) and JSON files associated with the provided ID.
    - Output:
        - On success: Returns a success message with 201 status code
        - On error: It logs any issues to the console and returns a server error response.

2. GET /finalResult/: identification: Retrieve and Process Patient Data

    - Input: The route expects an identification parameter to retrieve and process patient and fetus data.
    - Process:
        - Retrieve Data:
            - Retrieves patient and fetus data using the encrypted ID from the MongoDB collections (Patient and Fetus).
            - If the data for the patient or fetus is not found, it returns a 404 error.
        - Apply Formulas:
            - The **postnatal growth formulas** (via main1.main()) are applied to the patient's data.

        - The **fetal growth formulas** (via main2.main()) are applied to the fetus data.
    - Age-Based Emotion Detection:
        - If the patient is between **6 and 25 months old**, the **emotion detection** script is triggered by executing a Python script (Emotion_Detection_FER/main.py).
        - The Python script processes the webcam video associated with the patient's ID, and its results are parsed.
        - The emotion detection test result is used to further assess the possibility of **Failure to Thrive (FTT)** in the child.
        - Based on the result, a message (affect) is generated, indicating whether the emotion detection test affects the overall growth assessment.
    - **Save Results**:
        - The complete result (formulas + emotion detection results) is saved back into the patient's document in MongoDB.

- Output: JSON response containing:
    - Growth formulas results.
    - Fetal growth results.
    - Emotion detection results (if applicable).
    - Message indicating whether the emotion detection test affects the diagnosis.
    - Error message on server failure.

routes/ patient.js

This JavaScript file defines several Express.js routes for managing patient data. The key functionality includes creating or updating a patient's growth information (based on age in months) and retrieving patient records by their identification (which is encrypted).

- Dependencies:
    - express: Manages routing for HTTP requests.
    - bcrypt: Used to hash sensitive data (not directly used in this file, but the saltRounds variable is declared).
    - Patient: The Mongoose model for interacting with the patient collection in the MongoDB database.
    - cryptData: A custom encryption module used to encrypt and decrypt patient IDs.
- Routes Explanation:

1. POST /DiagnoseForm: Add or Update Patient Data

- Input: A JSON object containing patient data (identification, birth date, weight at different months, etc.).
- Process:
    - First, it retrieves all patient records from the MongoDB collection.
    - It decrypts the stored id field for each patient and checks if it matches the provided identification field. If a match is found, the patient's record is retrieved.
    - If the patient does not exist, a new Patient object is created using the provided data and saved to the database.
    - If the patient exists, the weight information is updated based on the patient's **age in months**. Specific fields (birthWeight, Month6Weight, etc.) are updated depending on the patient's age range.
    - Finally, the updated or new patient document is saved to the database.
- Output

- On success: Returns the updated or newly created patient document.
- On failure: Returns a server error message.

2. POST /getID: Placeholder Route

- Input: the request body contains patient data.

- **Process**: This route logs the request body to the console and responds with a success message.
- **Output**: Always returns a success message with `201` status.

3.GET /by-identification/:identification: Retrieve Patient by ID

- Input: The route expects a patient's identification in the URL as a parameter.
- Process:
  - It retrieves all patient records from the MongoDB collection.
  - It decrypts each stored id and checks if it matches the provided identification parameter. If a match is found, the patient's document is returned.

routes/createCode.js

This JavaScript file defines two routes, /createCode and /checkCode, to handle creating and checking encrypted codes in a MongoDB database. It uses **Express.js** for routing, and **Mongoose** to interact with the database, with encryption and decryption managed by a custom cryptData module.

- Dependencies:
    - express: Handles HTTP requests and routing.
    - mongoose: Used to interact with the MongoDB database where encrypted codes are stored.
    - cryptData: This is a custom module that provides two key functions:
        - encrypt(data): Encrypts the given data.
        - decrypt(data): Decrypts the given encrypted data.
- Database Model:
    - Code: Represents a MongoDB schema (probably using Mongoose) for storing the encrypted code data in the database.
- Routes Explanation:

1. **POST /createCode**

    - Input: A request containing a code in the request body (JSON format)
    - Process:
        - The code is extracted from the request body.
        - It ensures that the code is a string by casting it with String(code).
        - The code is encrypted using cryptData.encrypt(code).
        - A new Code object is created with the encrypted code.
        - The Code object is saved to the MongoDB database.
    - Output:
        - On success: Returns a 201 status code and a JSON response indicating that the code was added.
        - On error: Returns a 500 status code and an error message if something goes wrong during the save operation.

2. POST /checkCode

- Input: A request containing a code in the request body (plaintext)
- Process:
    - All stored codes are retrieved from the database using Code.find().
    - Each code from the database is decrypted using cryptData.decrypt(obj.code).
    - The decrypted code is compared with the provided plaintext code from the request.
    - If a match is found, it sends a success response. If no match is found after iterating through all stored codes, it returns an "invalid code" response.
- Output:
    - On success (if a code matches):

Copy code

{

"status": "Success",

"message": "Code found successfully!"

}

    - On failure (if no matching code is found or a decryption error occurs):

{

"message": "Invalid code."

}

routes/fetus.js

This JavaScript file defines two routes under /pediatrician/fetus, allowing the pediatrician to manage **fetal data** in a MongoDB database. The first route (POST /fetus) allows the creation of new fetal records, while the second route (GET /by-identification/:identification) retrieves a fetal record based on the **identification number**, with both the ID and the fetus data being securely encrypted.

- Dependencies:
  - express: Manages HTTP requests and routing.
  - mongoose: Used to interact with MongoDB for CRUD operations.
  - cryptData: A custom module responsible for encrypting and decrypting data such as fetal IDs.
  - Fetus: Represents the Mongoose schema for managing fetal data in the database.
- Database Model:
  - Fetus: A MongoDB model for storing encrypted fetal information, including identification, birth weight, mass, and length at specific weeks of development.
- Routes Explanation:

1. POST /fetus: Add New Fetus Data

- Input: A JSON object with fields identification, week16Mass, week16Length, week32Mass, week32Length, and birthWeightFetus.
- Process:
  - Checks if all required fields (week16Mass, week16Length, week32Mass, week32Length, identification, and birthWeightFetus) are provided.
  - Encrypts the identification field using cryptData.encrypt().
  - Checks the database to see if a fetus with the encrypted ID already exists.
  - If no existing record is found, creates a new Fetus object, encrypts the identification, and stores the new fetal data in the database.

- Output: A success message if a new fetus record is added or if it already exists. Error messages if required fields are missing or a database issue occurs.

2. **GET /by-identification/:identification:** Retrieve Fetus Data by Identification

- Input : A plaintext identification number in the URL.
- Process:
  - Retrieves all fetal records from the database using Fetus.find().
  - Iterates through the records and decrypts each id using cryptData.decrypt().
  - If a decrypted id matches the given identification, the corresponding fetal record is returned.
- Output: The decrypted fetal data if a match is found. If not, a "Fetus not found" message is returned. On error, a "Server error" message is returned.

routes/pediatrician.js:

This JavaScript file defines routes to handle **pediatrician registration** and **login** for a healthcare system. It uses **bcrypt** for hashing passwords and IDs, **JWT (JSON Web Token)** for managing user sessions, and **MongoDB** (via Mongoose) to store and retrieve pediatrician data.

- Dependencies:
  - bcrypt: A password-hashing library used to securely store passwords and sensitive data (e.g., IDs).
  - express: Manages HTTP routing and requests.
  - jwtManager: A custom module that generates a JWT (used to issue access tokens for authenticated users).
  - Pediatrician: Represents the Mongoose schema used to manage pediatrician data in the MongoDB database.
- **Routes Explanation:**

1. **POST /register**: Register a New Pediatrician

- Input: A JSON object with the fields id, username, password, and medicalClinic.

- Process:
    - First, checks if the required fields (id, username, password, and medicalClinic) are present. If any are missing, it returns a 400 status.
    - Converts the username to lowercase to ensure case-insensitive handling.
    - Checks the database to see if the username is already in use.
    - If the username is available:
        - **Hashes the password** and **ID** using bcrypt.hash() with a salt round value provided in the environment variables (process.env.SALT_ROUNDS).
        - Creates a new Pediatrician object with the hashed values and stores it in the MongoDB database.
        - Generates a **JWT access token** using jwtManager(newPediatrician).
        - Saves the new pediatrician in the database.
- Output: A success message if registration is successful, along with an access token and the user's details. An error message if the username is already in use or a required field is missing.

2. **POST /login: Login for an Existing Pediatrician:**
    - Input : A JSON object with the fields id, username, password, and medicalClinic.
    - Process:
        - Converts the username to lowercase to ensure case-insensitive lookups.
        - Looks up the pediatrician by username in the MongoDB database.
        - If the pediatrician is found, compares the **plaintext password** provided in the request with the **hashed password** stored in the database using bcrypt.compare().

- If the password comparison is successful, a **JWT access token** is generated and returned to the user.

- Output: A success message if login is successful, along with an **access token**. An error message if the username or password is invalid or if there is a server issue.

18.5 Python scripts explanation

emotionDetection.py

The purpose of this Python file is to perform **emotion detection** on video frames and track the **movements of the upper and lower parts of the face** using both the FER library for emotion recognition and Mediapipe for face landmark detection. The results are saved in a JSON file, and the code identifies which part of the face (upper or lower) shows more frequent movements.

- Libraries used:
    - FER (Facial Emotion Recognition): The FER library detects emotions such as happiness, sadness, anger, etc., from each frame of the input video.
    - Mediapipe Face Mesh: Mediapipe's FaceMesh detects and tracks **landmarks** on the face. In this script, the face mesh tracks specific **upper face** (eyes, eyebrows, forehead) and **lower face** (mouth, jaw) movements.
- Code Workflow Explanation:
    - Input: ID A string representing the unique identifier for the video file (e.g., "001111111"). The script will look for a video file in the directory "./Emotion_Detection_FER/videos/".
    - Emotion Detection:
        - FER is used to analyze each frame of the video for emotional expressions.
        - The detected emotions are stored in a **list** of dictionaries (emotion_results), where each dictionary contains emotional scores for that particular frame.
        - These results are saved in a **JSON file** located at "./Emotion_Detection_FER/" + ID + ".json".
    - Face Movement Detection:

        - The **upper face movements** (eyes, eyebrows and forehead) and **lower face movements** (mouth, jaw, chin) are tracked using **Mediapipe's FaceMesh**.
        - Movements are compared between consecutive frames, and the system counts how frequently movements exceed a defined threshold.
        - The script prints which face region (**upper** or **lower**) had more frequent movements during the video analysis.
    - Threshold for Movements:
        - A threshold of movement is set at 5 pixels, meaning if the face landmarks move more than 5 pixels between frames, the movement is counted.
    - Output:
        - A **JSON file** with detected emotions for each frame, located at './Emotion_Detection_FER/' + ID + '.json'.
        - A **message** indicating whether the **upper** or **lower face** had more frequent movements.

findMostFrequency.py:

The purpose of this Python script is to analyze emotion detection results stored in a JSON file and identify the **most frequent emotion** across all frames. It reads a JSON file generated from a prior emotion detection process, counts the occurrences of each emotion, and prints the most frequent one.

- Libraries used:
    - json: The json library is used to parse and manipulate JSON data in Python. In this script, it reads the JSON file generated by the emotion detection process and loads it into a Python dictionary for further analysis.
    - collections.Counter: Counter is a subclass of the dict type from the collections module that helps in counting hashable objects. In this script, it is used to count the occurrences of each detected emotion across the video frames.

- **sys**: The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. While not directly used in the provided script, it is imported to allow for further enhancement (such as handling command-line arguments or exceptions).

- Code Workflow Explanation:
    - Input: ID A string that represents the unique identifier for the JSON file (e.g., "001111111"). The script will read the file from './Emotion_Detection_FER/' + ID + '.json'.
    - Emotion Data:
        - The JSON file contains an array where each element represents a frame's emotion analysis.
        - Each frame's emotion data includes a dictionary of emotions (e.g., happiness, sadness, anger) with corresponding scores that represent the intensity of each emotion in that frame.
    - Emotion Counting:
        - The script uses Python's Counter class from the collections module to count the occurrences of each emotion across all frames.
        - The update() method of Counter is used to accumulate emotion scores from each frame's emotions dictionary.
    - Finding the Most Frequent Emotion: After counting the occurrences of each emotion, the script uses the max() function to determine the **emotion with the highest count**.
    - Output: The **most frequent emotion**.

main.py

the main script integrate different emotion detection functionalities, allowing the developer to detect emotions from a video, find the most frequent emotion. This script serves as a **main entry point** for running the entire process by calling other modules/functions (emotionDetection, findMostFrequency).

- Modules Used:
    - emotionDetection: This module is responsible for analyzing a video and detecting emotions frame by frame.
    - findMostFrequency: This module takes the results of the emotion detection and identifies the most frequent emotion.
- Code Workflow Explanation:
    - Input: The input to the script is an **ID**, which is passed as a command-line argument. This **ID** corresponds to the video and associated JSON files that store emotion data.
    - Emotion Detection:
        - The function emotionDetection.emotionDetection(id) is called first. It analyzes the video file associated with the given ID and performs emotion detection. The results are saved in a JSON file.
    - Find Most Frequent Emotion:
        - After the emotion detection is complete, the function findMostFrequency.findMostFrequency(id) is invoked. It reads the JSON file created by the emotionDetection module, counts the occurrences of each emotion, and identifies the most frequent one.
    - Command-Line Arguments:
        - **sys.argv** is used to capture the **ID** from the command line.
        - If an ID is provided, the script runs the main() function, passing the ID.
        - If no ID is provided, the script will print: "No ID provided" and exit.
    - Output:
        - The script produces several outputs based on the functions called:

1. Emotion Detection Results: A JSON file containing emotion data is generated.
2. Console Output: The most frequent emotion is printed to the console.

Server.js

This JavaScript file sets up an **Express.js server** that interacts with various routes for **handling pediatric, patient, and fetus data**, **uploading video files**, and **interacting with OpenAI's GPT API** to provide a **pediatrician assistant** function. It integrates **MongoDB** for data storage, **multer** for handling file uploads, and **axios** to send requests to external APIs.

- Dependencies:
    - axios: Used to make HTTP requests to the **OpenAI API** for generating responses based on user input (chat functionality).
    - express: Web framework for handling routing, middleware, and HTTP requests/responses.
    - child_process (spawn): Allows the server to spawn a new process, which can be used to run external Python scripts (although not directly used in this version).
    - multer: A middleware for handling multipart/form-data, primarily used for **file uploads** (in this case, video files).
    - cors: Middleware to allow cross-origin requests.
    - body-parser: Parses incoming request bodies in a middleware before your handlers, available under req.body.
    - mongoose: MongoDB object modeling library for Node.js, used for connecting to and interacting with a MongoDB database.
    - dotenv: Loads environment variables from a .env file into process.env (used for storing sensitive information like API keys and database URLs).
    - fs: File system module used to handle file operations such as checking if directories exist or creating them.
- MongoDB Connection:
    - The MongoDB database connection is initialized using mongoose.connect(). The URL for the connection is fetched from the .env file using process.env.mongodb_url.
    - If the connection is successful, a message is logged to the console; otherwise, errors are caught and logged.

- Routers:
    - Several routers are imported to handle different domains of the application:
        - pediatricianRouter: Manages pediatrician-related routes.
        - patientRouter: Manages patient-related routes.
        - fetustRouter: Manages fetus-related routes.
        - finalResults: Handles the final result route.
        - demoFinalResult: Demo final result route.
        - creatingCode: Handles creating code-related routes.
    - Each router is mounted at a specific route, e.g., /pediatrician, /patient, /fetus, etc.
- File Upload Handling:
    - Multer is used to handle video uploads from the client. The storage configuration ensures that uploaded files are stored in the directory ./Emotion_Detection_FER/videos.
    - The route /upload accepts a POST request with a file (video) and saves it in the specified directory. After successful upload, it returns the path to the uploaded file.

- Interacting with OpenAI API:
    - The /chat route interacts with OpenAI's GPT API. It accepts a prompt from the client, which is used to generate a completion.
    - If no prompt is provided, the API returns a 400 error.
    - If the prompt is valid, an **axios** request is sent to the OpenAI API, and the result (GPT-generated response) is returned to the client.
- Environment Variables:
    - The server uses the dotenv package to load environment variables, including sensitive information like OPENAI_API_KEY and mongodb_url. This ensures security by keeping secrets outside of the codebase.

- Server Initialization:
    - The server listens on port **5001**, which is specified as a constant at the top of the file. When the server starts, it logs the URL (http://localhost:5001) to the console.
- Input:
    - The application accepts JSON requests for various routes.
    - For /upload, it accepts a file (video) and stores it on the server.
    - For /chat, it accepts a prompt string that is sent to the OpenAI API.
- Output:
    - For /upload, the server returns the file path of the uploaded video.
    - For /chat, the server returns the AI-generated response from OpenAI.

18.6 Frontend explanation

The frontend of this project is built using React and focuses on providing an intuitive interface for pediatricians to log in, perform diagnostics on patients, and receive detailed results. The application handles various components, including user authentication, patient form submissions, webcam video recording for emotion detection, and dynamic result display. It leverages axios for API requests to communicate with the backend, sending and retrieving patient data, video files, and AI-generated nutrition plans.

The application also utilizes React Router for client-side routing, allowing users to navigate through different parts of the application such as the login page, diagnosis form, and result page. LocalStorage and React Context are used to persist and share data across components (e.g., patient identification). Additionally, multer handles file uploads for video recordings in real-time, and the OpenAI API is integrated to provide customized nutrition plans based on diagnosis results. Overall, the frontend offers a structured and interactive flow that enables pediatricians to perform and document patient diagnostics efficiently.

App.js:

This is a **React application** that uses **React Router** for client-side routing. The application consists of multiple components and pages, some of which are protected by **Private Routes** that require user authentication. Here's a breakdown of the structure and functionality:

Components and Pages:

1. WebcamVideo:
    - A component that probably utilizes a webcam for some functionality, like recording video or taking pictures.
    - It's wrapped in a PrivateRoute, meaning the user must be authenticated to access it.
2. Login:
    - The login page where users can enter their credentials to authenticate themselves.
3. Home:
    - The home page that is only accessible to authenticated users via PrivateRoute.
4. SignUp:
    - A registration page where new users can sign up.
    - It's wrapped in a RegisterPrivateRoute, which likely adds some protection or constraints on the registration process.
5. PatientForm:
    - A form where patient data is entered. This page is also wrapped in a PrivateRoute, ensuring that only authenticated users can access it.
6. LandingPage:
    - The landing page (homepage) that visitors see when they first navigate to the app (at the root /).
7. FetusPage:
    - This component is probably used for entering or viewing data about fetal development. It's protected by a PrivateRoute and requires user authentication.

8. About:
    - The about page that contains information about the app.
9. Footer:
    - The footer component that is always present at the bottom of the page.
10. PatientID:
    - Likely used for diagnosing a patient, also protected by PrivateRoute.
11. FAQ:
    - Frequently Asked Questions page for user assistance.
12. IdentificationProvider:
    - Context provider used in several routes to pass down identification-related data to the child components.
13. FinalResultTest:
    - A page that shows the final results of some tests. Protected by a PrivateRoute.
14. Demo Components:
    - DemoDiagnose, DemoPatientForm, DemoFinalResult: Components used to demonstrate the app's functionality in a test/demo mode. These components do not appear to be protected by PrivateRoute.
15. InsertCode:
    - A component used for entering some code, possibly for authentication or verification purposes.
16. Guide:
    - A page that likely provides guidance to users on how to use the app.

Key Functionality:

1. Router and Routing:
    - The app uses **React Router** (BrowserRouter, Routes, Route) to define multiple routes for different pages.
    - Each route points to a specific component that will be rendered when the user navigates to that path.

2. PrivateRoute:
    - Some routes, such as /home, /diagnose, and /fetus, are wrapped in PrivateRoute components, meaning that users must be authenticated to access these pages. The authentication check likely happens inside the PrivateRoute component.
    - If the user is not authenticated, they may be redirected to the login page.
3. RegisterPrivateRoute:
    - This is used to protect the registration page, ensuring that only authorized or specific users can register.
4. IdentificationProvider:
    - This context provider is used in various routes (like /diagnose, /fetus, and /PatientForm). It likely supplies identification data (e.g., patient ID or user ID) that is needed by the components rendered in those routes.
5. Footer:
    - The footer is a persistent part of the UI, displayed on every page below the routes.

19. References

[1].Abramson, L. (1991). Facial expressivity in failure to thrive and normal infants: Implications for their capacity to engage in the world. *Merrill-Palmer Quarterly (1982-)*, 159-182.

https://www.jstor.org/stable/23087342

[2]. Byrd, R. S., Hoekelman, R. A., & Auinger, P. (1999). Adherence to AAP guidelines for well-child care under managed care. *Pediatrics*, *104*(3), 536-540. http://pediatrics.aappublications.org/content/104/3/536

[3]. Elice, C. E., & Fields, H. W. (1990). Failure to thrive: review of the literature, case reports, and implications for dental treatment. *Pediatr Dent*, *12*(3), 185-189. https://pubmed.ncbi.nlm.nih.gov/2150222/

[4]. Homan, G. J. (2016). Failure to thrive: a practical guide. *American family physician*, *94*(4), 295-299. https://pubmed.ncbi.nlm.nih.gov/27548594/

[5]. Kuczmarski, R. J. (2002). *2000 CDC growth charts for the United States: methods and development* (No. 246). Department of Health and Human Services, Centers for Disease Control and Prevention, National Center for Health Statistics.

https://www.cdc.gov/growthcharts/background.htm

[6]. Levi-Soskin, N., Shaoul, R., Kohen, H., Jbara, A., & Dori, D. (2019). Model-based diagnosis with FTTell: Assessing the potential for pediatric failure to thrive (FTT) during the perinatal stage. In *Information Systems: Research, Development, Applications, Education: 12th SIGSAND/PLAIS EuroSymposium 2019, Gdansk, Poland, September 19, 2019, Proceedings 12* (pp. 37-47). Springer International Publishing. https://incose.onlinelibrary.wiley.com/doi/10.1002/sys.21674?af=R

[7]. Levi-Soskin, N., Yasin, F., Dori, D., & Shaoul, R. (2023). Model-based diagnosis with FTTell: Diagnosing early pediatric failure to thrive. *Systems Engineering*.

https://incose.onlinelibrary.wiley.com/doi/10.1002/sys.21674?af=R

[8]. Rowan-Legg, A., Bayoumi, I., Kwok, B., Leduc, D., Rourke, L. L., Rourke, J., & Li, P. (2021). The 2020 Rourke Baby Record release: a time for reflection and looking forward. *Paediatrics & Child Health*, *26*(5), 283-286. https://pubmed.ncbi.nlm.nih.gov/34336056/

[9]. Schwartz, I. D. (2000). Failure to thrive: an old nemesis in the new millennium. *Pediatrics in review*, *21*(8), 257-264. https://publications.aap.org/pediatricsinreview/article-abstract/21/8/257/61629/Failure-To-Thrive-An-Old-Nemesis-in-the-New?redirectedFrom=fulltext

[10]. GitHub Link: https://github.com/EhsanSarboukh/FTTell-System.git