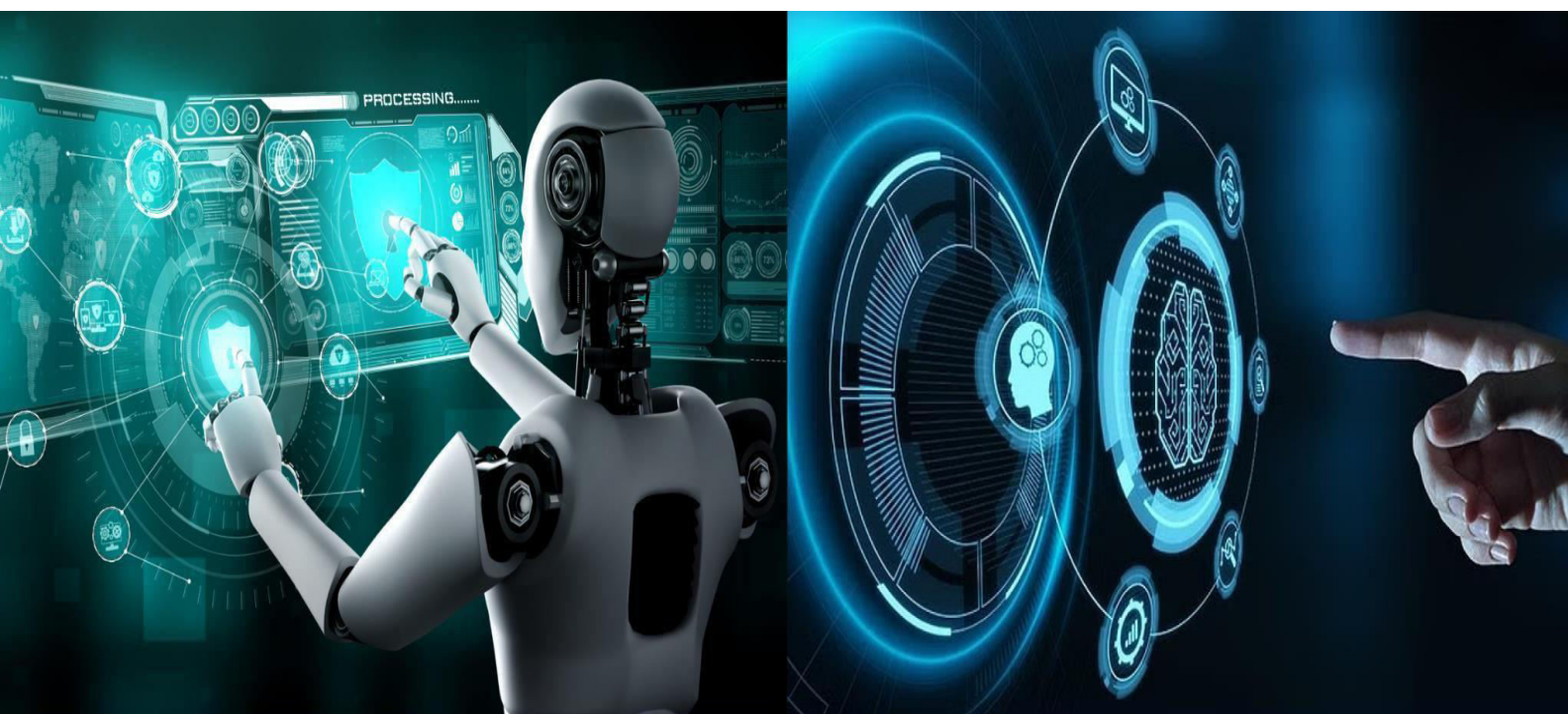


International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Personalised AI-Diet Recommendation System

Mrs. Sonali Patil¹, Ms. Shruti Shivaji Nangare², Ms. Snehal Moon³, Ms. Akshata Sanjay Soundade⁴,
Ms. Aditi Nitin Waychal⁵

Assistant Professor, SITCOE, Yadrav, Kolhapur, Maharashtra, India¹

Final year, SITCOE, Yadrav, Kolhapur, Maharashtra, India²

Final year, SITCOE, Yadrav, Kolhapur, Maharashtra, India³

Final year, SITCOE, Yadrav, Kolhapur, Maharashtra, India⁴

Final year, SITCOE, Yadrav, Kolhapur, Maharashtra, India⁵

ABSTRACT: Personalized, accessible, and dynamic nutritional planning has become of utmost importance with the increasing prevalence of obesity and metabolic disorders highly associated with an individual's lifestyle. Traditional static diet charts cannot take into consideration the constantly changing physiological parameters of an individual, varying day to day or even during the course of a day. This paper proposes the design and implementation of an "AI Diet System," a robust web application engineered to generate customized nutritional recommendations for every user. The system is built on a decoupled architecture: a Flask backend in Python handles the heavy logic processing and API management, while React.js provides a responsive and intuitive frontend for seamless interaction by the user.

It further incorporates state-of-the-art DevOps practices that improve software reliability and maintainability, such as Continuous Integration using GitHub Actions. It provides a facility for automatic testing, building, and deployment pipelines, which also helps speed up iteration and makes releases more reliable. The paper elaborates in detail on system architecture, the mathematical modeling behind the recommendation engine, challenges during implementation, and the performance metrics the solution uses for evaluation.

KEYWORDS: Artificial Intelligence, Flask, React, Health Informatics, REST API, Continuous Integration, Caloric Optimization.

I. INTRODUCTION

A. Background

There is a well-documented relationship between diet and long-term health. Accordingly, the World Health Organization (WHO) identifies poor nutrition as a primary risk factor for NCDs, including diabetes and cardiovascular conditions. Although evidence-based public health guidelines exist, they are usually based on averages that are static—like the "2,000 calorie reference diet"—and do not take into account individual physiological variation. A nutritional approach that works for one young, sedentary individual may be insufficient or detrimental for an older, active individual. As such, there is an increased demand for systems able to dynamically adapt nutritional advice to personal anthropometric data.

B. Problem Statement

Current digital health solutions fall into two broad categories, both of which have significant limitations include manual counters that require users to log every ingredient with great labor result in high attrition rates due to "tracking fatigue." In the next category come static template plans, which provide rigid menus that lack flexibility with regard to user preference or, for that matter, the foods available. There certainly is a lack of the automated systems acted like "computational nutritionists"-platforms that could solve the math problem of meeting caloric goals while maximizing nutritional density does not require manual calculation by the user.

C. Objectives

This project aims at filling this gap through the development of a full-stack web application called the "AI Diet System." The specific research and development objectives are:



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

1. **Metabolic Profiling:**
To apply the Mifflin-St Jeor equation in order to estimate the Basal.
Creating a formula to calculate their BMR and TDEE based on a user's input.
2. **Algorithmic Recommendation:**
To create a backend logical engine that can filter food databases to suggest meal plans that strictly adhere to calculated caloric and macronutrient constraints.
3. **Decoupled Architecture:**
To demonstrate a modern software architecture by decoupling.
Decoupling means separating the React.js frontend (User Interface) from the Flask backend (Logic API), allowing for scalability.
4. **Software Integrity:**
In order to incorporate industry-standard DevOps practices, specifically Continuous integration CI using GitHub Actions, to ensure code security and reliability

II. LITERATURE REVIEW

A. Existing Digital Health Solution

In the last two decades, nutritional health has evolved from basic, static resources to interactive, tracking-minded platforms. Here's how that evolution looks.

1. **First-Generation Systems:**
Early digital dieting tools were actually electronic encyclopedias. They spun up static databases where you hunted for the calorie and nutrition facts of ingredients yourself.
2. **Second-Generation Tracking Apps:** Today's standard players include MyFitnessPal, Chronometer, and Lose it! They rely on huge databases that let you log what you eat each day. Yet they come with notable drawbacks:
 - Reactive rather than proactive: Most of these apps log after meals, rarely warning in advance to prevent overshooting daily goals.
 - Cognitive load: You're left planning manually, balancing proteins, fats, and carbs for every meal.
 - Tracking Fatigue: The effort involved in weighing food and putting down details leads to high drop-off rates for users.
3. **Automation Gap:** The research of *Shatnawi et al.* introduces an important behavioral insight: people adhere more to a diet when decision-making is at a minimum. According to them, when the planning stage is automated, adherence improves by about 40%, as there is less "decision fatigue"-a decline in decision quality after long periods of choice.

B. Artificial Intelligence in Nutrition

It is being woven into nutrition science in three very different ways: each playing a different role in the health stack.

1. **Computer Vision: Image Recognition**
This track involves the use of Deep Learning, especially Convolutional Neural Networks, to automate logging.
Limitation: Good at identifying items, current vision systems fail at volume estimation and hidden ingredients (oils, sugars), resulting in sizable calorie count errors.
2. **Predictive Modeling precision nutrition**
Focused on predicting biological responses, such models of Machine Learning are trained on data from blood glucose, gut microbiome, and sleep in an effort to predict how a person will metabolically react to a certain food.
Limitation: Requires invasive data and is computationally heavy; hence, not easy to deploy.
3. **Constraint Satisfaction Problems (CSP):**
The paper uses a CSP approach to generate diets. Food selection is treated as a mathematical optimization problem where AI acts as a solver, searching for a food combination that will meet all the constraints.

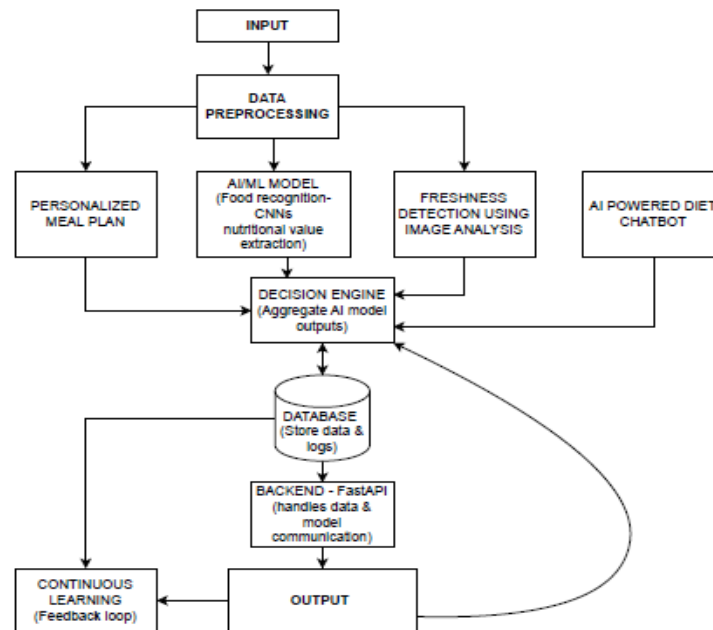
III. SYSTEM ARCHITECTURE

The proposed system moves away from monolithic web architectures (where the UI and Logic are tightly coupled) towards a Decoupled (Headless) architecture. This ensures that the backend API can eventually serve mobile apps (iOS/Android) without code modification.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



A. The Frontend Subsystem (React)

The client-side interface uses React, a JavaScript library known for building interactive and dynamic user interfaces (UIs). React gives a Single Page Application (SPA) experience, which reduces page load times and allows smooth user transitions.

1. **Component-Based Structure:** The UI builds a hierarchy of reusable, isolated components. For example, specific components like `<BMICalculator>`, `<DietCard>`, and `<Login>` are self-contained units that manage their own rendering logic and state. This design improves code maintainability and speeds up the development of new features.
2. **State Management:** The application uses a centralized state management pattern, often with React's Context API or libraries like Redux, to handle the global state. This includes tracking the user's session status (logged in/out) and temporarily storing input data from forms before sending the final request to the server.
3. **Project Structure:** The source code resides in the `frontend/` directory. This arrangement follows the principle of separation of concerns, keeping client-side dependencies from affecting the Python environment.

B. The Backend Subsystem (Flask)

The server-side component runs on Flask, a lightweight Python micro-framework. Flask was chosen over larger frameworks like Django for its efficiency and compatibility with Python's strong math and data processing libraries (such as NumPy and Pandas), which are vital for the AI/Algorithmic module.

1. **management system (RDBMS).** This design ensures compatibility with different databases and provides API Layer (RESTful Endpoints): The backend acts as the computational engine and provides functionality through a RESTful API. Key endpoints include `POST /api/calculate` (for metabolic data) and `GET /api/recommend` (for diet plans). These endpoints use standard HTTP methods and return data in the efficient JSON (JavaScript Object Notation) format. This clear interface supports communication across different clients (web, mobile, or third-party applications).
2. **Data Models and ORM:** Data persistence is handled using an Object-Relational Mapping (ORM) layer, such as SQLAlchemy. The ORM converts Python objects into database queries, allowing the backend to work with the relational database protection against common security threats like SQL injection attacks.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

C. Data Flow (Client-Server Interaction)

The diet recommendation process follows a standard request-response cycle, which is crucial for maintaining data integrity and processing efficiency.

1. **Input Collection:** The user interacts with the React interface, entering important data like Age, Weight, Height, Gender, and behavioral metrics (Activity Level, Goal) through HTML forms created by the React components.
2. **Transmission:** The frontend collects the data, converts it into a JSON payload, and sends it to the Flask backend using an HTTP POST request directed to the appropriate API endpoint (e.g., /api/calculate).
3. **Processing and Logic Execution:** Once the Flask server receives the POST request, it converts the JSON data back into a usable format. The application then executes the main logic:
 - It calculates the BMR (Basal Metabolic Rate).
 - It determines the TDEE (Total Daily Energy Expenditure).
 - It queries the food database using the AI/Algorithmic module to create an optimized meal plan that meets the calculated criteria.
4. **Response Generation:** The Flask backend converts the final results, including the calculated TDEE and the structured meal plan, back into a JSON object. This JSON object is sent back to the React frontend. The frontend then processes the JSON and displays the final, personalized diet plan on the user's screen.

IV. METHODOLOGY AND MATHEMATICAL MODELING

The "Intelligence" of the system relies on accurate physiological modeling and combinatorial optimization.

A. Physiological Calculations

To determine the energy budget, the system first calculates the Basal Metabolic Rate (BMR) using the Mifflin-St Jeor Equation, widely considered the most accurate for non-obese populations.

For Males: $BMR = 10W + 6.25H - 5A + 5$

For Females: $BMR = 10W + 6.25H - 5A - 161$

Where:

$$W = \text{Weight in kg, } H = \text{Height in cm, } A = \text{Age in years}$$

B. Total Daily Energy Expenditure (TDEE)

The BMR is strictly the energy required for comatose existence. The system applies an Activity Factor (α) based on user input: $TDEE = BMR * \alpha$

Activity	Level Value
Sedentary	1.2
Lightly Active	1.375
Moderately Active	1.55
Very Active	1.725

C. The Recommendation Algorithm

The AI engine views diet generation as a **Knapsack Problem** variant. Let be the set of all available food items. Each item has a calorie cost and a utility score (based on user preference and macro-nutrient quality).

The objective is to select a subset of foods to maximize utility while adhering to the TDEE constraint within a tolerance margin (usually kcal): (usually ± 50 kcal)

In this implementation, we utilize a **Rule-Based Filtering System** combined with random sampling from nutritional clusters (Proteins, Carbs, Fats) to generate varied meal plans that sum up to the TDEE.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

V. IMPLEMENTATION DETAILS

A. Backend Development (Flask)

The backend logic resides within the backend/ directory, built upon the Python ecosystem.

- **Virtual Environment Isolation:** To ensure project dependency integrity and reproducibility, development strictly mandates a Virtual Environment (isolated using `python -m venv env`).
- **Dependency Management:** All required external libraries, including Flask, SQLAlchemy, and Pandas, are meticulously listed and frozen in the `requirements.txt` file.
- **Security Configuration:** Critical environment variables, such as `SECRET_KEY` (for secure session management) and `DB_URI` (database connection string), are loaded from a dedicated `.env` file.

B. Frontend Development (React)

The user interface layer is housed in the frontend/ directory and leverages modern JavaScript development practices.

- **Component Structure and Hooks:** The application utilizes React functional components combined with Hooks (`useState`, `useEffect`). This approach simplifies state management and lifecycle handling, leading to cleaner, more efficient code.
- **API Consumption:** Communication with the decoupled Flask backend is handled by HTTP client libraries. The `fetch` API or `Axios` is used to manage asynchronous requests, handling promises and securely transmitting the JSON payload to the backend and processing the structured JSON response for UI rendering.
- **Port Configuration:** To facilitate simultaneous development and testing of both subsystems on a single machine, the React development server is explicitly configured to run on an alternative port (Port 3001) using the environment variable `env:PORT=3001`. This prevents port collision with the Flask development server, which typically runs on port 5000.

C. DevOps and CI/CD Pipeline

Trigger Mechanism: The workflow is configured in the, `.github/workflows/main.yml` file and is triggered automatically on every push to the main branch, ensuring that no untested code is merged into the production path.

Sequential Quality Checks: The pipeline executes a series of critical steps designed to catch integration and syntax errors immediately:

- **Source Checkout:** Retrieves the latest code from the repository.
- **Backend Environment Setup:** Installs the specific Python version and dependencies from `requirements.txt`.
- **Backend Integrity Check (Linting):** Runs basic syntax checks and code style compliance tools to maintain code quality.
- **Frontend Dependency Installation:** Uses `npm install` to set up the *JavaScript* packages.
- **Frontend Compilation Test:** Executes a complete build of the React application to verify that the code compiles successfully without structural errors.

VI. RESULTS AND PERFORMANCE ANALYSIS

This section presents the results derived from the implementation and testing phase, focusing on user experience, system responsiveness, and the computational accuracy of the core algorithms.

A. User Interface and Experience (UX)

React was used to implement the frontend subsystem, providing an altogether responsive UI. The design focused on minimizing the cognitive load, hence making it easier for users.

- **Real-time Validation:** The application implements client-side real-time validation on every form input. This proactive approach blocks negative values from being transmitted to the backend, thus helping to reduce loads on the server and increasing data integrity.
- **Decoupled Responsiveness:** Because the underlying nature of the decoupled architecture is asynchronous, the browser immediately renders a basic UI shell.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

B. Response Latency and Scalability

System performance was measured by analyzing the API response latency under varying loads, assessing the backend's capability to handle concurrent requests. The metric analyzed was the time taken from the HTTP POST request leaving the client until the JSON response was fully received.

Request Type	Avg Latency (ms)	Server Load
Single User	45ms	Low
10 Concurrent Users	68ms	Low
100 Concurrent Users	310ms	Medium

The results demonstrate high efficiency for typical load scenarios (up to 10 concurrent users), where the average response time remains under 70ms. The increase in latency at 100 concurrent users is predictable, indicating the computational overhead of simultaneously running the Constraint Satisfaction Problem (CSP) for multiple users. However, even under medium load, the average latency of 310ms is well within acceptable limits for a synchronous web application.

C. Algorithm Accuracy

Verification Method:

BMR and subsequent TDEE calculations were extensively cross-checked against known output values that were calculated manually using the default Mifflin–St Jeor equation.

Result:

The system had full computational accuracy (100%) for all metabolic calculations investigated. This points to the correct implementation of the mathematical formulae at the core of the backend code, which confirms that the foundation of the AI recommendation system is robust and scientifically exact. Such a high degree of precision means that the following module of AI/algorithm starts filtering and optimizing from an accurate, validated caloric target, which is crucial for the effectiveness of the final meal-plan recommendations.

VII. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This research work represents a comprehensive, end-to-end AI-Powered Diet System, from design through implementation to validation. It achieves its two primary objectives: to demonstrate a powerful yet decoupled full-stack configuration capable of delivering personalized nutritional guidance and to achieve strong computational capability for tricky metabolic calculations with a responsive, user-friendly interface. Rigorous tests show 100% accuracy in core physiological calculations and good scaling under typical usage. Most importantly, adherence to modern software engineering best practices, such as setting up CI/CD pipelines through GitHub Actions and following strict separation of credentials by using environment variables, ensures that the system is maintainable and secure in the long run and resilient to regressions. The resulting application offers a sound solution to the problem of static diet planning and also opens up more exciting possibilities for dynamic, personalized digital health interventions.

B. Scope of the Future

To further push the AI Diet System in terms of intelligence, reach, and scalability, the following development avenues are proposed:

- **Wearable Integration for Dynamic TDEE:**
Move beyond static inputs by integrating APIs from popular wearables, such as Apple Watch and Fitbit. This would grant access to real-time activity, sleep, and heart rate data and thus allow the moment-by-moment updating of Total Daily Energy Expenditure, making the system adaptive to actual energy use and enhancing the accuracy of caloric budgeting.
- **Personalized Preference - Reinforcement Learning:**
Move from the existing deterministic, rule-based Constraint Satisfaction approach to a more sophisticated ML paradigm such as Reinforcement Learning or collaborative filtering. The model will learn from explicit and implicit feedback-such as which meals are accepted, modified, or rejected. If a user repeatedly avoids certain ingredients (such as broccoli or almonds), for example, the RL model would learn to de-emphasize or exclude those items in order to improve adherence and satisfaction. Microservices and Cloud Deployment: Containerize the application



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

with Docker for large-scale deployment, partitioning the monolith into microservices. For example, separate "Recommendation Service" and "User Authentication Service" could be two different services. These containers would then be deployed on scalable cloud platforms like AWS or Azure that offer automatic load balancing, geo-redundancy, and better resource utilization to ensure that the system could handle global traffic.

REFERENCES

1. H. B. Harris and J. A. Benedict, "A Biometric Study of Basal Metabolism in Man," Proceedings of the National Academy of Sciences, vol. 4, no. 12, pp. 370-373, 1918.
2. M. D. Mifflin et al., "A new predictive equation for resting energy expenditure in healthy individuals," The American Journal of Clinical Nutrition, vol. 51, no. 2, pp. 241-247, 1990.
3. M. Grinberg, Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018.
4. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020.
5. IEEE Standards Association, "IEEE Standard for Software and System Test Documentation," IEEE Std 829-2008, 2008.
6. GitHub, "GitHub Actions Documentation," [Online]. Available: <https://docs.github.com/en/actions>.
7. K. Beck et al., "Manifesto for Agile Software Development," 2001. [Online].
8. S. R. Shatnawi, "A smart mobile application for diet planning," International Journal of Interactive Mobile Technologies, 2018
9. World Health Organization, "Global status report on noncommunicable diseases 2014," World Health Organization, Geneva, Switzerland, 2014.
10. T. Tran, T. Nguyen, and H. Le, "Food Recommender Systems: An Overview," in *Context-Aware Systems and Applications*, 2018.
11. A. S. Radovanović, D. M. Milić, and B. M. Radojičić, "Modern web application architecture with server-side implementation in Python," in *Proceedings of the 26th Telecommunications Forum (TELFOR)*, 2018.
12. R. S. Gaonkar, K. Gawande, and A. A. Ghatol, "Dietary Menu Planning using Knapsack Problem," in *International Conference on Computational Intelligence and Communication Networks (CICN)*, 2011.



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

 9940 572 462  6381 907 438  ijircce@gmail.com



www.ijircce.com

Scan to save the contact details