SOMAIYA VIDYAVIHAR UNIVERSITY

# K. J. Somaiya School of Engineering

## Department of Computer Engineering

| | | |
|---|---|---|
| **Batch: C1** | **Roll No.:** | **16010122323** |
| **Experiment No 3** | | |
| **Group No:** | | |

**Title: Prepare design document and Plan of project**

---

**Objective:** Chapter No.3 of Mini Project Report will include detailed design document and plan of implementation of the project

---

**Expected Outcome of Experiment:**

| | **At the end of successful completion of the course the student will be able to** |
|---|---|
| CO2 | Identify various hardware and software requirements for problem solution |
| CO5 | Prepare a technical report based on the Mini project. |

---

**Books/ Journals/ Websites referred:**

**1.**
**2.**
**3.**

---

**The students are expected to prepare chapter no 3 in the format given below**

# Chapter 3

## Design Document and Project plan

*A **design document** is crucial in a software project because it serves as a blueprint that outlines the architecture, components, data flow, and technical specifications of the system before implementation **Clear Vision & Planning** will improve collaboration with in the team members.*

A **design document** is crucial in a software project because it serves as a blueprint that outlines the architecture, components, data flow, and technical specifications of the system before implementation.

Typically a design document will have following sections:

# 1. Introduction:

### 1.1 Purpose of the Document

The purpose of this document is to provide a comprehensive technical design overview of the Bytes Diet Planner system. It describes the architecture, module functionalities, data flows, and design principles guiding the development of the platform. This document serves as a reference for the development team, testers, and stakeholders to ensure alignment and consistency during implementation.

---

### 1.2 Intended Audience

This document is intended for the following stakeholders:

| Role | Interest / Use of This Document |
|---|---|
| **Software Developers** | Understand system architecture, module responsibilities, and integration flow. |
| **Testers/QA Engineers** | Design relevant test cases based on modules, workflows, and component interactions. |
| **Project Managers** | Monitor progress, deliverables, deadlines, and implementation plan. |
| **UI/UX Designers** | Align interface design with data flows and user navigation logic. |

| | |
|---|---|
| **Academic Reviewers** | Assess design quality, completeness, and alignment with the Software Engineering curriculum. |
| **Future Developers** | Use the document as a technical reference for maintenance, scaling, or enhancements. |

---

### 1.3 Scope of the Project (Brief)

**Bytes** is a deep-learning-powered web application that focuses on **food recognition**, **calorie estimation**, and **personalized habit tracking** through **image inputs**. Aimed at promoting healthier eating habits and providing insightful nutritional data, the system uses a smartphone camera to capture food images and then detects food types and calculates estimated calorie intake using trained machine learning models.

While the detailed scope is discussed in the project introduction chapter (SRS), this document narrows down on **design-specific concerns**—including:

- How the system will be structured,
- How modules will communicate,
- Database schema design,
- UI/UX components,
- External integrations,
- Deployment and testing strategies.

---

### 1.4 Definitions, Acronyms, and Abbreviations

| Term/Acronym | Definition |
|---|---|
| **Bytes** | Name of the project – A smart food logging and calorie estimation web app. |
| **SRS** | Software Requirements Specification – A document describing functional and non-functional requirements. |
| **ML** | Machine Learning – A field of AI focused on teaching systems to learn from data. |
| **DL** | Deep Learning – A subfield of ML using neural networks with many layers. |

| | |
|---|---|
| **UI/UX** | User Interface / User Experience – Design aspects related to look and feel. |
| **API** | Application Programming Interface – A set of protocols for integrating components or services. |
| **UML** | Unified Modeling Language – A standard way to visualize system architecture and design. |
| **CI/CD** | Continuous Integration / Continuous Deployment – DevOps practices for automating deployment. |
| **CRUD** | Create, Read, Update, Delete – Basic operations for data handling. |

---

### 1.5 References

The following sources were used to draft this Design Document:

1. **Software Requirements Specification (SRS)** for Bytes – for deriving functionalities, modules, and constraints.
2. IEEE 1016-2009 – **IEEE Standard for Information Technology – System Design Description**.
3. ISO/IEC/IEEE 12207 – Systems and Software Engineering – Software Life Cycle Processes.
4. Academic and industrial best practices in software design documentation.
5. Official documentation for:
    a. MERN Stack technologies (MongoDB, Express.js, React, Node.js)
    b. Python libraries and frameworks for ML/DL (e.g., TensorFlow, OpenCV)
    c. UI Design resources (Apple Human Interface Guidelines, Material Design)

# 2. System Overview

## 2.1 System Architecture

#### High-Level Description

The architecture of **Bytes** is based on a modular, scalable **client-server model** leveraging a **MERN stack frontend**(MongoDB, Express.js, React.js, Node.js) and a **Python-powered backend** for deep learning and image processing.

The system follows a **microservices-inspired separation of concerns**: the frontend handles user interaction, the backend handles API routing and business logic, and the **AI engine** is responsible for image classification and calorie estimation.

## Core Components and Interactions

### 1. User Interface (React.js Web App)

The frontend of Bytes is built using **React.js**, providing a responsive and intuitive interface where users can:

- Sign up and log in securely.
- Upload images of their meals.
- Log food items by specifying the meal type (breakfast, lunch, etc.).
- View detailed calorie and macro estimations.
- Track daily and weekly nutritional progress via dashboards.
- Generate and customize daily meal plans based on personal goals, preferences, and seasonal recommendations.
- View seasonal suggestions and export/print their personalized plan.
- Navigate to an About page explaining the platform's mission and tech stack.

**Interactive Features**:

- Food logger

- Meal planner
- Dashboard insights
- Preference filters (veg/non-veg, cuisine, allergies)
- Seasonal highlight section
- Print/export button

---

## 2. Backend Server (Node.js + Express.js)

Acts as the **communication layer** between the frontend and backend logic. It:

- Handles user requests (image upload, profile inputs, logging meals).
- Validates and parses form data (meal time, food tags, etc.).
- Routes images to the AI Engine.
- Manages API requests for food macros and nutrition.
- Stores or retrieves data from the MongoDB database.
- Orchestrates the ML-based meal plan creation pipeline.

---

## 3. AI Engine (Python: Flask or FastAPI)

The AI layer contains the **core intelligence of the Bytes system**, split into the following key services:

- **Food Recognition**:
    - Uses a ResNet50 CNN model trained on **Food-101** (11,000+ images).
    - Classifies the food item from uploaded images.
    - Integrates with a **Nutrition API** to fetch macros and calories per identified food.
- **Meal Plan Generation**:
    - Accepts user details (age, weight, activity level, dietary preferences, allergies).
    - Calculates **BMR and TDEE**.
    - Uses **KNN** to identify suitable nutritional clusters.
    - Applies **Random Forest** to split meals into breakfast, lunch, snacks, and dinner.
    - Applies filters (veg/non-veg, seasonal foods, cuisine, allergies) before finalizing recommendations.
- **Seasonal Food Engine** (Optional ML Component):
    - Fetches current season and filters food items optimal for that time of year.

---

## 4. Database (MongoDB)

All dynamic data is stored in MongoDB collections, including:

- User profile (age, weight, preferences, allergies, dietary goals).

- Food log entries (image, classified food name, meal time, macros, timestamp).
- Meal plans and history.
- BMR/TDEE values per user.
- ML outputs (cluster results, RF meal splits).
- Analytics history for summaries and feedback.

---

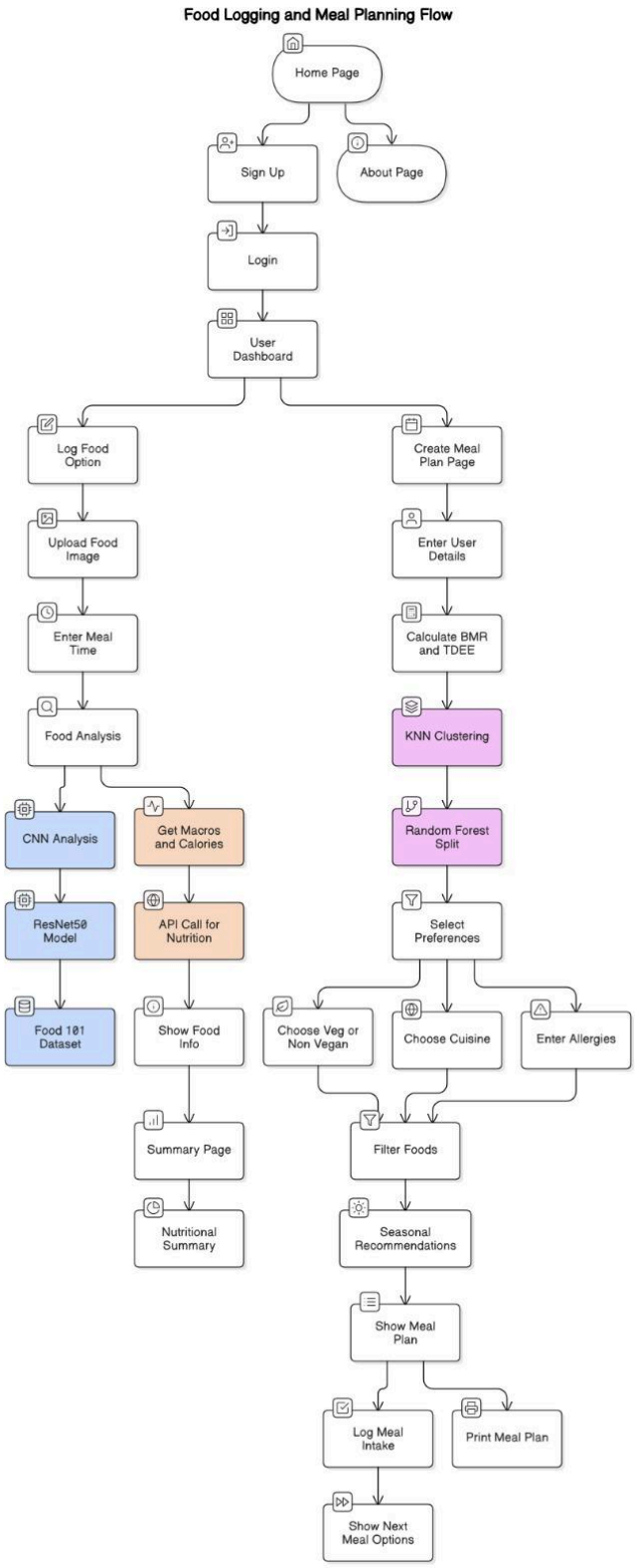
## 5. Auxiliary Services

- **Authentication Service**:
    - Implements **JWT-based secure login** and session management.
    - Ensures protected endpoints for sensitive operations (logging food, generating plans).
- **Data Analytics Module**:
    - Aggregates logged data to generate:
        - Weekly nutritional summaries
        - Daily macro distribution charts
        - Alerts for missing meals or calorie thresholds
    - Drives insights and habit recommendations based on trends.
- **Print/Export Service**:

- Formats the user's daily meal plan into a printable or downloadable PDF format.

- **Admin Panel** (Optional, for future enhancement):
  - Monitor image classification accuracy.
  - Adjust seasonal food mappings.
  - Review model performance and user feedback.

**High-Level Architecture Diagram:**


Food Logging and Meal Planning Flow
Home Page
Sign Up
About Page
Login
User Dashboard
Log Food Option
Upload Food Image
Enter Meal Time
Food Analysis
CNN Analysis
Get Macros and Calories
ResNet50 Model
API Call for Nutrition
Food 101 Dataset
Show Food Info
Summary Page
Nutritional Summary
Create Meal Plan Page
Enter User Details
Calculate BMR and TDEE
KNN Clustering
Random Forest Split
Select Preferences
Choose Veg or Non Vegan
Choose Cuisine
Enter Allergies
Filter Foods
Seasonal Recommendations
Show Meal Plan
Log Meal Intake
Print Meal Plan
Show Next Meal Options

### 2.2 Design Goals

To ensure Bytes is **robust, efficient, and user-friendly**, the system is designed with the following core goals:

---

**1. Scalability**

- The architecture is designed to handle increasing user loads without major changes.
- Uses **stateless APIs**, **horizontal scaling**, and **cloud deployment compatibility** (e.g., Docker, Kubernetes-ready if scaled).

---

**2. Security**

- Implements **JWT-based Authentication & Authorization**.
- **Secure image uploads** using input sanitization, file size/type restrictions.
- **Rate limiting and API throttling** to prevent abuse.
- Follows **OWASP** best practices (e.g., for data storage, session management, and APIs).

---

**3. Performance**

- Uses **lightweight ML models** optimized for inference (e.g., quantized models).
- Implements **caching** for frequently accessed food-calorie mappings.
- Asynchronous request handling and optimized image preprocessing to reduce latency.

---

**4. Maintainability**

- Modular codebase with **clearly defined interfaces**.
- Separation of concerns between:
  - UI
  - API logic
  - ML models
  - Database layer
- Version-controlled APIs and structured database models to support future enhancements.
- Documentation for every service and well-commented codebase.

---

**5. Usability**

- Clean and intuitive UI with **minimalistic, Apple-inspired design**.
- Designed for both casual and fitness-conscious users.
- Uses engaging UI elements like **progress rings, smart suggestions, and weekly summaries**.

---

**6. Extensibility**

- Future-proofed for adding:
  - Barcode scanning,
  - Sustainability scores,
  - Grocery integration,
  - Community features (sharing meals, recipes),
  - Wearable support.

---

## 3.1 Module Description

The **Bytes application** is designed with a modular architecture to ensure flexibility, scalability, and high maintainability. Each module has a specific role, and together they form a cohesive ecosystem delivering an intelligent, personalized nutrition planning experience.

---

**A. User Interface Module (React.js)**

**Responsibilities:**
Facilitates all interactions between the user and the system in an intuitive, responsive, and informative way.

**Key Components:**

- Signup & Login pages (JWT-based auth).
- Food logging interface (image + meal time selection).
- Nutrition summary dashboard (calories, macros, graphs).
- Meal planner (preference selection + personalized plan display).
- Seasonal food recommendations.
- Print/Export functionality.

- About page with platform details and vision.

**Interaction:**

- Communicates with backend (Node.js) using RESTful APIs.
- Sends images and user inputs.
- Receives predictions, analytics, and recommended meal plans.

---

**B. Authentication & Authorization Module**

**Responsibilities:**
Ensures secure access to user-specific data and protects all private operations.

**Key Features:**

- JWT-based user authentication.
- Registration, login, and session management.
- Route protection on both client and server sides.

**Interaction:**

- Built into the backend Express server.
- Every request from the frontend includes a token, which is verified before accessing sensitive data.

---

**C. Food Recognition Module (AI Engine – Python, ResNet50)**

**Responsibilities:**
Analyzes food images and identifies the type of food along with its nutritional content.

**ML Implementation:**

- Utilizes **ResNet50**, pre-trained on the **Food-101** dataset (11,000+ images).
- Classifies images and maps labels to macros and calories using a **Nutrition API**.

**Interaction:**

- Receives base64 or multipart image from backend.
- Returns: Food name, calories, macro breakdown (protein, carbs, fats).
- Logs results to the database for summary generation.

---

**D. Calorie Logging & Food History Module**

**Responsibilities:**
Maintains a detailed record of user's daily meals, calorie intake, and food logs.

**Key Features:**

- Saves meal time, food item, and associated macros.
- Allows deletion or updating of meals.
- Tracks consumption over days/weeks.

**Interaction:**

- CRUD operations on MongoDB for food logs.
- Supports analytical aggregation and trend generation for weekly summaries.

---

**E. Meal Plan Generation Module (ML Engine – KNN + Random Forest)**

**Responsibilities:**
Generates personalized daily meal plans using machine learning.

**ML Implementation:**

- Calculates **BMR** and **TDEE** based on user profile (age, height, weight, sex, activity level).
- Uses **KNN** to cluster users by nutritional needs.
- **Random Forest** assigns appropriate food items into meal slots (breakfast, lunch, snacks, dinner).

- Applies filters:
    - Diet Type (Veg/Non-Veg)
    - Cuisine Preference
    - Allergies
    - Seasonality (filtered by season-appropriate food lists)

**Interaction:**

- Receives user profile from frontend.
- Returns optimized meal plan with foods segmented by meal type and calories matched to remaining TDEE.

---

**F. Seasonal Food Recommendation Module**

**Responsibilities:**
Enriches user experience with contextual food recommendations based on current season.

**Key Features:**

- Detects current season and location.
- Suggests foods that are locally and seasonally optimal.
- Integrates into the meal planner for enhanced filtering.

**Interaction:**

- Uses pre-tagged seasonal food datasets.
- Works alongside the meal plan module.

---

**G. Analytics & Insights Module**

**Responsibilities:**
Provides users with visual feedback, trends, and health progress.

**Key Features:**

- Nutritional summaries: Calories consumed, remaining macros.
- Weekly/monthly trends: Graphs, charts.
- Behavioral insights: Missed meals, excess consumption, etc.
- Habit tracking (e.g., hydration, meal times).

**Interaction:**

- Aggregates data from food logs and meal plans in MongoDB.
- Sends visualizations to the frontend dashboard.

---

**H. API Gateway (Node.js + Express.js)**

**Responsibilities:**
Serves as the central entry point for all API calls and manages service routing.

**Key Features:**

- Image upload and parsing.
- Route requests to:
    - AI Engine (image analysis)
    - Database (food logs, profile data)
    - ML Engine (meal planning)
- Error handling, request validation, rate limiting.

**Interaction:**

- Mediates communication between frontend and backend services.
- Performs security checks (token verification).

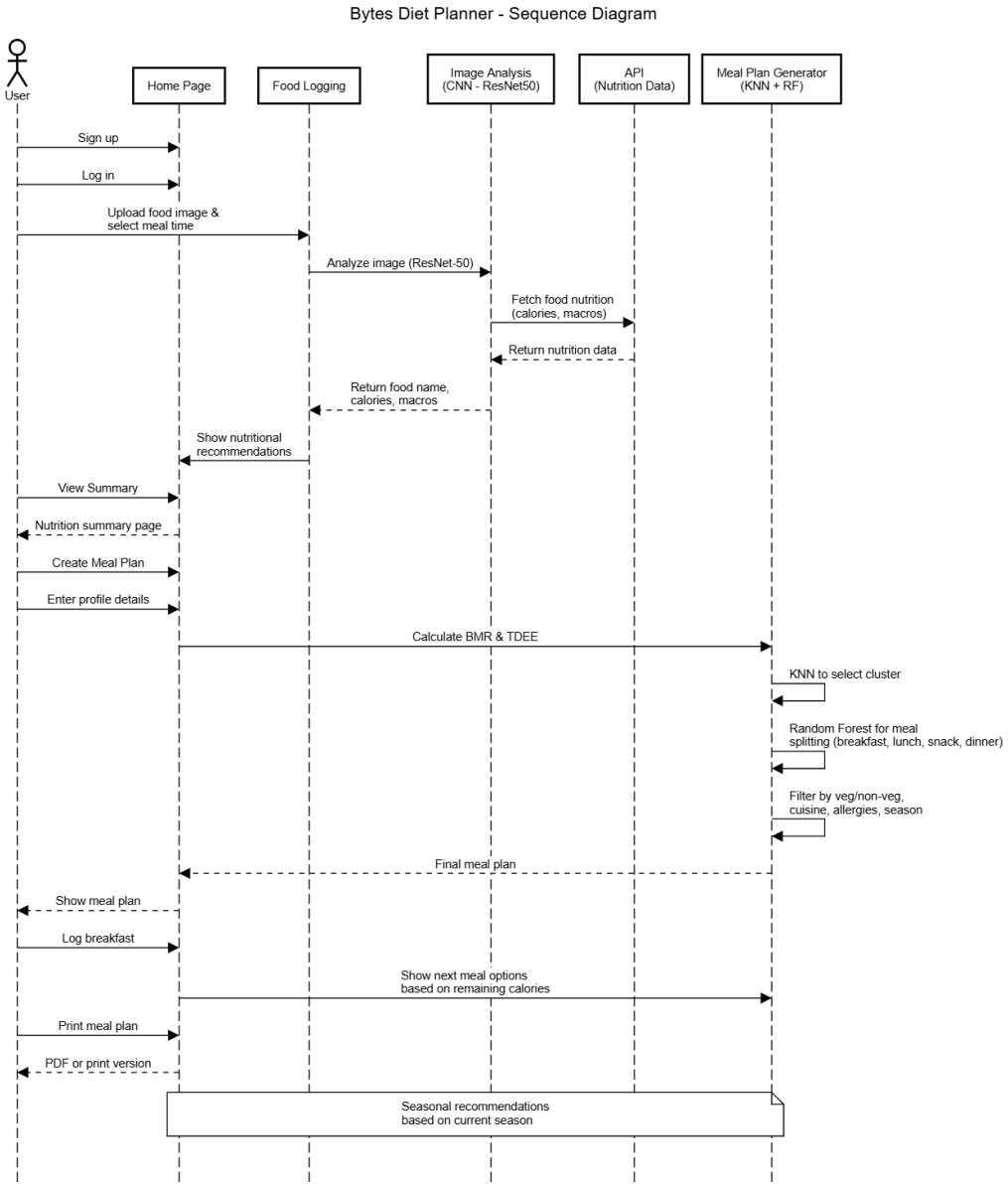
---

## 3.2 Data Flow & Components

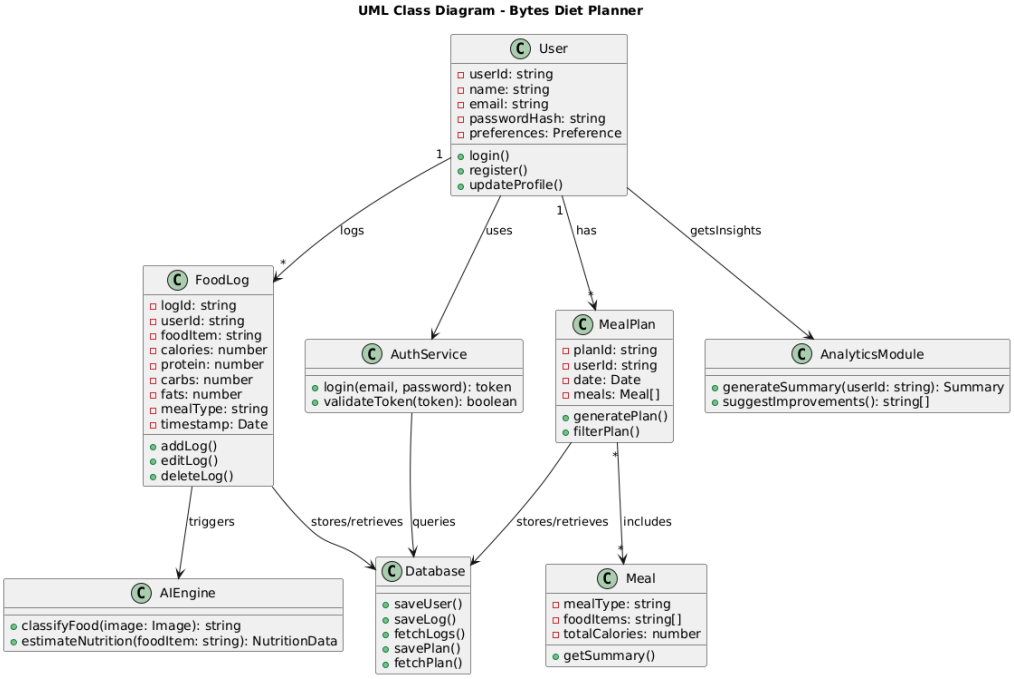
**A. Logical Data Flow:**

1. **User uploads an image** of their meal and selects the meal time (e.g., breakfast).
2. **API Gateway** receives the image and forwards it to the **Food Recognition Module**.
3. **AI Engine** classifies the food and retrieves its nutritional data.
4. Classification results and macros are **saved in MongoDB** under the user's food log.
5. The frontend queries this data and displays it in:
    - Dashboard
    - Nutritional summaries
6. When the user navigates to the **Meal Planner**:
    - Inputs profile details (height, weight, preferences)
    - Backend calculates BMR and TDEE.
    - **KNN & RF** generate and serve a **daily plan**, considering dietary and seasonal filters.
7. **User logs meals** throughout the day.
    - Remaining meal suggestions are dynamically adjusted based on prior intake.
8. At any time, user can:
    - View graphs of progress.
    - Download or print their **personalized meal plan**.
    - Explore seasonal food recommendations.

## B. Sequence Diagram




## C. UML Diagram:




## 3.3 Database Design

### A. Database Type

- **MongoDB (NoSQL)** is used due to its flexibility and scalability with unstructured or semi-structured data like image references, logs, and user habits.

### B. Core Collections

| Collection | Description |
|---|---|
| `userModel.js` → `users` | Stores user data (credentials, preferences, demographics) |
| `foodModel.js` → `meals` | Logs meals with timestamps, images, predicted food items, calories |

| | |
|---|---|
| MealPlan.js → mealPlans | Predefined or recommended meal plans (for day) |

**C. Sample Schema:**

```
const mongoose = require('mongoose');

const foodSchema = new mongoose.Schema({
  user_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  food_item: String,
  estimated_calories: Number,
  meal_time: Date,
  image_url: String,
  confidence: Number
}, { timestamps: true });

module.exports = mongoose.model('Food', foodSchema);
```

**D. Indexing Strategy**

- Index on `user_id` and `meal_time` for quick user-specific queries
- Compound index on `food_item + meal_time` for analytics
- Text index on `food_item` for search functionality

## 3.4 User Interface Design

### A. Design Philosophy

- **Clean, futuristic UI with Apple-like glass morphism and fluid animations**
- **Accessibility and minimalism prioritized**

### B. Core Screens

| Screen | Description |
|---|---|
| Login/Signup | Secure user onboarding with form validation |
| Dashboard | Summary of calories, meals, progress |
| Upload Page | Upload food image, preview, and submit |
| Meal Log | Detailed list of past meals, calories, and timings |
| Meal Plan | Renders weekly/daily plan from `mealPlanRoutes.js` |
| Analytics Page | Graphs, trends, goal progress |

### C. Navigation Flow

Login → Dashboard → Upload Food → View Result → Meal Log → Meal Plan

## 3.5 External Interfaces

### A. RESTful APIs

| Endpoint | Method | Description |
|---|---|---|
| `/api/login` | POST | Authenticate user |
| `/api/register` | POST | Register new user |
| `/api/upload` | POST | Upload food image |
| `/api/meals` | GET | Fetch user meal logs |
| `/api/habits` | GET/POST | Log and fetch habits |
| `/api/insights` | GET | Get analytical summaries |

### B. Third-Party Integrations

| Service | Purpose |
|---|---|
| **Cloudinary / Firebase Storage** | Image upload and storage |

| | |
|---|---|
| **Open Food Facts / USDA API** (Optional) | Food-to-calorie mapping |
| **Figma** | UI/UX prototyping |
| **Mongoose ODM** | Database modeling |

---

### C. Hardware Interfaces

- Currently limited to **smartphone camera** for food image input
- Optional future extensions: barcode scanners, wearables

## 4. Project and Implementation Plan:

#### 4.1 Deliverables:

- **Source Code**
  The complete source code for the Bytes project, including the backend built with Node.js and Express, the frontend developed using React.js, and the AI engine implemented in Python using Flask or FastAPI.

- **Database Models**
  MongoDB schemas for all key entities such as users, meals, and plans, along with seed data to populate initial values and facilitate testing.

- **REST APIs**
  A full set of RESTful API endpoints that support authentication, food image uploads, meal logging, and dynamic meal plan generation.

- **User Documentation**
  A comprehensive user guide explaining how to use the Bytes platform, including instructions for login, uploading food images, viewing health insights, and managing meal logs.

- **Installation & Deployment Guide**
  Detailed step-by-step instructions for setting up and deploying all components

of the project, including the frontend, backend server, AI service, and the MongoDB database.

- **UML & Architecture Diagrams**
  A collection of diagrams such as class diagrams, data flow diagrams, and system/component interaction diagrams that explain the architecture and design of the platform.

- **Testing Report**
  Documentation of the testing process, including unit tests, integration tests, and user acceptance testing (UAT). This includes logs, screenshots, and summaries of outcomes.

- **Model Weights (Optional)**
  Pre-trained weights of the food classification AI model, provided optionally for deployment purposes or further training/customization.

### 4.2 Team Roles and Responsibilities and delivery schedule

| Name of the Task | Developer | Tester | Approver | Date of Delivery |
|---|---|---|---|---|
| User Auth (Register/Log in) | Vedansh | Rhea | Project Mentor | 2025-03-25 |
| Food Upload & Classification | Hriday | Vedansh | Project Mentor | 2025-03-28 |
| Diet Plan Recommender | Rhea | Hriday | Project Mentor | 2025-04-03 |
| Nutritional Summary Module | Hriday | Vedansh | Project Mentor | 2025-04-03 |
| Database Setup & Seeding | Vedansh | Rhea | Project Mentor | 2025-04-05 |
| Analytics Module | Rhea | Hriday | Project Mentor | 2025-04-05 |

### 4.3 Risk Management Plan

In the Bytes project, risk management is a continuous and proactive process. Potential risks are identified early in the development cycle and categorized based on their impact and likelihood. These risks include technical challenges, data security issues, integration failures, deployment errors, and delayed timelines. Each risk is assessed, and mitigation strategies are put in place.

For technical issues, thorough code reviews, continuous integration, and version control are used to maintain code quality and prevent system-breaking bugs. In case of data breaches or privacy violations, strict security protocols like encryption, authentication mechanisms, and regular audits are implemented. To handle integration risks between modules or external APIs, sandbox environments and mock services are used during development and testing. For deployment-related risks, automated CI/CD pipelines are established with rigorous testing before any production push. If delays occur, project timelines are adjusted and additional resources may be allocated to meet deliverables. A proper escalation matrix is also in place to ensure quick response and resolution.

---

## 5. Testing & Deployment Plan

### 5.1 Testing Strategy

The testing approach for Bytes ensures the stability, functionality, and user satisfaction of the platform. It is divided into several levels.

- **Unit Testing** focuses on individual components or functions. Each function is tested in isolation using frameworks like Jest or Mocha to ensure logic correctness.
- **Integration Testing** verifies that different modules of the application work together as expected. This includes testing data flow between the frontend, backend, and database.
- **System Testing** validates the entire system as a whole. It includes functional testing, performance testing, and load testing to check how the system performs under expected and peak conditions.
- **User Acceptance Testing (UAT)** involves real users testing the platform to confirm that it meets business requirements. Feedback from UAT is used to make final adjustments before deployment.

### 5.2 Deployment Plan

Deployment of Bytes is executed through a structured and automated pipeline to minimize errors and downtime. The environment is first prepared with all necessary dependencies, configurations, and environment variables. Deployment begins in a staging environment, where final tests are conducted to simulate the production setup.

After successful staging validation, the system is pushed to production using tools like GitHub Actions or Docker-based containers.

To ensure reliability, a **blue-green deployment** strategy or **canary release** may be used to gradually expose the new version to users, minimizing impact in case of issues. If a failure is detected post-deployment, a **rollback mechanism** is in place to quickly revert to the previous stable version. Logs and monitoring tools are continuously used to track system health and performance post-deployment.

*The next chapter, chapter no . 4 will explain test cases, test plan and test reports in detail*