**FrontEnd  
1. Main Components**

* **PatientDashboard: This component handles the patient-specific data and displays relevant health information.**
* **DoctorDashboard: Similar to PatientDashboard but designed for doctors to view patient data summaries and updates.**

**2. Supporting Components**

**These components serve specific functions and could be linked to the main dashboards:**

* **ActivityRecording and ActivityDataTable: Capture and display activity-related data.**
* **BloodSugarInput, BloodSugarTable, and BloodSugarChart: Handle blood sugar data input and visualization.**
* **DurationInput: Allows the input of time-based data, possibly related to activities or medications.**
* **EnhancedMealInsulin: Might manage insulin data connected to meal intake.**
* **FoodDatabase: Possibly an interface to display or input information from a food database.**
* **MealInput and MealHistory: Record and display meal data, supporting blood sugar and insulin tracking.**
* **Register and Signin: Handle user authentication and account setup.**

**PatientConstants: Likely contains constants or configuration specific to patient data.  
backend  
  
Project Structure  
  
FrontEnd**

**├── Main Components**

**│ ├── PatientDashboard**

**│ └── DoctorDashboard**

**│**

**├── Supporting Components**

**│ ├── Activity Components**

**│ │ ├── ActivityRecording**

**│ │ └── ActivityDataTable**

**│ │**

**│ ├── Blood Sugar Components**

**│ │ ├── BloodSugarInput**

**│ │ ├── BloodSugarTable**

**│ │ └── BloodSugarChart**

**│ │**

**│ ├── Insulin and Meal Management**

**│ │ ├── EnhancedMealInsulin**

**│ │ ├── MealInput ---food section(subcomponent)**

**│ │ └── MealHistory**

**│ │**

**│ ├── Food and Duration Management**

**│ │ ├── FoodDatabase**

**│ │ └── DurationInput**

**│ │**

**│ └── Authentication Components**

**│ ├── Register**

**│ └── Signin**

**│**

**├── Constants**

**│ ├── PatientConstants**

**│ ├── activityConstants.js**

**│ └── conversionConstants.js**

**│**

**└── Hooks**

**└── usePatientConstants**

backend/

├── main.py

├── config.py

├── activity.py

├── blood\_sugar.py

├── constants.py

├── enhanced\_meal\_insulin.py

├── meal\_insulin.py

├── models/

│ ├── \_\_init\_\_.py

│ ├── food\_data.py

│ └── portion\_converter.py

├── routes/

│ ├── \_\_init\_\_.py

│ ├── auth\_routes.py

│ ├── doctor\_routes.py

│ ├── food\_routes.py

│ └── test\_routes.py

├── services/

│ └── food\_service.py

└── utils/

├── \_\_init\_\_.py

├── auth.py

└── error\_handler.py

**Key Features**

1. Diabetes Management System:
   * Meal insulin calculations
   * Blood sugar monitoring
   * Activity tracking
   * Food data management
2. Authentication System:
   * Token-based authentication
   * 24-hour token expiry
   * Doctor-specific routes
3. Data Management:
   * MongoDB integration
   * Food data services
   * Portion conversion utilities

 Food Management:

* Multi-category food database
* Flexible search functionality
* Portion conversion system
* Special portion handling

 Authentication:

* JWT-based authentication
* User verification
* Token expiration handling
* Secure route protection

 Error Handling:

* Centralized error management
* Standardized API responses
* Detailed error logging

**Common Integration Points**

* MongoDB connection available via app.mongo
* Logging via app.logger
* CORS configured for frontend integration
* Error handling middleware

**Database Integration**

* MongoDB connection via app.mongo
* User authentication checks
* Food data storage and retrieval

**Authentication Flow**

1. Token Requirements:
   * Bearer token in Authorization header
   * Valid JWT signature
   * Non-expired token
   * Existing user in database
2. Error Handling:
   * Centralized error logging
   * Standardized API responses
   * Detailed error tracking

**Project Technical Overview**

**Core Application Structure**

**Application Factory (\_\_init\_\_.py)**

* Creates and configures Flask application
* Key configurations:
  + MongoDB connection: mongodb://localhost:27017/native\_new
  + CORS enabled for http://localhost:3000
  + Configurable environment (testing/production)
  + Token expiry: 24 hours

**Main Application (main.py)**

* Entry point for the application
* Registered Blueprints:
  + food\_routes
  + auth\_routes
  + doctor\_routes
  + test\_routes
  + meal\_insulin\_bp
  + activity\_bp
  + blood\_sugar\_bp
* Global error handlers for 404 and 500
* MongoDB instance available to all blueprints via app.mongo
* Runs on host '0.0.0.0', port 5000 in debug mode

**Configuration (config.py)**

* MongoDB configuration
* Logging setup (DEBUG level)
* CORS configuration
* Core app settings:
  + Secret key configuration
  + UTC timezone
  + Token expiration
  + Allowed origins

**Constants (constants.py)**

Important calculation factors and default values:

1. Absorption Factors:

{ 'fast': 1.0, 'medium': 0.9, 'slow': 0.8, 'unknown': 0.9 }

1. Timing Factors:

{ 'breakfast': 1.2, 'normal': 1.0, 'late\_night': 0.9 }

3. Default Values:

* Insulin to Carb Ratio: 1/10
* Correction Factor: 50
* Target Glucose: 100
* Activity Coefficients: Range from -0.2 to 0.2

1. Insulin Timing Guidelines:

* Fast absorption: 15 minutes before meal
* Medium absorption: 10 minutes before meal
* Slow absorption: 5 minutes before meal

**Services**

**Food Service System (food\_service.py)**

1. Food Categories:
2. { 'basic': FOOD\_DATABASE, 'starch': STARCH\_LIST, 'fruits': FRUITS, 'dairy': MILK\_AND\_DAIRY, 'sweets': SWEETS\_AND\_DESSERTS, 'snacks': SNACKS, 'common\_snacks': COMMON\_SNACKS, 'high\_protein': HIGH\_PROTEIN\_FOODS, 'high\_fat': HIGH\_FAT\_FOODS, 'indian': INDIAN\_DISHES, 'chinese': CHINESE\_DISHES, 'italian': ITALIAN\_DISHES }  
     
   Key Functions:

* get\_food\_details(food\_name): Retrieves food details from any category
* search\_food(query, category=None): Searches food items with optional category filter
* calculate\_absorption\_factor(absorption\_type, meal\_timing): Calculates absorption based on food type and timing
* convert\_to\_standard\_nutrients(portion\_size, measurement\_type, food\_details, converter): Handles portion conversion

Special Portion Measurements:  
{ 'plate': 300g, 'palm': 85g, 'handful': 30g, 'fist': 150g, 'bowl': 400ml }  
  
**UTILS**  
  
**Authentication System (utils/auth.py)**

1. Token-based Authentication:

* JWT implementation
* Token validation middleware
* User verification against MongoDB

1. Key Decorator: @token\_required

* Validates Authorization header
* Decodes JWT token
* Verifies user existence
* Handles token expiration
* Error states:
  + Missing token (401)
  + Expired token (401)
  + Invalid token (401)
  + User not found (401)

**Error Handling (utils/error\_handler.py)**

1. Global API Error Handler:

* Decorator: @api\_error\_handler
* Centralizes error logging
* Returns standardized 500 error response

Logs detailed error information

utils/\_\_init\_\_.py:   
Exports core utility functions:

* token\_required: Authentication decorator
* api\_error\_handler: Error handling decorator
* Creates a single import point for common utilities

**API Routes Structure  
routes/\_\_init\_\_.py:**A blueprint registry that:

* Imports all route blueprints (auth, doctor, food, test)
* Makes routes available for registration in main.py
* Centralizes route management
* Exports route blueprints via **all**

**Authentication Routes (auth\_routes.py)**

1. Endpoints:

POST /login - Handles user authentication - Returns JWT token - Includes user type and basic info POST /register - New user registration - Sets default values for diabetes management - Validates unique username/email GET /dashboard - Protected route - Returns user profile info

1. User Registration Data:

* Default values set for new users:
  + insulin\_to\_carb\_ratio
  + correction\_factor
  + target\_glucose
  + activity\_coefficients

**Doctor Routes (doctor\_routes.py)**

1. Protected Endpoints:

GET /api/doctor/patients - Lists all patients - Filtered patient info (no passwords) GET /api/doctor/patient-constants/<patient\_id> - Retrieves patient's medical constants POST /api/doctor/update-patient-constants - Updates patient's treatment parameters - Validates all required fields

1. Access Control:

* All routes require doctor user\_type
* JWT validation for each request
* Error handling for unauthorized access

**Food Management Routes (food\_routes.py)**

1. Core Endpoints:

GET /api/food/categories - Lists all food categories - Returns measurement systems - Includes standard portions GET /api/food/search - Search functionality with category filter - Returns detailed food information POST /api/food/custom - Add custom food items - Validates serving sizes and nutrients GET /api/food/nutritional-summary - Calculates meal nutrient totals - Converts portions to standard units

1. Additional Features:

GET/POST /api/food/favorite - Manage favorite foods - User-specific favorites list GET /api/food/measurements - Lists supported measurement types - Standard portion conversions

**Testing Route (test\_routes.py)**

GET /test

- Simple health check endpoint

- Verifies backend status

**Data Models**

**Food Categories Structure:**

**{ 'basic\_foods': FOOD\_DATABASE, 'starch': STARCH\_LIST, 'vegetables': STARCHY\_VEGETABLES, 'pulses': PULSES, 'fruits': FRUITS, 'dairy': MILK\_AND\_DAIRY, 'sweets': SWEETS\_AND\_DESSERTS, 'snacks': [SNACKS, COMMON\_SNACKS], 'cuisines': { 'indian': INDIAN\_DISHES, 'chinese': CHINESE\_DISHES, 'italian': ITALIAN\_DISHES } }**

**Custom Food Structure:**

**{ 'user\_id': str, 'name': str, 'serving\_size': { 'amount': float, 'unit': str }, 'carbs': float, 'protein': float, 'fat': float, 'description': str, 'absorption\_type': str, 'created\_at': datetime }**

**API Integration Patterns**

**Authentication Flow:**

***# Login Request* POST /login { "username": str, "password": str, "user\_type": str } *# Response* { "token": jwt\_token, "user\_type": str, "firstName": str, "lastName": str }**

Food Management Flow:

*# Search Request* GET /api/food/search?q=query&category=optional\_category *# Nutritional Summary Request* POST /api/food/nutritional-summary { "meal\_items": [ { "name": str, "portion": float, "measurement": str } ] }

**Doctor-Patient Management:**

*# Update Patient Constants* POST /api/doctor/update-patient-constants { "patientId": str, "constants": { "insulin\_to\_carb\_ratio": float, "correction\_factor": float, "target\_glucose": float, "ACTIVITY\_COEFFICIENTS": dict } }

**Error Handling**

1. Common HTTP Status Codes:
   * 400: Bad Request (Invalid input)
   * 401: Unauthorized (Invalid/missing token)
   * 403: Forbidden (Wrong user type)
   * 404: Not Found (Resource missing)
   * 500: Internal Server Error
2. Error Response Format:

{ "error": "Error description", "message": "Optional detailed message" }

**Security Features**

1. Authentication:
   * JWT-based token system
   * 24-hour token expiry
   * User type verification
   * Password hashing
2. Authorization:
   * Route-specific user type checking
   * Protected routes with @token\_required
   * Patient data access control

**Models  
models/portion\_converter.py:  
Key Components**

**Base Measurement Units**

1. Volume Measurements:
   * Base unit: milliliters (ml)
   * Supported conversions:
     + cup (240ml)
     + tablespoon (15ml)
     + teaspoon (5ml)
2. Weight Measurements:
   * Base unit: grams (g)
   * Supported conversions:
     + kilogram (1000g)
     + ounce (28.35g)
     + pound (453.6g)

**Core Functions**

**convert\_to\_standard(amount, from\_unit)**

* Purpose: Converts any measurement to its base unit (ml or g)
* Parameters:
  + amount: Numerical value of the measurement
  + from\_unit: Original unit of measurement
* Returns: Converted value in base units (ml or g)

**convert\_between\_units(amount, from\_unit, to\_unit)**

* Purpose: Converts measurements between different units
* Parameters:
  + amount: Numerical value to convert
  + from\_unit: Original unit
  + to\_unit: Target unit
* Returns: Converted value in target unit

**get\_supported\_measurements()**

* Purpose: Provides all supported measurement types
* Returns: Dictionary containing:
  + volume: List of volume measurements
  + weight: List of weight measurements
  + standard\_portions: Standard portion definitions

**Integration Points**

* Works with food\_data.py's STANDARD\_PORTIONS
* Used by food\_service.py for portion calculations
* Supports food\_routes.py for API responses

**Usage Example**

converter = PortionConverter()

*# Convert 2 cups to milliliters*

ml\_amount = converter.convert\_to\_standard(2, "cup") *# Returns 480*

*# Convert between units*

oz\_amount = converter.convert\_between\_units(1000, "g", "oz") *# Returns ~35.27*

**models/\_\_init\_\_.py:**  
 This file exports all food-related data models and the portion converter:

* Makes food databases accessible (various cuisine types, food categories)
* Exports the PortionConverter class
* Creates a centralized access point for all food-related constants

**Core Backends**

**1. Activity Management (activity.py)**

* Handles activity tracking and impact calculations
* Key Features:
  + Records expected and completed activities
  + Duration parsing/formatting (HH:MM format)
  + Activity impact calculation based on intensity levels (-2 to 2)
  + Historical activity retrieval for users and doctors
  + Activity coefficient calculation for insulin adjustments

**2. Blood Sugar Management (blood\_sugar.py)**

* Manages blood sugar monitoring and data
* Key Features:
  + Blood sugar recording (mg/dL)
  + Value validation (0-600 mg/dL range)
  + Unit conversion (mg/dL ↔ mmol/L)
  + Historical data retrieval with date filtering
  + Doctor-specific patient data access

**3. Meal and Insulin Management (meal\_insulin.py)**

* Complex insulin calculation system
* Key Features:
  + Meal nutrition calculation
  + Advanced insulin suggestions considering:
    - Carb-to-insulin ratios
    - Protein/fat contributions
    - Activity impact
    - Time-of-day factors
    - Blood glucose corrections
  + Portion conversion system
  + Meal logging with detailed breakdowns
  + Historical meal data retrieval

**Integration Points**

**Database Schema**

1. Activities Collection:

{ user\_id: String, type: String (expected/completed), level: Number (-2 to 2), duration: String (HH:MM), expectedTime/completedTime: DateTime, timestamp: DateTime }

1. Blood Sugar Collection:

user\_id: String, bloodSugar: Number (mg/dL), timestamp: DateTime }  
  
Meals Collection:

{ user\_id: String, mealType: String, foodItems: Array, activities: Array, nutrition: Object, bloodSugar: Number, suggestedInsulin: Number, insulinCalculation: Object, timestamp: DateTime }  
  
**API Endpoints**

**Activity Endpoints:**

* POST /api/record-activities: Record expected/completed activities
* GET /api/activity-history: Get user's activity history
* GET /api/patient/{patient\_id}/activity-history: Doctor access to patient activities

**Blood Sugar Endpoints:**

* POST /api/blood-sugar: Record blood sugar reading
* GET /api/blood-sugar: Get user's blood sugar history
* GET /doctor/patient/{patient\_id}/blood-sugar: Doctor access to patient readings

**Meal/Insulin Endpoints:**

* POST /api/meal: Submit meal and get insulin calculation
* GET /api/meals: Get user's meal history
* GET /api/doctor/meal-history/{patient\_id}: Doctor access to patient meals

**Key Calculations**

1. Activity Impact:

level\_impact = { -2: -0.2, *# Sleep* -1: -0.1, *# Very Low* 1: 0.1, *# High* 2: 0.2 *# Vigorous* }  
  
2. Meal Timing Factors:  
  
timing\_factors = { 'breakfast': 1.2, 'lunch': 1.0, 'dinner': 0.9, 'snack': 1.0 }  
  
3. Insulin Calculation Components:

* Base insulin = (carbs / insulin\_to\_carb\_ratio)
* Protein/fat contribution = (protein \* 0.1 + fat \* 0.1) / insulin\_to\_carb\_ratio
* Activity adjustment = base\_insulin \* (1 + activity\_coefficient)
* Correction insulin = (current\_glucose - target\_glucose) / correction\_factor

**Authentication & Security**

* Token-based authentication required for all endpoints
* Doctor-specific route protection
* Input validation for all numerical values
* Standardized error handling and logging

**3. Enhanced Meal and Insulin Management (enhanced\_meal\_insulin.py)**

* Advanced insulin calculation system with enhanced features
* Key Features:
  + Sophisticated absorption rate handling
  + Detailed nutrition calculations
  + Enhanced insulin requirement calculations
  + Meal summary and analytics
  + Improved meal timing factors

**Absorption Rates System**

DEFAULT\_ABSORPTION\_RATES = { 'fast': 1.2, *# Simple sugars* 'medium': 1.0, *# Complex carbs* 'slow': 0.8 *# High fiber foods* }  
  
**Enhanced Nutrition Tracking**

* Detailed nutrition breakdown:
  + Calories
  + Carbohydrates
  + Protein
  + Fat
  + Fiber
  + Absorption-adjusted carbs

**Advanced Insulin Calculations**

1. Base Components:
2. timing\_coefficients = { 'breakfast': 1.1, *# Dawn phenomenon* 'late\_night': 0.9, *# Night sensitivity* 'normal': 1.0 }  
   Calculation Flow:

* Base insulin = carbs \* insulin\_to\_carb\_ratio
* Timing adjustment = base\_insulin \* timing\_coefficient
* Activity adjustment = timing\_adjusted\_insulin \* (1 + activity\_coefficient)
* Final insulin = activity\_adjusted\_insulin + correction\_insulin

**New API Endpoints**

1. Enhanced Meal Endpoints:

* POST /api/v2/calculate-insulin: Advanced insulin calculation
* POST /api/v2/meal: Submit meal with enhanced data
* GET /api/v2/meals/{meal\_id}: Get specific meal details
* GET /api/v2/meals/summary: Get meal analytics

**Enhanced Meal Schema**

{ "user\_id": "String", "timestamp": "DateTime", "meal\_type": "String", "meal\_items": "Array", "nutrition": { "calories": "Number", "carbs": "Number", "protein": "Number", "fat": "Number", "fiber": "Number", "absorption\_adjusted\_carbs": "Number" }, "blood\_glucose": "Number", "activities": "Array", "insulin\_calculation": { "base\_insulin": "Number", "activity\_adjusted": "Number", "correction\_dose": "Number", "total\_suggested": "Number", "factors": { "activity\_coefficient": "Number", "timing\_adjustment": "Number" } }, "notes": "String", "tags": "Array", "meal\_timing": "String" }  
  
**Meal Summary Analytics**

* Total meals count
* Average carbohydrates
* Average insulin doses
* Meal type distribution
* Common foods analysis
* Timing distribution

**Integration Points**

[Previous Integration Points sections remain the same]

**Additional Key Calculations**

1. Absorption Factor Calculation:
2. base\_factors = { 'fast': 1.0, 'medium': 0.9, 'slow': 0.8, 'unknown': 0.9 } timing\_factors = { 'breakfast': 1.2, 'normal': 1.0, 'late\_night': 0.9 } absorption\_factor = base\_factor \* timing\_factor  
     
   Enhanced Nutrition Calculation:

* Per-item nutrition calculation
* Portion-adjusted values
* Absorption rate application
* Total nutrition aggregation

1. Advanced Insulin Requirement:

* Carb-based calculation
* Activity impact adjustment
* Meal timing modification
* Blood glucose correction
* Total insulin computation