High-Level Structure of the Egg Incubator System

The egg incubator system consists of three main components, each responsible for specific tasks to ensure the proper operation and monitoring of the incubation process:

Hardware Layer (Arduino Microcontroller):

This layer is responsible for real-time data collection and execution of control commands. It interfaces with sensors (temperature, humidity) and actuators (rotation motor, heater, humidifier) to maintain optimal incubation conditions.

Backend System (Server):

The back-end serves as the central processing unit for data management and analysis. It connects the hardware layer with the front-end, processes real-time data, stores logs, and performs predictive analysis for system optimization.

Frontend Interface (Web/Mobile App):

The front-end provides a user-friendly interface for monitoring and controlling the incubator. It visualizes data, displays predictions and notifications, and allows users to send commands to the system.

Features of the Software System

Arduino Program (Embedded Software)

Framework: Arduino IDE with C++ libraries.

Libraries:

Sensors: DHT for temperature/humidity.

Communication: WiFi or Serial libraries.

Backend (Server)

Framework: Node.js with Express.js.

Database: MongoDB for storing settings, real-time data, logs, and predictions.

Libraries:

Data Processing: TensorFlow.js for prediction models, SciKit-learn for data analysis.

Endpoints:

Data: POST /data: Receive real-time data from Arduino.

GET /data: Send historical and real-time data to the front-end.

Frontend (Web/Mobile App)

Framework: Flutter for cross-platform compatibility.

UI Components:

Graph Viewer: Real-time and historical data visualization.

Control Panel: Manual/automatic controls for temperature, humidity, and rotation.

Notifications: Alerts for anomalies, hatching events, and success predictions.

Libraries:

Graphs: fl_chart for rendering dynamic charts and graphs.

Networking: http/dio for API calls to the back-end.

Data/Process Flow

Initialization (Setup Parameters):

- ✓ User inputs setup parameters (optimal temperature, humidity, start time, etc.) via the front-end.
- ✓ The server validates inputs, stores them in the database, and sends them to the Arduino for configuration.

Real-Time Monitoring and Data Logging:

- ✓ Arduino sends temperature, humidity, rotation events, and timestamps at regular intervals (every 10–30 seconds; this balance prevents overloading while ensuring frequent updates).
- ✓ Server updates the database with these readings.
- ✓ Server optionally logs any additional metadata or error information at this step.

Data Retrieval and Presentation:

- ✓ The server retrieves data (temperature, humidity, rotation events, timestamps) from the database for front-end display.
- ✓ Real-time and historical data are provided as arrays or formatted objects for visualization.

Prediction:

- ✓ The server uses machine learning or heuristic-based algorithms to predict future trends (temperature, humidity, rotations, hatching time, success rate, etc.).
- ✓ Predictions are stored and made accessible to the front-end.

Notifications and Alerts:

- ✓ Server monitors conditions for critical events (e.g., significant deviation from optimal temperature/humidity, hatching time).
- ✓ Alerts are logged in the database and sent to the front-end.

Command Execution:

- ✓ User commands (e.g., turn ON/OFF, modify parameters) are sent from the front-end to the server.
- ✓ The server processes and relays commands to the Arduino.

Backend Architecture (Class Diagram)

Class Name	Attributes	Methods	Description
Starter	db_connection arduino_connection	create_database() initialize_arduino(parameters) start_logging_process()	Handles initialization of the database, Arduino communication, and logging setup.
DataManager	temperature_data humidity_data rotation_data timestamps	update_database(data) retrieve_data(data_type, time_range) log_event(event_type, description)	Manages the storage and retrieval of data to/from the database and logs key events such as updates and errors.
PredictionManager	historical_data prediction_model	retrieve_historical_data(data_type, range) predict_future_values(data_type, n_steps) calculate_success_rate()	Uses past data to predict future values and calculate performance metrics like hatching success rates.
NotificationManager	critical_events alerts_log	check_conditions(data) generate_alert(alert_type, description) log_notification(notification)	Monitors system state, identifies critical events, generates notifications, and logs them in the database.
FrontendInterface	response_data	format_data_for_frontend(data) send_to_frontend(data) receive_commands_from_frontend()	Formats data to be sent to the front-end, transmits it, and processes user commands received from the front-end.
CommandHandler	command_queue	process_command(command) send_to_arduino(command)	Manages commands received from the front-end, processes them, and sends instructions to the Arduino program.

Key Notes:

Relationships:

- ✓ FrontendInterface interacts with NotificationManager and DataManager to fetch and send data.
- ✓ PredictionManager relies on DataManager for historical data.
- ✓ CommandHandler uses Starter for Arduino initialization and directly sends instructions to the Arduino.

Interactions:

- ✓ The FrontendInterface acts as the middle layer between the backend and the user-facing frontend.
- ✓ Starter initializes the database and sets up communication with the Arduino.
- ✓ NotificationManager and PredictionManager provide higher-level functionality by processing raw data.