Project Report: ChronoSync - A Full-Stack Wellness Application

---

Executive Summary

ChronoSync is an innovative web application designed as a personal wellness companion, built to help users master their natural circadian rhythms. It provides a holistic approach to health by connecting a user's daily habits—from sleep and sunlight to diet and stress—with a powerful AI analysis. The project successfully showcases a seamless integration of modern web technologies, secure user authentication, and a private, locally-run AI model, all wrapped in an intuitive and visually engaging user interface. ChronoSync is more than a simple data logger; it's a tool built to empower users with self-knowledge, encouraging them to achieve a healthier, more productive lifestyle.

---

1. Project Vision & Core Functionality

Objective: To create a digital tool that provides users with a personalized Circadian Rhythm Score (CRS) and actionable advice based on their lifestyle data.

Key Features Implemented:

- **Secure Authentication**: A complete sign-up and login system allows for a personalized user experience. The system handles password hashing, session management, and redirection to the user's private dashboard.
- **AI-Powered Questionnaire**: The core of the application is a detailed quiz that collects a variety of data points, including sleep habits, work schedules, diet, and stress. The front-end is designed with custom-styled form fields for a clean, user-friendly experience.
- **Local AI Backend**: The project uses **Ollama** to run a local large language model (e.g., Llama 3). This is a crucial design choice that ensures data privacy and eliminates external API costs.
- **Personalized Dashboard**: A private page accessible only to logged-in users, displaying a greeting (Hello, username) and a log of their last 7 days of quiz submissions. This serves as a central hub for tracking progress.
- **Gamification**: A "ChronoPoints" trophy-icon button introduces an element of gamification, encouraging user engagement and long-term retention.

---

2. Technical Stack & Architecture

The application is built on a modern, scalable stack, primarily centered around the **Django Model-View-Template (MVT)** architecture.

- **Backend**: **Django (Python)**
    - Handles all server-side logic, routing, and data processing.
    - Uses the requests library to make API calls to the Ollama server.
    - Manages user authentication with Django's built-in auth module.

- **Frontend**: **HTML & Tailwind CSS**
    - The user interface is built with clean HTML templates.
    - Styling is handled by the **Tailwind CSS framework**, which enables rapid and consistent UI development.
    - Custom CSS and JavaScript are used for dynamic animations and interactive elements.
- **Database**: **SQLite**
    - The project uses Django's default SQLite database for storing user accounts and quiz submissions.
- **AI Integration**: **Ollama & Llama 3**
    - Ollama acts as a local AI server, running a large language model like Llama 3. This approach is highly efficient for development and maintains data security.

---

3. Backend Implementation & Data Flow

The entire system's functionality is orchestrated through a clean and secure data flow.

1. **Form Submission**: The user fills out the crs_ai.html form and clicks "Get My CRS." The form, which includes a {% csrf_token %} for security, sends a **POST** request to the crs_ai view.
2. **View Processing**: The crs_ai function in views.py receives the request. It validates the data using the QuizForm and constructs a detailed **prompt_text** from all the user's answers.
3. **Ollama API Call**: Django then sends this prompt as a **JSON payload** in an HTTP POST request to the local Ollama server at http://localhost:11434/api/generate.
4. **Report Generation**: The Ollama server processes the prompt and returns a **JSON response** containing the AI's generated report.
5. **Database Storage & Rendering**: The views.py function extracts the report text from the JSON response, saves it to the QuizSubmission model, and then renders the crs_ai_results.html page, dynamically displaying the AI's feedback.

---

4. Key Development Challenges & Resolutions

- **API Connection Errors**: Initial attempts to connect to external APIs often resulted in 400 Bad Request or 401 Unauthorized errors. These were resolved by correctly formatting the request payload and headers, and by switching to a more reliable local Ollama backend.
- **Database Synchronization**: An OperationalError was fixed by running Django's makemigrations and migrate commands to update the database schema after adding new fields to the QuizSubmission model.
- **Frontend/Backend Integration**: A key challenge was ensuring that the frontend form correctly communicated with the backend view. This was solved by using Django's template tags and secure POST requests.

- **User Authentication**: A common issue of users being stuck on the login page was resolved by ensuring the views correctly handled authentication and redirection using the login() function and LOGIN_REDIRECT_URL.

Overall, this project is a robust, well-engineered application that provides a strong foundation for a valuable wellness tool.