**REPUBLIQUE DU CAMEROUN**

**PAIX – TRAVAIL – PATRIE**

**\*\*\*\*\*\*\*\*\*\***

**UNIVERSITE DE BAMENDA**

**\*\*\*\*\*\*\*\*\*\***

**FACULTE DES SCIENCES**

**\*\*\*\*\*\*\*\*\*\***

**Centre de cybersécurité et de cryptologie mathématique**

**\*\*\*\*\*\*\*\*\*\*\***

**REPUBLIC OF CAMEROON**

**PEACE – WORK – FATHERLAND**

**\*\*\*\*\*\*\*\*\*\***

**UNIVERSITY OF BAMENDA**

**\*\*\*\*\*\*\*\*\*\***

**FACULTY OF SCIENCE**

**\*\*\*\*\*\*\*\*\*\***

**Centre for Cybersecurity and Mathematical Cryptology**

**\*\*\*\*\*\*\*\*\*\*\***

**CYB6215 : APPLICATION DEVELOPMENT WITH PYTHON**

**HEALTH TRACKER PROJECT**

NGAFFO LABOU RAISSA FLORA **UBa24EP010**

OPTION:MASTERS IN CYBERSECURITY AND MATHEMATICALCRYPTOLOGY

**Lecturer:** **MR. EVRAD KAMTCHOUM**

# **Health Tracker Project Report**

**Project Name:** Health Tracker

**Technology Stack:** Django, Python, Bootstrap,

SQLite/PostgreSQL

**Date:** July 5, 2025

## **Executive Summary**

The Health Tracker is a modern, user-friendly Django web application designed to help users monitor and visualize their health metrics. The application provides comprehensive health tracking capabilities with an intuitive dashboard, advanced analytics, and personalized features.

### Key Achievements

* Fully functional health tracking system
* User-friendly interface with no complex roles or 2FA
* Advanced analytics and data visualization
* Export capabilities (CSV, PDF)
* Mobile-responsive design
* Security features implemented

## Project Objectives

### Primary Goals

1. Health Monitoring: Enable users to track daily health metrics
2. Data Visualization: Provide clear, interactive charts and analytics
3. User Experience: Create an intuitive, accessible interface
4. Data Management: Allow users to export and manage their health data
5. Goal Tracking: Help users set and monitor health goals

### Success Criteria

* Users can register and log in successfully
* Health records can be added and viewed
* Dashboard displays comprehensive analytics
* Data export functionality works
* Mobile-responsive design
* Security measures in place

## Features Implemented

### Core Features

1. User Authentication System

* User registration and login
* Password validation and security
* Session management

1. Health Record Tracking

* Sleep hours (0-24 hours)
* Water intake (liters)
* Weight tracking with BMI calculation
* Mood tracking (Excellent, Good, Neutral, Bad, Terrible)
* Goal setting for weight, sleep, and water intake

1. Dashboard & Analytics

* Interactive charts for all health metrics
* Weekly and monthly statistics
* Goal progress tracking
* Streak calculations
* Best/worst day identification

1. Data Export & Reports

* CSV export functionality
* PDF report generation
* Date range filtering

1. Notifications & Reminders

* Daily reminder system
* Health milestone notifications
* Customizable reminder settings

**SECURITY IMPLEMETATIONS**

**1. Authentication**

**Where:**

tracker/models.py (CustomUser model)

tracker/views.py (register, login, logout)

tracker/backends.py (custom backend)

settings.py (AUTH\_USER\_MODEL, AUTHENTICATION\_BACKENDS)

**How it helps:**Ensures only registered users can access their data and features. Supports login by username or email, and securely manages user credentials.**Example:**Only users who register and log in can add or view health records.**Prevents:**Unauthorized access, impersonation.

**2. Authorization**

**Where:**

@login\_required in tracker/views.py

role\_required in tracker/decorators.py

role field in CustomUser (models.py)

**How it helps:**Restricts access to sensitive pages and actions. Only authorized users (by login or role) can access certain features, protecting user data and admin functions.**Example:**Only logged-in users can access the dashboard or add health records. Admin-only features can be protected by role\_required.**Prevents:**Privilege escalation, unauthorized data access.

**3. Secure Session Management**

**Where:**

settings.py (SESSION\_COOKIE\_AGE, SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE)

Django’s session middleware

**How it helps:**Manages user sessions securely, controls session duration, and ensures sessions expire on browser close, reducing the risk of session hijacking.**Example:**If a user closes their browser, their session ends, so others can’t access their account.**Prevents:**Session hijacking, session fixation.

**4. Input Validation and Sanitization**

**Where:**

tracker/forms.py (form validation)

tracker/models.py (field validators)

**How it helps:**Prevents invalid or malicious data from being saved, protecting against attacks like SQL injection and ensuring data integrity.**Example:**HealthRecordForm ensures sleep hours are between 0 and 24, water intake is realistic, etc.**Prevents:**SQL injection, data corruption, XSS.

**5. CSRF Protection**

**Where:**

settings.py (django.middleware.csrf.CsrfViewMiddleware)

{% csrf\_token %} in all forms

**How it helps:**Prevents Cross-Site Request Forgery attacks, ensuring that only legitimate users can submit forms or perform actions.**Example:**All forms include {% csrf\_token %}, so attackers can’t forge requests.**Prevents:**CSRF attacks (e.g., tricking a user into submitting a form without their consent).

**6. Logging and Monitoring**

**Where:**

settings.py (LOGGING config)

/logs/ directory

**How it helps:**Records important events and errors, helping you detect suspicious activity, debug issues, and maintain an audit trail.**Example:**All login attempts, errors, and important actions are logged for review.**Prevents:**Undetected attacks, helps with incident response.

**7. Account Lockout and Rate Limiting**

**Where:**

tracker/backends.py (account lockout)

tracker/rate\_limit.py (rate limiting)

settings.py (RATE\_LIMIT\_DEFAULTS)

**How it helps:**Protects against brute-force attacks by locking accounts after repeated failed logins and limiting the rate of sensitive actions.**Example:**If a user enters the wrong password 5 times, their account is locked for 15 minutes.**Prevents:**Brute-force attacks, credential stuffing.

**8. Content Security Policy (CSP)**

**Where:**

settings.py (CSP settings)

csp.middleware.CSPMiddleware

**How it helps:**Prevents many types of attacks (like XSS) by restricting which scripts, styles, and resources can be loaded in the browser.**Example:**Only scripts and styles from trusted sources are allowed to run.**Prevents:**Cross-site scripting (XSS), malicious resource loading.

**9. Error Handling and Information Disclosure**

**Where:**

tracker/views.py (handler404, handler500)

health\_project/urls.py (custom error handlers)

Custom error templates

**How it helps:**Prevents sensitive information from being shown to users when errors occur, and provides user-friendly error pages.**Example:**If a user visits a non-existent page, they see a friendly 404 page, not a stack trace.**Prevents:**Information leakage, targeted attacks using error details.

**10. No Hard Coding of Secrets**

**Where:**

settings.py (uses os.getenv)

.env file

**How it helps:**Keeps sensitive information (like passwords and API keys) out of your codebase, reducing the risk of accidental leaks.**Example:**Database passwords and email credentials are stored in .env, not in code.**Prevents:**Secret/key exposure in version control or public repos.

**11. Password Security**

**Where:**

Django’s password hashing (default)

tracker/views.py (uses create\_user)

(Optionally, AUTH\_PASSWORD\_VALIDATORS in settings.py)

**How it helps:**Ensures user passwords are never stored in plain text, protecting users even if your database is compromised.**Example:**Even if someone accesses your database, they can’t read user passwords.**Prevents:**Password theft, credential compromise.

**12. Secure Dependencies**

**Where:**

requirements.txt

Use of virtual environment (venv/)

**How it helps:**Ensures you use only trusted, up-to-date packages, reducing the risk of vulnerabilities from third-party code.**Example:**All dependencies are listed and can be updated regularly.**Prevents:**Supply chain attacks, vulnerabilities from outdated packages.

**Not Fully Implemented (Deployment-Dependent)**

**Encryption in Transit (HTTPS):**

Needs to be set up in your production server (Nginx, Apache, or cloud provider).

Protects all data sent between users and your server from eavesdropping or tampering.

## Technical Architecture

### Technology Stack

* Backend: Django 5.2.3
* Frontend: HTML5, CSS3, JavaScript, Bootstrap
* Database: SQLite (development), PostgreSQL (production)
* Python Version: 3.12.7

## Challenges Faced

### Technical Challenges

1. **Dependency Management**

* Challenge: Missing xhtml2pdf package causing import errors
* Solution: Installed required dependencies and updated requirements.txt

1. **Authentication System**

* Challenge: Initially implemented complex 2FA system that was not user-friendly
* Solution: Simplified to basic authentication for better user experience

1. **Content Security Policy**

* Challenge: CSP configuration errors with django-csp package
* Solution: Updated to new CSP format and configured proper security headers

1. **Database Migrations**

* Challenge: Migration conflicts and missing dependencies
* Solution: Resolved by installing missing packages and running proper migrations

1. **Import Errors**

* Challenge: UnboundLocalError with timedelta import in views
* Solution: Removed redundant imports and fixed import structure

## **Deployment & Maintenance**

### Development Environment

* Local Development: Django development server
* Database: SQLite for development and postgres for production

### Production Readiness

* Database: PostgreSQL support
* Web Server: Gunicorn configuration
* Static Files: Collected and served
* Backup System: Automated backup scripts

## Conclusion

The Health Tracker project has successfully delivered a comprehensive, user-friendly health monitoring application. The system provides robust health tracking capabilities with advanced analytics, while maintaining simplicity and accessibility for end users.