# Structure of the project

# Architecture & Frontend Patterns (Midas)

The RFID Tracker (codename **Midas**) follows a traditional Django structure but uses a “Single File Component” philosophy for its views.

# Project Organization

The project is split into the core configuration and the functional app logic.

| Directory / File | Role |
| --- | --- |
| `midas/` | The active module containing the latest logic, models, and views. |
| `webui/` | Legacy code kept for bookkeeping; **do not use** for new features. |
| `templates/midas/` | The “Face” of the app. Contains HTML layouts, CSS, and JS. |
| `models.py` | Database schema (RFID tags, teams, hours). |
| `views.py` | The “Controller.” Fetches data from DB and performs calculations. |
| `urls.py` | Maps URLs to specific Python functions in `views.py`. |

---

# The Frontend Pattern

## Layout Strategy

Every page follows this block-based hierarchy:

1. `{% extends 'midas/base.html' %}`: Inherits the sidebar, navbar, and global styles.
2. `{% block head %}`: Contains page-specific CSS and external libraries (like Chart.js).

3. `{% block main %}`: The actual HTML content (cards, forms, tables).
4. `<script>`: Local logic for charts or UI interactions.

# The “Data Bridge” (Python to JavaScript)

To keep the code clean and secure, we use a specific pattern to pass data from the Python backend to the JavaScript frontend.

## The `json_script` Filter

Instead of messy string concatenation, we use Django's `json_script` tag. This safely injects Python dictionaries into the HTML as a JSON object that JS can read.

**In the HTML Template:**
`{{ page_data|json_script:"frontend-data" }}`

**In the JavaScript block:**
`const appData = JSON.parse(document.getElementById('frontend-data').textContent);`
`// Now appData.chart.labels is ready for Chart.js!`

# Backend Logic & Calculation

The heavy lifting is done in the Python files before the page even loads:

- **Calculations:** Logic like “Total Hours” or “Avg per Member” is calculated in `views.py` or helper files like `statistics.py`.
- **Database Interaction:** Uses Django’s ORM (Object-Relational Mapper) to query the `db.sqlite3` (dev) or the production DB.
- **Forms:** Django’s `forms.py` handles input validation when you change dates or teams in the dashboard.

# Admin dashboard

Django provides its own built-in admin dashboard which can be reached through “Admin Dashboard” button on the sidebar. (if you are a superuser).

---

Revision #4
Created 2026-04-15 11:07:07 UTC by Illia Guzerya
Updated 2026-04-16 09:03:28 UTC by Dmytro Khorsun