Software Requirements Specification (SRS) for EcoSpark

1. Introduction

1.1 Purpose

This document specifies the software requirements for EcoSpark — a web application to help users find certified e-waste recycling centers, learn about safe disposal, earn credits for recycling, and track eco-impact. The target audience is course instructors, maintainers, and future developers.

1.2 Scope

EcoSpark is a Django-based web application providing:

- A searchable map and listings of e-waste recycling centers
- A pickup scheduling form saved to admin
- AI-driven education modules (optional)
- A challenge/rewards system configurable via Django admin
- User accounts and per-user progress tracking

1.3 Definitions, acronyms and abbreviations

- AI: Artificial Intelligence
- SRS: Software Requirements Specification

2. Overall Description

2.1 Product perspective

EcoSpark is a monolithic Django project whose primary modules live in the `core` app. It uses Bootstrap 5 for UI and provides static assets under `core/static/`.

2.2 User classes and characteristics

- Anonymous user: can browse content, view centers, and complete challenges stored in session
- Authenticated user: can persist challenge completions to the database and schedule pickups
- Admin: manages `Challenge`, `Pickup`, `ChallengeCompletion`, and `RecyclingCenter` via Django admin

2.3 Operating environment

- Development: SQLite, Django development server
- Recommended production: PostgreSQL, collectstatic, secure cookies, and environment variables for secrets

3. System Features

3.1 Pickup Scheduling

- Users submit pickup requests (name, email, phone, address, waste type, pickup date/time).
- Requests are stored in the `Pickup` model and visible to admin.
- Confirmation message is displayed immediately (AI quote removed for responsiveness).

3.2 Challenges and Progress

- Challenges are defined by the `Challenge` model and editable via admin.
- Authenticated users have completions stored in `ChallengeCompletion` (unique per user & challenge).
- Anonymous users store completions in `request.session['challenges_completed']` as fallback.
- When a user logs in, session completions are merged into DB.

3.3 Home Page and Static Assets

- Home page uses local SVG icons for features and larger image assets for hero/cards/carousel.
- Small icons load eagerly (no lazy blur) to avoid rendering artifacts.

4. External Interfaces

4.1 Web Browser

- Supported: modern Chromium-based browsers, Firefox, Safari
- Responsive design via Bootstrap 5

4.2 Admin Interface

- Uses Django admin to manage models: `Pickup`, `Challenge`, `ChallengeCompletion`, `RecyclingCenter`.

5. Nonfunctional Requirements

- Performance: Pickup confirmation should be instant (no blocking AI calls).
- Security: CSRF for forms, avoid committing secrets; use environment variables for API keys.
- Reliability: Host key images locally rather than hotlinking to external sites.

6. Data Models

- `Pickup`: name, email, phone, address, waste_type, drive_type, pickup_date, pickup_time, created_at
- `Challenge`: title, co2_saved, is_active, order, created_at
- `ChallengeCompletion`: user FK, challenge FK, completed_at; unique constraint (user, challenge)

7. Migration and Persistence Behavior

- A DB migration was added for `ChallengeCompletion` and applied.
- Session keys for anonymous challenge completions use IDs formatted as `ch{db_id}`; retain format to ensure compatibility.

8. Deployment Notes

- Collect static files and serve them from a static host (CDN or cloud storage) for production.
- Configure `SECRET_KEY` and third-party credentials via environment variables.

9. Appendix: Next Steps

- Replace hotlinked hero/card/carousel images with local static copies (assistant can download and update templates).
- Add unit tests for challenge persistence and session→DB merge.
- Add CI steps to run tests and build assets.

Generated from in-conversation SRS summary on 2025-11-17.