Solar PV Feasibility – Application Documentation

1. Overview

Solar PV Feasibility is a web application that estimates **solar PV production**, **savings**, and **payback** for a given location. It uses:

- **PVGIS** (European Commission JRC) for irradiance and PV yield data.
- A **3-step wizard** (location → PV system → economics) to collect user inputs.
- A **Node/Express backend** that validates inputs (Zod), calls PVGIS, applies optional shading factors, and computes financial metrics.
- **Bilingual UI** (English and Romanian) and an optional **map + geocoding** (Nominatim) for location selection.

---

2. High-Level Flow

```
User opens app (/) → Wizard (3 steps) → Submit → POST /api/simulate
       → Backend: validate (Zod) → PVGIS (PVcalc + optional horizon)
       → Apply area-type shading → KPIs + finance + charts + insights
       → Store result → Navigate to /results → Show KPIs, charts, insights
```

- **Step 1 – Location:** Area type (rural/suburban/urban) and coordinates (map or search). Required to proceed.
- **Step 2 – PV system:** System size (kWp), losses %, horizon shading, optimal/fixed tilt & azimuth, technology, mounting, optional radiation database.
- **Step 3 – Economics:** CAPEX, buy/sell prices, self-consumption %, OPEX, degradation, analysis years, discount rate, price escalation.

On **Run simulation**, the client sends one request to `POST /api/simulate`. The server validates, calls PVGIS, applies shading and finance logic, and returns one result object. The client stores it and navigates to `/results`.

---

3. Inputs (What the App Collects)

All inputs are defined in `server/src/schema.ts` (Zod) and collected by the wizard and sent as the request body to the API.

3.1 Location

Field	Type / constraints	Description
`lat`	number, -90..90	Latitude
`lon`	number, -180..180	Longitude
`area_type`	`'rural'` | `'suburban'` | `'urban'` or null	Optional; used for shading

3.2 PV System

Field	Type / constraints	Description
`peakpower_kw`	number, (0, 1000]	System size in kWp
`loss_percent`	number, 0..80, default 14	System losses (%)

`usehorizon`	boolean, default true	Use terrain horizon (PVGIS printhorizon)
`optimalangles`	boolean, default true	If false, use manual angle/azimuth
`angle_deg`	number 0..90 or null	Tilt angle (used when not optimal)
`aspect_deg`	number -180..180 or null	Azimuth, 0=South (used when not optimal)
`pvtechchoice`	enum: crystSi, crystSi2025, CIS, CdTe, Unknown	PV technology
`mountingplace`	`'free' | 'building'`	Mounting type
`raddatabase`	string or null	Optional radiation DB (e.g. PVGIS-SARAH3, etc.)

3.3 Economics

Field	Type / constraints	Description
`capex`	number > 0	System cost (CAPEX)
`price_buy`	number > 0	Electricity price (buy)
`self_consumption`	number 0..1, default 0.5	Fraction self-consumed
`price_sell`	number ≥ 0, default 0	Export price
`opex_yearly`	number ≥ 0, default 0	Yearly OPEX
`degradation`	number 0..0.1, default 0.005	Degradation per year (e.g. 0.5%)
`analysis_years`	integer 1..40, default 25	Analysis period in years
`discount_rate`	number 0..1, default 0.06	Discount rate for NPV
`price_escalation`	number 0..1, default 0	Yearly electricity price escalation

---

4. Outputs (What the API Returns)

The API returns a single JSON object (`SimulationResult` in `server/src/simulate.ts` , `SimulateResponse` in `client/src/app/simulate.service.ts`).

4.1 Structure

Section	Content
pvgis	`inputs` (echo from PVGIS), `monthly` (per-month energy E_m, SD_m, H_i_m), `totals` (E_y, SD_y, H_i_y, l_total, LCOE_pv)
kpis	`annual_kwh`, `specific_yield_kwh_per_kwp`, `capacity_factor_pct`, `best_month` / `worst_month`, `seasonality_ratio`, `uncertainty_annual_kwh`
finance	`savings_year1`, `payback_years`, `roi`, `npv`, `cashflow_yearly`, `cashflow_cumulative`

charts	`monthly_energy_kwh` (month, kwh), `cashflow_cumulative` (year, value) for bar charts
insights	Array of `{ type: 'info' | 'warning', text, key? }` (e.g. area shading, high losses, orientation)
meta	`area_type_applied` (rural/suburban/urban or null)

4.2 KPIs Explained

- **annual_kwh** – Estimated yearly PV production (kWh).
- **specific_yield_kwh_per_kwp** – kWh per kWp per year (system performance indicator).
- **capacity_factor_pct** – Annual energy / (peak power × 8760 h) as percentage.
- **best_month / worst_month** – Month index and energy for highest and lowest production.
- **seasonality_ratio** – Ratio best month / worst month (higher = more seasonal).
- **uncertainty_annual_kwh** – Standard deviation of annual yield (from PVGIS).

4.3 Finance Explained

- **savings_year1** – Net savings in year 1 (self-consumed value + export value − OPEX).
- **payback_years** – First year when cumulative cashflow ≥ 0 (null if never).
- **roi** – (Total savings − CAPEX) / CAPEX over analysis period.
- **npv** – Net present value of cashflows minus CAPEX (uses discount_rate).
- **cashflow_yearly** – Net savings per year (after degradation and optional price escalation).
- **cashflow_cumulative** – Cumulative cashflow (year 0 = −CAPEX, then year 1..N).

---

5. Architecture

5.1 Client (Angular)

- **Routes:** `''` → Wizard, `'results'` → Results, `'**'` → redirect to `''`.
- **Wizard** (`wizard.component.ts`): Three steps, signals for all form fields, calls `SimulateService.simulate(body)` on submit; on success stores result in `SimulationStoreService` and navigates to `/results`.
- **SimulationStoreService:** Holds `lastRequest`, `lastResponse`, `isLoading`, `errorMessage` (signals). Shared between wizard and results.
- **LocationPickerComponent:** Leaflet map, draggable marker, search via `GeocodingService` (Nominatim proxy at `/nominatim`), emits `lat` / `lon` to parent.
- **Results** (`results.component.ts`): Reads `store.lastResponse()`, displays KPIs, monthly production bar chart, cumulative cashflow bar chart, and insights (translated via `LanguageService`).
- **LanguageService:** EN/RO translations including insight keys; language persisted in `localStorage` under `solar-lang`.
- **Proxy:** `client/proxy.conf.json` forwards `/api` → `http://localhost:3000`, `/nominatim` → Nominatim API.

5.2 Server (Express)

- **POST /api/simulate:** Parses body with `SimulateSchema.safeParse(req.body)`. On success runs `runSimulation(parsed.data)` and returns JSON; on validation failure returns 400 with Zod error shape; on PVGIS or other errors returns 502 with a safe message.
- **GET /api/health:** Returns `{ ok: true }`.
- **simulate.ts:** Builds PV params (including horizon from `getHorizonProfile` when `usehorizon` is true), calls `callPVGIS('PVcalc', ...)`, parses monthly and totals from PVGIS response, applies

`AREA_TYPE_FACTOR` (rural 1.0, suburban 0.92, urban 0.85), then computes KPIs, finance, chart data, and insights.

- **pvgis.ts:** Rate-limited and cached (Bottleneck + LRUCache) HTTP client for PVGIS; `callPVGIS(tool, params, radiation_database)` and `getHorizonProfile(lat, lon)` (uses printhorizon → 48 elevation angles).

5.3 External Services

- **PVGIS** (re.jrc.ec.europa.eu): PVcalc for monthly/yearly yield; printhorizon for terrain horizon profile.
- **Nominatim** (OpenStreetMap): Geocoding search and reverse geocoding, proxied via dev server to avoid CORS.

---

6. End-to-End Data Flow

1. User selects location (and optional area type) and PV + economics in the wizard.
2. User clicks **Run simulation** → `SimulateService.simulate(body)` → `POST /api/simulate` with full input.
3. Server validates with `SimulateSchema`; if invalid, responds with 400 and Zod error structure.
4. Server runs `runSimulation(input)`:
   - Optionally fetches horizon profile (PVGIS printhorizon) and passes it to PVcalc.
   - Calls PVGIS PVcalc → receives monthly and yearly totals.
   - Applies area-type shading factor to monthly and yearly energy (and uncertainty).
   - Computes KPIs (annual energy, specific yield, capacity factor, best/worst month, seasonality, uncertainty).
   - Computes finance: yearly cashflow with degradation and optional price escalation, cumulative cashflow, payback year, ROI, NPV.
   - Builds chart data (monthly energy, cumulative cashflow including year 0 = −CAPEX) and insights (area shading, high losses, orientation, variability, high yield).
5. Server responds with the single `SimulationResult` JSON.
6. Client stores it in `SimulationStoreService.lastResponse` and navigates to `/results`.
7. Results page reads `lastResponse`, displays KPIs, bar charts, and translated insights.

---

7. Key Files Reference

Path	Purpose
`server/src/schema.ts`	Zod schema and `SimulateInput` type for API contract
`server/src/simulate.ts`	`runSimulation`, KPIs, finance, charts, insights
`server/src/pvgis.ts`	PVGIS API client, rate limit, cache, horizon profile
`server/src/index.ts`	Express app, POST /api/simulate, GET /api/health
`server/src/utils.ts`	`num()`, `getMonthlyArray()`, azimuth helper
`client/src/app/simulate.service.ts`	`SimulateRequest`/`SimulateResponse`, POST /api/simulate
`client/src/app/simulation-store.service.ts`	lastRequest, lastResponse, isLoading, errorMessage

`client/src/app/wizard/wizard.component.ts`	3-step wizard, form state, submit
`client/src/app/results/results.component.ts`	Results view, charts, insights
`client/src/app/location-picker/location-picker.component.ts`	Map, marker, search, lat/lon output
`client/src/app/geocoding.service.ts`	Nominatim search and reverse geocoding
`client/src/app/language.service.ts`	EN/RO translations
`client/proxy.conf.json`	/api → backend, /nominatim → Nominatim

---

8. Running the Application

- **Backend:** From `server/` : `npm run dev` (or `npm run build` then `npm start`). Listens on port 3000 by default (`PORT` env).
- **Frontend:** From `client/` : `ng serve` (or `npm start`). Uses proxy for `/api` and `/nominatim` ; typically at http://localhost:4200.
- Ensure the backend is running before running simulations; otherwise the client will show a network error.

Generated for Solar PV Feasibility (solar-licenta).