# Product Requirements Document: Reinforcement‑Learning–Driven Building Energy Optimisation

## 1. Executive Summary

**Problem.** Traditional Building Energy Management Systems (BEMS) rely on static control schedules or model‑based techniques that do not adapt to real‑time conditions. The source document highlights that existing control strategies are unable to handle fluctuating occupancy and weather; they also lack tools for building managers and perform poorly when system models are unknown. The absence of empirically validated reinforcement‑learning (RL) frameworks and user‑friendly interfaces limits practical adoption.

**Target users.** Primary users are **building managers** and **facility operators** responsible for HVAC systems. Secondary users include **energy analysts** comparing algorithms and **researchers** experimenting with control strategies. End‑users (occupants) influence comfort metrics but do not directly interact with the system.

**Goals.** Develop a Python/Streamlit application that trains RL models (Q‑Learning, Deep Q‑Network, Proximal Policy Optimisation) on the Building Data Genome Project 2 dataset and delivers control recommendations that minimise HVAC energy consumption while maintaining thermal comfort. The application must include preprocessing, feature engineering, model training, evaluation against non‑RL baselines (Random Forest, Support Vector Machines, Neural Networks), visualisation dashboards, and simulation tools.

**Non‑goals.** Physical deployment in real buildings; optimisation of non‑HVAC loads or integration with electricity pricing or renewable generation; support for unsupervised or hybrid ML techniques; and high‑fidelity building simulations beyond the dataset scope.

**North Star.** *“Empower building managers with an interactive, RL‑driven tool that continuously learns energy‑optimal HVAC control strategies from real data while maintaining occupant comfort.”*

**Success metrics.**

| Metric | Target (p95) | Rationale |
| --- | --- | --- |
| **Energy savings** | ≥ 20 % reduction compared with a fixed rule‑based policy | Demonstrates RL advantage over static schedules. |
| **Thermal comfort maintenance** | ≥ 90 % of occupied hours within 20–24 °C comfort band | Ensures occupant satisfaction. |
| **Prediction latency** | ≤ 500 ms for single inference; ≤ 2 s for batch analysis | Meets responsiveness expectations for edge deployment. |
| **Model convergence** | RL models converge within 1,000 training episodes with moving average cumulative reward stable | Validates training efficiency. |
| **User engagement** | ≥ 80 % of building managers who run at least one simulation scenario per month (industry standard) | Indicates adoption and utility of the interface. |

## 2. Scope & Out‑of‑Scope

### In‑scope

* Acquisition of BDGP2 dataset and automated preprocessing: cleaning missing values, outlier capping, normalisation, and feature engineering.
* Development of RL models (Q‑Learning, DQN, PPO) to optimise HVAC control policies.
* Implementation of non‑RL baseline models (Random Forest, SVM, ANN) for benchmarking.
* Simulation environment replicating building dynamics using recorded data (Markov Decision Process) and reward function balancing energy and comfort.
* User interface for data ingestion, parameter configuration, training, evaluation, dashboards, and scenario simulation.
* System architecture with modular components for ingestion, training, evaluation, and visualisation.
* Ethical guidelines: anonymisation, local computation, transparency, sustainability.

### Out‑of‑scope

* Real‑time control of HVAC systems in physical buildings (no hardware integration).
* Optimisation of lighting, plug loads, or renewable integration beyond HVAC.
* Support for reinforcement learning algorithms other than Q‑Learning, DQN, and PPO (assumption).
* Integration with third‑party building management systems or vendor APIs (assumption).
* Consideration of electricity tariffs or dynamic pricing (scope explicitly excludes pricing).

## 3. Users & Use Cases

### Personas

| Persona | Role & capabilities | Constraints |
| --- | --- | --- |
| **Building Manager (Primary)** | Oversees HVAC operations, monitors energy bills, adjusts comfort preferences. Needs simple UI to run simulations and adopt recommendations. | Limited technical expertise; must ensure occupant satisfaction and comply with regulations. |
| **Facility Operator** | Maintains mechanical systems, may implement recommended setpoints manually. | Works on-site; safety and reliability are paramount. |
| **Energy Analyst** | Compares algorithmic performance, tunes hyperparameters, and reports outcomes to stakeholders. | Requires access to detailed logs and metrics; may have data science expertise. |
| **Researcher** | Experiments with new RL algorithms and publishes results. | Requires reproducibility, access to dataset and training pipelines; may modify code (industry standard). |
| **Occupant (Indirect)** | Experiences thermal comfort; influences occupancy patterns. | No direct system access; comfort thresholds must be respected. |

### Use‑case matrix (MoSCoW)

| Persona | Goal | Frequency | Priority |
| --- | --- | --- | --- |
| Building Manager | View current energy consumption and comfort metrics | Daily | Must |
| Building Manager | Run simulation scenarios adjusting comfort ranges or RL model parameters | Weekly | Must |
| Building Manager | View RL recommendations and apply them manually | Daily | Should |
| Facility Operator | Retrieve recommended setpoints and schedule maintenance | Weekly | Should |
| Energy Analyst | Train RL models and benchmark against baseline models | Monthly | Must |
| Energy Analyst | Perform hyperparameter tuning and evaluate convergence | Monthly | Should |
| Researcher | Extend RL algorithms or reward functions | As needed | Could |
| Occupant | Provide feedback on thermal comfort (e.g., via surveys) | Quarterly | Could |

### Roles & permissions

| Role | Dataset (CRUD) | Model (CRUD) | Simulation & Reports | Administration |
| --- | --- | --- | --- | --- |
| Building Manager | R (read) | C,R (create models, view) | C,R,D (create scenarios, read dashboards, delete own simulations) | — |
| Facility Operator | R | R | R | — |
| Energy Analyst | R | C,R,U,D (full control) | C,R,U,D | — |
| Researcher | R | C,R,U (no delete) | C,R,U | — |
| Admin (assumption) | C,R,U,D | C,R,U,D | C,R,U,D | Manage users, system settings |

## 4. Functional Requirements

For each feature, requirements reference source‑doc lines or are marked **(assumption)** or **(industry standard)**. Telemetry events use names like event\_name with properties.

### FR‑001 – Data Acquisition & Preprocessing

* **Description.** Load the BDGP2 dataset from local storage or user upload; apply cleaning, imputation, outlier capping, normalisation, and feature engineering.
* **Rationale.** Ensures data quality and consistent input for model training.
* **Preconditions.** Dataset accessible; user has rights to upload or select file.
* **Dependencies.** None.
* **Detailed behaviour.**
* On user action to upload/select dataset, the system ingests time‑series data (energy, temperature, humidity, occupancy, metadata).
* For gaps ≤3 hours, apply linear interpolation; for longer gaps, forward fill.
* Identify outliers using IQR; cap at 99th percentile.
* Remove duplicates and misaligned timestamps.
* Scale numeric features to [0,1] using min‑max scaling.
* One‑hot encode categorical features (building type, climate zone) (assumption).
* Generate engineered features: temporal sine/cosine encodings, occupancy status, thermal comfort deviation, rolling averages.
* Split chronologically: training (2016), validation (Jan–Jun 2017), test (Jul 2017–Jan 2018).
* **Data I/O.** Input: CSV/Parquet dataset with columns (timestamp, meter\_reading, temp\_in, temp\_out, humidity, occupancy, etc.). Output: cleaned Pandas DataFrame; features vector per timestamp; metadata dictionary.
* **Acceptance criteria.**
* **GIVEN** a dataset with missing values **WHEN** preprocessing runs **THEN** gaps ≤3 hours are linearly interpolated and longer gaps forward filled.
* **GIVEN** outlier values **WHEN** preprocessing executes **THEN** values above the 99th percentile are capped.
* **GIVEN** raw features **WHEN** feature engineering executes **THEN** the resulting DataFrame contains temporal features (sine/cosine), occupancy status, thermal comfort deviation, rolling averages.
* **GIVEN** a dataset **WHEN** splitting **THEN** training/validation/test sets are non‑overlapping and ordered chronologically.
* **GIVEN** multiple runs **WHEN** preprocessing is repeated **THEN** results are reproducible (same random seed, same outputs) (industry standard).
* **Telemetry.** data\_preprocessed with properties: dataset\_id, records\_processed, interpolation\_method, outliers\_capped\_count.
* **Open questions.** Should the application allow users to choose different imputation strategies (user‑definable)?

### FR‑002 – Reinforcement Learning Model Training

* **Description.** Train RL models (Q‑Learning, DQN, PPO) in a simulated environment to learn HVAC control policies.
* **Rationale.** RL enables adaptive control without explicit system models.
* **Preconditions.** Preprocessed training and validation data are available; simulation environment is initialised; hyperparameter defaults provided.
* **Dependencies.** FR‑001.
* **Detailed behaviour.**
* The environment formulates states using engineered features and actions representing thermostat setpoint adjustments (increase/decrease/maintain for Q‑Learning & DQN; continuous adjustments for PPO).
* Define reward function penalising energy use and comfort deviations in the 20–24 °C band.
* Q‑Learning: discrete Q‑table with learning rate α, discount factor γ, exploration rate ε; update Q(s,a) via the Bellman equation.
* DQN: neural network approximates Q-values; uses experience replay and target network; architecture comprises three fully connected layers with ReLU activation. Explore with ε‑greedy strategy; update target network periodically.
* PPO: actor‑critic network (two fully connected layers with GELU); continuous action space; clipped probability ratio to ensure stable updates.
* Hyperparameter tuning via grid search: Q‑Learning (α∈{0.1,0.2,0.3}, γ∈{0.8,0.9,0.95}, ε∈{0.1,0.2,0.3}); DQN (hidden layers 1–3, sizes {64,128,256}, activation functions {ReLU,GELU}, learning rates {1e−4,5e−4,1e−3}, batch sizes {32,64,128}); PPO (learning rate {1e−4,3e−4,5e−4}, clip range {0.1,0.2,0.3}, epochs {4,10,20}, batch sizes {64,128})【687278560629580†L2749-L2792】.
* Train each model for up to 1,000 episodes (each episode representing 24 hours of building operation), repeating with different random seeds, and monitor moving average cumulative reward for convergence.
* Store trained model parameters for deployment.
* **Data I/O.** Input: training/validation sets; hyperparameter configuration. Output: trained model file (pickle, TensorFlow or PyTorch weights), training logs (rewards, losses).
* **Acceptance criteria.**
* **GIVEN** preprocessed data and hyperparameters **WHEN** RL training runs **THEN** models train without runtime errors and produce reward trajectories.
* **GIVEN** three different random seeds **WHEN** training each model **THEN** the average cumulative reward is reported across runs.
* **GIVEN** DQN training **WHEN** training runs **THEN** experience replay and target network updates occur (industry standard).
* **GIVEN** PPO training **WHEN** training runs **THEN** clipped updates ensure the loss does not diverge.
* **GIVEN** models **WHEN** tuning completes **THEN** best hyperparameter sets are stored along with performance metrics【687278560629580†L2749-L2792】.
* **Telemetry.** model\_training\_started (model\_type, hyperparams); model\_training\_finished (model\_type, episodes, convergence\_status, best\_reward).
* **Open questions.** Should hyperparameter search be customisable by users (e.g., random search)? What hardware acceleration (CPU/GPU) is required? (assumptions).

### FR‑003 – Baseline Model Training

* **Description.** Train non‑RL machine learning models (Random Forest, SVM, ANN) for benchmarking.
* **Rationale.** Provides a comparative framework to assess RL advantages.
* **Preconditions.** Preprocessed data available.
* **Dependencies.** FR‑001.
* **Detailed behaviour.**
* Random Forest: train ensemble of decision trees using engineered features; hyperparameters include number of trees (100, 200) and max depth (None, 10, 20) (assumed values).
* SVM: multi‑class classifier with RBF kernel; tune C and gamma parameters; uses the same feature set.
* ANN: feedforward network with three layers (sizes 64, 128, 64; ReLU) and dropout; trained with Adam optimiser and categorical cross‑entropy loss.
* Train on 2016 data; validate on Jan–Jun 2017; test on Jul 2017–Jan 2018.
* **Data I/O.** Input: labelled dataset (state features, rule‑based actions). Output: trained model files, metrics (accuracy, precision, recall, F1‑score).
* **Acceptance criteria.**
* **GIVEN** preprocessed data **WHEN** baseline training runs **THEN** models train without errors.
* **GIVEN** test set **WHEN** evaluation runs **THEN** accuracy, precision, recall, and F1‑score are computed.
* **GIVEN** baseline models **WHEN** comparing with RL models **THEN** results appear in evaluation dashboard.
* **Telemetry.** baseline\_training\_finished (model\_type, accuracy, precision, recall, f1\_score).
* **Open questions.** Should baseline training support additional algorithms (e.g., k‑NN)? (assumption).

### FR‑004 – Simulation Environment & Reward Function

* **Description.** Provide a simulation environment that emulates building thermal dynamics and occupancy using historical data; implement a reward function balancing energy consumption and thermal comfort.
* **Rationale.** Simulated environment enables safe RL training and evaluation without deploying to physical systems.
* **Preconditions.** Preprocessed dataset available.
* **Dependencies.** FR‑001.
* **Detailed behaviour.**
* Represent environment as a Markov Decision Process with states (indoor temperature, outdoor temperature, humidity, occupancy status, time of day, engineered features), actions (setpoint adjustments), reward (negative energy use plus penalty for comfort deviations).
* At each time step, sample next state from historical record; apply action; compute energy consumption difference and comfort deviation; return reward and new state.
* For baseline models, environment still provides states and expects predicted discrete actions for evaluation.
* **Data I/O.** Input: current state and chosen action. Output: next state, reward, done flag (episode ends after 24 hours).
* **Acceptance criteria.**
* **GIVEN** a state vector **WHEN** an action is applied **THEN** the environment returns a next state drawn from the dataset.
* **GIVEN** an action that increases energy consumption outside the comfort range **WHEN** reward is computed **THEN** the reward is negative with magnitude proportional to energy and comfort deviation.
* **GIVEN** an action that maintains comfort within 20–24 °C **WHEN** reward is computed **THEN** the reward includes a positive component for comfort maintenance.
* **GIVEN** the end of a 24‑hour period **WHEN** environment step executes **THEN** the done flag is true and episode resets.
* **Telemetry.** environment\_step (state\_hash, action, reward, next\_state\_hash, done).
* **Open questions.** Should the environment incorporate stochastic noise or energy price signals? (industry standard: optional).

### FR‑005 – Model Evaluation & Metrics

* **Description.** Compute performance metrics for RL and baseline models, including energy savings, thermal comfort maintenance, accuracy, cumulative reward, and convergence【687278560629580†L2893-L2944】.
* **Rationale.** Enables objective comparison and ensures models meet goals.
* **Preconditions.** Trained models and test set available.
* **Dependencies.** FR‑002, FR‑003.
* **Detailed behaviour.**
* For RL models: compute total energy consumption under learned policy and baseline rule‑based policy; calculate percentage reduction .
* Evaluate thermal comfort as percentage of hours within 20–24 °C and Predicted Mean Vote (PMV) index.
* Record cumulative reward over test episodes; plot moving average to assess convergence.
* For baseline models: compute classification metrics (accuracy, precision, recall, F1‑score).
* Present results side‑by‑side in dashboards.
* **Data I/O.** Input: test set, trained models. Output: metrics dictionary, plots (matplotlib) for rewards and energy trajectories.
* **Acceptance criteria.**
* **GIVEN** a trained RL model and test data **WHEN** evaluation runs **THEN** energy savings are computed relative to rule‑based baseline and reported.
* **GIVEN** RL results **WHEN** thermal comfort is computed **THEN** the proportion of hours within comfort band is ≥ target threshold (as per success metric) if model meets goals.
* **GIVEN** baseline models **WHEN** evaluation runs **THEN** classification metrics appear.
* **GIVEN** evaluation output **WHEN** displayed in UI **THEN** metrics and plots are clear and interactive.
* **Telemetry.** evaluation\_completed (model\_type, energy\_savings, comfort\_pct, accuracy, f1\_score, cumulative\_reward).
* **Open questions.** Should additional metrics such as HVAC equipment cycling frequency be included? (assumption).

### FR‑006 – Visualisation & User Interface

* **Description.** Provide a Streamlit‑based UI with pages for dataset upload, preprocessing status, model training configuration, progress, evaluation dashboards, and scenario simulation.
* **Rationale.** Enables non‑technical users to interact with the system and interpret results.
* **Preconditions.** User is authenticated (if required) and has necessary permissions.
* **Dependencies.** FR‑001 to FR‑005.
* **Detailed behaviour.**
* Home page summarises the project and allows selection of dataset.
* Preprocessing page displays data quality reports, missing value counts, and feature distributions; includes progress bars and logs.
* Training page allows users to select model type (Q‑Learning, DQN, PPO, or baseline) and configure hyperparameters; training progress is shown via charts (episodes vs reward).
* Evaluation page presents performance metrics and charts side‑by‑side; allows comparison across models; provides export to CSV.
* Simulation page enables building managers to adjust comfort preferences, run what‑if scenarios (e.g., adjusting comfort band or energy price), and view predicted energy use and comfort outcomes.
* Navigation via sidebar; each page uses descriptive headings and tooltips; interactive components such as sliders, dropdowns, and toggles.
* Provide accessibility features: keyboard navigation, ARIA labels, high contrast mode, and screen‑reader friendly tables (industry standard).
* **Data I/O.** Input: user interactions. Output: dynamic UI components, charts (matplotlib), tables (Streamlit DataFrame), downloadable files.
* **Acceptance criteria.**
* **GIVEN** an uploaded dataset **WHEN** viewing the preprocessing page **THEN** the system shows missing values, distribution plots, and allows user to proceed to training.
* **GIVEN** the user selects a model and hyperparameters **WHEN** clicking “Train” **THEN** progress indicators display training status and allow cancellation.
* **GIVEN** evaluation metrics **WHEN** the evaluation page is loaded **THEN** metrics are presented clearly and can be sorted or filtered.
* **GIVEN** comfort preferences changes in the simulation page **WHEN** the user runs simulation **THEN** predicted energy consumption and comfort outcomes update in real time (within specified latency).
* **GIVEN** keyboard navigation **WHEN** tabbing through components **THEN** focus order is logical and all elements are accessible (industry standard).
* **Telemetry.** page\_view (page\_name); simulation\_run (parameters, outputs); training\_cancelled; download\_export.
* **Open questions.** Should dark mode or mobile responsiveness be included? (assumption).

### FR‑007 – System Architecture & Integration

* **Description.** Implement a modular architecture with data ingestion, model training, evaluation, and visualisation modules; support optional deployment on edge devices and local compute.
* **Rationale.** Facilitates scalability, reproducibility, and potential integration with real BEMS.
* **Preconditions.** None.
* **Dependencies.** FR‑001 to FR‑006.
* **Detailed behaviour.**
* Data ingestion module manages reading from file system, data cleaning, feature engineering, and dataset partitioning.
* Training module encapsulates RL algorithms and baseline models; uses standard libraries (Stable-Baselines3 for DQN and PPO; custom Q‑Learning; Scikit‑learn for baselines).
* Evaluation module computes metrics and logs predictions.
* Visualisation module generates dashboards and simulation pages.
* Architecture follows three‑tier design: IoT layer (sensors, simulated data), edge layer (local processing/inference), cloud layer (data storage and training). Communication is asynchronous and secure.
* Provide RESTful internal APIs for other components (e.g., training service accepts dataset ID, returns job status; evaluation service returns metrics).
* **Data I/O.** Input: dataset file paths, API calls. Output: data frames, model states, metrics, JSON responses.
* **Acceptance criteria.**
* **GIVEN** the modular codebase **WHEN** a module fails (e.g., training fails) **THEN** other modules continue to function independently (fault tolerance).
* **GIVEN** asynchronous requests **WHEN** training or evaluation runs in the background **THEN** the UI remains responsive and displays job status (industry standard).
* **GIVEN** a REST API request **WHEN** retrieving evaluation metrics **THEN** the JSON response includes energy\_savings, comfort\_pct, reward\_trajectory, and classification metrics.
* **Telemetry.** module\_error (module\_name, error\_type); api\_call (endpoint, latency, status\_code).
* **Open questions.** Should the architecture integrate with message queues or microservices (e.g., RabbitMQ) for scaling? (assumption).

### FR‑008 – Ethical & Privacy Controls

* **Description.** Ensure that personal data is not exposed; support anonymisation, local processing, and transparency in model behaviour.
* **Rationale.** The dataset is anonymised and aggregated; privacy must be preserved.
* **Preconditions.** Data contains no personally identifiable information (PII).
* **Dependencies.** FR‑001, FR‑006.
* **Detailed behaviour.**
* Confirm that dataset only contains building‑level or zone‑level data; no occupant identities.
* Perform all computation locally or on edge servers; avoid sending sensitive data to third‑party services.
* Provide transparent documentation of model architecture, hyperparameters, and training procedures.
* Use reproducible, open‑source libraries; publish evaluation results and code (industry standard).
* **Data I/O.** Input: anonymised dataset. Output: models and metrics with no PII.
* **Acceptance criteria.**
* **GIVEN** dataset metadata **WHEN** loading data **THEN** no fields contain names, addresses, or individual identifiers.
* **GIVEN** data processing **WHEN** performing computations **THEN** data never leaves the local environment.
* **GIVEN** model outputs **WHEN** reporting results **THEN** aggregated and anonymised metrics are used.
* **Telemetry.** privacy\_check\_passed (dataset\_id, checks\_run).
* **Open questions.** Should the system support GDPR‑like data retention policies or allow data deletion on request? (assumption).

## 5. Information Architecture & Navigation

### Sitemap

* **Home** – Project overview, dataset selection.
* **Preprocessing** – Data quality report and cleaning operations.
* **Training** – Configure and launch model training; view progress.
* **Evaluation** – Compare metrics across RL and baseline models; export results.
* **Simulation** – Adjust comfort preferences and run “what‑if” scenarios.
* **About / Ethics** – Display privacy, sustainability, and methodology information.
* **Admin** (hidden if not admin) – Manage users and roles (assumption).

### Navigation rules

* Use Streamlit’s sidebar for navigation; clicking a menu item loads the corresponding page while preserving session state for unsaved work.
* Deep links include query parameters like ?page=training&model=DQN to preselect pages and options.
* Browser back/forward controls work within Streamlit session by updating URL parameters; the UI must listen to st.experimental\_get\_query\_params() and update state accordingly.
* Modal dialogs for confirming deletion of simulation runs or resetting hyperparameters; modals trap focus until dismissed (accessibility).

## 6. UX & Content Specifications

### Screen specifications

Each screen uses Streamlit’s container or columns for layout. Avoid horizontal scrolling. Provide clear labels, tooltips, and icons only when necessary.

#### Home

* **Purpose:** Introduce the project, show a summary of dataset (records count, date range), and provide file upload/select button.
* **Primary actions:** Upload dataset (File Uploader), select sample dataset.
* **Components:** Hero section with tagline; dataset selection section; call‑to‑action button leading to Preprocessing page.
* **Layout:** Two columns: left with description, right with file upload.

#### Preprocessing

* **Purpose:** Show data quality summary and allow user to clean and engineer features.
* **Components:** Progress bar; table summarising missing values per feature; distribution plots; button to start preprocessing; logs area.
* **Layout:** Vertical stacking with collapsible sections for cleaning steps.
* **Empty state:** Show message “No dataset loaded” with instructions.
* **Loading state:** Indeterminate progress bar with text “Preprocessing data…”.
* **Error copy:** Show error message if file format invalid or cleaning fails.

#### Training

* **Purpose:** Configure model type and hyperparameters; monitor training progress.
* **Components:** Dropdown for model selection; hyperparameter fields; Start/Cancel buttons; progress chart; log panel.
* **Layout:** Form on top; progress section below.
* **Long‑running tasks:** Use st.progress and st.status to show training status; disable inputs during training.
* **Error handling:** Show messages for hyperparameter validation errors.

#### Evaluation

* **Purpose:** Present metrics and compare models.
* **Components:** Multi‑select to choose models; metrics table; line charts for cumulative reward and energy use; bar charts for comfort and accuracy; export button.
* **Layout:** Metrics table on top; charts in tabs below.

#### Simulation

* **Purpose:** Let building managers adjust comfort range or occupancy scenarios and see predicted outcomes.
* **Components:** Sliders for comfort band (e.g., min/max temperature); date pickers; run button; line chart showing predicted energy consumption and comfort; summary metrics.
* **Loading state:** Show spinner during simulation; update results once complete.
* **Accessibility:** All controls support keyboard navigation; use ARIA labels and alt text for charts.

#### About / Ethics

* **Purpose:** Explain methodology, ethical considerations, and privacy practices.
* **Components:** Text blocks, lists, and citations; link to research paper.

### Content guidelines

* Use concise, jargon‑free language. Avoid abbreviations unless defined.
* For numbers, use units (kWh, °C) and adopt user’s locale formatting (Africa/Lagos timezone). Use comma as thousands separator and period as decimal point (industry standard).
* Provide contextual hints (e.g., “Energy savings compare RL policy to rule‑based setpoints at 22 °C during occupied hours”).
* Use charts to illustrate trends; never encode long text in tables.
* Provide alt text describing charts (e.g., “Line chart showing cumulative reward over episodes”).

### Accessibility (WCAG 2.1 AA)

* All interactive elements reachable via keyboard and have focus indicators.
* Use appropriate aria-label attributes for buttons and sliders.
* Ensure colour contrast ratios of at least 4.5:1 (Streamlit default theme may need override).
* Provide descriptive headings (h1, h2, etc.), skip links, and semantic HTML via Streamlit components.
* Support screen‑reader friendly tables by using st.dataframe with use\_container\_width=True.

## 7. Data Model & Storage

### Entities & attributes

| Entity | Attributes | Relationships |
| --- | --- | --- |
| **Dataset** | dataset\_id (UUID), file\_path (string), created\_at (datetime), records\_count (int), features (list of strings), status (enum: raw, cleaned, processed) | One dataset has many preprocessing logs; used by many training jobs. |
| **PreprocessingLog** | log\_id, dataset\_id (FK), operation (string), timestamp, details (JSON) | Belongs to Dataset. |
| **Model** | model\_id, dataset\_id (FK), model\_type (enum: QLearning, DQN, PPO, RandomForest, SVM, ANN), hyperparams (JSON), trained\_at, performance\_metrics (JSON), model\_path | Belongs to Dataset; has many training runs. |
| **TrainingRun** | run\_id, model\_id (FK), seed (int), episodes (int), status (enum: running, completed, failed), start\_time, end\_time, logs (JSON) | Belongs to Model. |
| **SimulationScenario** | scenario\_id, model\_id (FK), comfort\_range (tuple), start\_date, end\_date, results (JSON) | Generates many SimulationResults. |
| **User** | user\_id, role (enum), email, hashed\_password, created\_at, is\_active | Has many actions; roles determine permissions. |

### JSON / Pydantic schemas (example)

from pydantic import BaseModel, Field  
from typing import List, Dict, Tuple, Optional  
  
class DatasetSchema(BaseModel):  
 dataset\_id: str  
 file\_path: str  
 created\_at: datetime  
 records\_count: int  
 features: List[str]  
 status: str  
  
class HyperParams(BaseModel):  
 learning\_rate: float  
 discount\_factor: float  
 exploration\_rate: float  
 # additional fields for DQN/PPO  
  
class ModelSchema(BaseModel):  
 model\_id: str  
 dataset\_id: str  
 model\_type: str  
 hyperparams: Dict[str, float]  
 trained\_at: Optional[datetime]  
 performance\_metrics: Optional[Dict[str, float]]  
 model\_path: str  
  
class SimulationScenarioSchema(BaseModel):  
 scenario\_id: str  
 model\_id: str  
 comfort\_range: Tuple[float, float]  
 start\_date: datetime  
 end\_date: datetime  
 results: Dict[str, float]

### Persistence choice

Use **SQLite** via SQLAlchemy for development (lightweight, zero config) and **PostgreSQL** for production (transactional integrity, concurrency support). Provide migration scripts using Alembic. Models are stored as files on disk (/models/{model\_id}.pkl for scikit‑learn or .pt/.h5 for deep learning). Data retention: logs and training metrics kept for 3 years (assumption). Provide automated backups to cloud storage (e.g., AWS S3) weekly. PII is limited to user accounts; hashed passwords use bcrypt.

## 8. APIs & Integrations

### Internal API endpoints (example)

| Path | Method | Description | Request | Response | Errors |
| --- | --- | --- | --- | --- | --- |
| /api/datasets | POST | Upload dataset and start preprocessing | Multipart form with file; options for cleaning (boolean flags) | {dataset\_id} | 400 if file invalid; 500 on server error |
| /api/models | POST | Create training job | {dataset\_id, model\_type, hyperparams} | {model\_id, run\_id} | 404 if dataset not found |
| /api/models/{model\_id}/runs/{run\_id} | GET | Retrieve training status | n/a | {status, progress, metrics} | 404 if not found |
| /api/evaluate/{model\_id} | POST | Evaluate model on test set | {dataset\_id} | {energy\_savings, comfort\_pct, reward\_curve, metrics} | 400 if data missing |
| /api/simulate/{model\_id} | POST | Run simulation scenario | {comfort\_range, start\_date, end\_date} | {energy\_prediction, comfort\_prediction} | 422 if invalid dates |

### Third‑party services

The system avoids external services for data processing to maintain privacy. Use open‑source libraries: Pandas for data handling; NumPy; Scikit‑learn for baselines; Stable-Baselines3 and Gymnasium for RL. Optionally integrate with **Streamlit Cloud** or containerised deployment platforms (Docker) for hosting (industry standard). Use analytics service (e.g., PostHog) to collect telemetry events.

### Rate limiting & retries

For internal APIs, enforce rate limit of 10 requests/second per user (industry standard). Use idempotency tokens for training job submissions to avoid duplicate runs. Implement exponential backoff retries for transient errors.

## 9. Python + Streamlit Implementation Plan

### Architecture diagram (textual)

┌─────────────┐ HTTP/WebSockets ┌──────────────┐  
│ Streamlit │ ───────────────────────▶│ Service layer│  
│ Front‑end │◀─────────────────────── │ (FastAPI) │  
└─────────────┘ RPC/Events └──────────────┘  
 │ │  
 │SQLAlchemy, Alembic │RL libraries, ML libs  
 ▼ ▼  
┌─────────────────────┐ Filesystem ┌──────────────────┐  
│ Data layer (DB) │◀─────────────────▶│ Storage (models)│  
└─────────────────────┘ └──────────────────┘

### Project structure

/app.py # Entry point for Streamlit app  
/pages/  
 home.py  
 preprocessing.py  
 training.py  
 evaluation.py  
 simulation.py  
/components/  
 charts.py # Reusable plot functions using matplotlib  
 forms.py # Custom form widgets (hyperparameter input)  
/services/  
 data\_service.py # Loading, cleaning, feature engineering, splitting  
 rl\_service.py # Q‑Learning, DQN, PPO training and inference  
 baseline\_service.py # RandomForest, SVM, ANN training  
 evaluation\_service.py # Metric computation and report generation  
 simulation\_service.py # Environment interface and scenario simulation  
/models/  
 entities.py # SQLAlchemy models for Dataset, Model, etc.  
 schemas.py # Pydantic schemas  
/data/  
 raw/  
 processed/  
/tests/  
 test\_data\_service.py  
 test\_rl\_service.py  
 test\_evaluation\_service.py  
/.streamlit/config.toml # Theme, server configuration  
requirements.txt # Dependencies  
pyproject.toml (optional)

### State management

Use st.session\_state to persist across reruns:

* dataset\_id – currently selected dataset
* preprocessing\_status – enum: not\_started, in\_progress, completed
* trained\_models – dictionary of model\_id → summary
* current\_run – run\_id being trained
* evaluation\_results – DataFrame of metrics
* simulation\_params – currently selected comfort range, start/end dates

Naming convention: snake\_case; prefix keys with page name (e.g., training\_model\_type) to avoid conflicts. Reset state on user logout or dataset change.

### Caching strategy

* Use st.cache\_data(ttl=3600) for data ingestion and preprocessing results; recompute if dataset changes.
* Use st.cache\_resource for loading large models (DQN network weights) and environment objects; keep across sessions until invalidated.
* Set TTL of 1 hour for environment to ensure updated data when new dataset loaded.
* Invalidate caches upon dataset upload or hyperparameter change.

### Long‑running tasks

* Training and simulation can take seconds to minutes; run them in separate threads using asyncio or background tasks via concurrent.futures to avoid blocking UI. Use st.progress and st.status to show progress. Provide option to cancel.

### File handling

* Limit uploaded file size to 100 MB (industry standard). Accept CSV and ZIP containing dataset. Validate MIME types. Store files temporarily in /tmp/uploads and move to /data/raw once validated. Auto‑delete temp files older than 24 hours.
* For downloads, provide energy and comfort results in CSV/Excel; ensure no PII. Use Streamlit st.download\_button.

### Secrets & configuration

* Use st.secrets for database connection strings, API keys (e.g., PostHog). For local development, load environment variables from .env. Never commit secrets.
* Provide /config.py to centralise configuration (DB URI, caching TTLs, default hyperparameters). Use dynaconf or Pydantic settings.

### Minimal code skeleton

# app.py  
import streamlit as st  
from pages import home, preprocessing, training, evaluation, simulation  
  
st.set\_page\_config(page\_title="RL Building Energy Optimisation", layout="wide")  
  
PAGES = {  
 "Home": home.render,  
 "Preprocessing": preprocessing.render,  
 "Training": training.render,  
 "Evaluation": evaluation.render,  
 "Simulation": simulation.render,  
}  
  
def main():  
 selection = st.sidebar.radio("Navigate", list(PAGES.keys()))  
 PAGES[selection]()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Each page’s render function imports services and uses st.session\_state to persist data. Services throw custom exceptions mapped to user‑friendly messages.

### Testing

* **Unit tests:** Use pytest to test each service function. Mock file reads/writes and ensure that data cleaning functions handle missing values correctly. Test RL training with small synthetic dataset to verify reward computation.
* **Integration tests:** Use pytest with FastAPI TestClient to test API endpoints. Validate that POST /api/models returns 201 and triggers training.
* **E2E tests:** Use Playwright with Streamlit to simulate user flows: upload dataset → preprocess → train model → evaluate → run simulation. Check that UI updates and downloads succeed.

### Performance targets & profiling

* Cold start time of Streamlit application ≤ 2 s on standard machine.
* Model inference latency ≤ 500 ms (p95) for single step and ≤ 2 s for daily simulation.
* Use line\_profiler and cProfile to profile long tasks; optimise loops and memory usage.
* Use GPU acceleration for DQN/PPO if available; fallback to CPU.

### Internationalisation & timezone

* Use Africa/Lagos timezone for date/time operations (per user context). Display date/time in DD MMM YYYY HH:MM format.
* Ensure number formatting matches Nigerian locale (comma thousands separator). Provide ability to change language (future work, assumption).

## 10. Non‑Functional Requirements (NFRs)

| Category | Requirement | Verification |
| --- | --- | --- |
| **Performance** | Preprocessing completes within 5 minutes for datasets ≤ 1 million rows (assumption). | Benchmark on test dataset; measure runtime. |
|  | Model training completes within 30 minutes for 1,000 episodes (Q‑Learning & DQN) and 2 hours for PPO (assumption). | Use training logs and measure runtime. |
|  | Inference latency ≤ 500 ms per step; ≤ 2 s per simulation. | Monitor runtime with timers. |
| **Reliability** | System availability 99 % (monthly uptime). | Deploy with health checks; use uptime monitoring. |
| **Scalability** | Handle multiple concurrent training jobs (up to 3) using job queue. | Simulate concurrent jobs and ensure system stability. |
| **Security** | Use secure authentication; store passwords hashed; enforce RBAC; input validation; sanitise file uploads. | Penetration testing; static code analysis. |
| **Privacy** | Process anonymised data locally; no PII exposure. | Review data schema; run privacy audit. |
| **Compliance** | Align with GDPR‑like principles: data minimisation, purpose limitation (industry standard). | Provide privacy policy; support data deletion on request. |
| **Accessibility** | Conform to WCAG 2.1 AA guidelines (focus order, contrast, alt text). | Conduct accessibility audit using automated tools. |
| **Maintainability** | Codebase follows PEP 8; functions documented with docstrings; modular architecture. | Code reviews; static linting. |
| **Portability** | Deploy via Docker; run on Linux and Windows; package requirements pinned. | Build and run container on multiple OS. |
| **Observability** | Emit logs, metrics, and traces; integrate with observability stack (Prometheus + Grafana). | Instrument code with logging; test dashboards. |

## 11. Security & Privacy

### Threat model (STRIDE)

| Threat | Mitigation |
| --- | --- |
| **Spoofing** | Use secure authentication (e.g., OAuth or JWT) and multi‑factor authentication for admin users (industry standard). |
| **Tampering** | Validate file uploads; compute hashes on dataset files; use HTTPS/TLS for all communication. |
| **Repudiation** | Maintain audit logs of all actions (dataset upload, training runs, simulations); timestamp logs. |
| **Information Disclosure** | Use role‑based access control; restrict sensitive metrics to authorised users. |
| **Denial of Service** | Rate limit API endpoints; validate input sizes; isolate training tasks with worker pools. |
| **Elevation of Privilege** | Strict separation of privileges; use principle of least privilege; review code for injection vulnerabilities. |

### Authentication & Authorisation

* Use JSON Web Tokens (JWT) for session management; tokens expire after 30 minutes of inactivity.
* Use role‑based access control to restrict model training to analysts and researchers; building managers can only view and simulate.
* Store hashed passwords using bcrypt.
* Use CSRF protection on forms and API endpoints.

### Secure file handling

* Verify file signatures and MIME types; reject executables.
* Store uploaded files in non‑public directories; generate random file names.
* Scan datasets for malicious content (industry standard).

### Input sanitisation

* Sanitize user inputs on hyperparameters, names, and file paths.
* Use prepared statements/ORM to prevent SQL injection.
* Validate JSON payloads against Pydantic schemas.

### Dependency management & SBOM

* Use tools like pip‑compile to pin dependencies; use safety or pip‑audit to scan for vulnerabilities.
* Generate Software Bill of Materials (SBOM) via tools like syft; store with release artifacts (industry standard).

## 12. Analytics & Experimentation

### Event taxonomy

| Event name | When | Properties |
| --- | --- | --- |
| dataset\_uploaded | User uploads dataset | user\_id, dataset\_id, file\_size, format |
| preprocessing\_completed | Preprocessing finishes | dataset\_id, duration, records\_processed |
| model\_training\_started | Training begins | model\_type, hyperparams, run\_id |
| model\_training\_finished | Training ends | model\_id, run\_id, episodes, best\_reward |
| evaluation\_viewed | Evaluation page loaded | user\_id, selected\_models |
| simulation\_run | Simulation executed | user\_id, model\_id, comfort\_range, energy\_prediction, comfort\_prediction |
| error\_occurred | Any error thrown | error\_type, module, timestamp |

### Core dashboards

* **Usage dashboard:** Number of datasets uploaded, models trained, simulations run over time; success/failure rates.
* **Model performance dashboard:** Energy savings and comfort across models; distribution of rewards; hyperparameter heatmaps.
* **User engagement dashboard:** Number of active building managers per week; average simulation runs per user.

### A/B testing plan

Test UI layouts or reward function variations: - Define control group using current reward function; treatment groups with modified weighting on comfort penalty. - Randomly assign training runs to groups; measure energy savings and comfort metrics. - Run statistical tests to assess significance; success if treatment achieves ≥5 % additional energy savings without comfort loss (assumption).

## 13. Rollout & Operations

### Environments

* **Development:** Local environment with SQLite; debug logging.
* **Staging:** Mirror of production; uses PostgreSQL; limited dataset; test user accounts; behind basic auth.
* **Production:** Secure environment; uses PostgreSQL; integrated with monitoring and CI/CD; auto‑scale training workers.

### Deployment options

* **Streamlit Cloud:** Quick deployment for prototypes; limited scalability; simple to set up.
* **Docker on cloud VM (e.g., AWS EC2, GCP Compute Engine):** Preferred for production; build image with Python runtime, install dependencies from requirements.txt, expose port 8501; use Nginx reverse proxy and TLS termination.
* Steps:
* Write Dockerfile with multi‑stage build (builder and runtime).
* Build image: docker build -t rl-bems-app ..
* Push to registry.
* Deploy with docker run or orchestrated via Kubernetes; mount persistent volumes for /data and /models.
* Set environment variables for DB, secrets, and caching.
* Use GitHub Actions for CI/CD: on push to main, run unit tests, lint with flake8, build Docker image, publish to registry, and deploy.

### Monitoring & alerting

* Use Prometheus to collect application metrics (latency, error rates, memory). Use Grafana dashboards.
* Set alert thresholds: training failure rate > 5 % (alert); inference latency > 1 s for 10 minutes (alert).
* Integrate with PagerDuty for on‑call notifications.
* Provide runbooks: guidelines for restarting services, handling failed training jobs, dealing with corrupted datasets.

### Feature flags & rollout

* Use feature flags for new algorithms or UI features (e.g., toggle new reward function). Deploy behind toggles; gradually enable for subsets of users; monitor metrics.
* Use unleash or ff4j library for flag management.

### On‑call notes

* Document common failure modes (e.g., memory errors during training, invalid dataset format) and recovery steps.
* Provide escalation contacts and support hours.

## 14. Timeline & Resourcing

### Milestones & effort estimates

| Milestone | Description | Est. effort |
| --- | --- | --- |
| **M1: Requirements & Design** | Finalise PRD, create architecture design, data model, UI wireframes. | 1 week |
| **M2: Data Ingestion & Preprocessing** | Implement data loader, cleaning functions, feature engineering, dataset partitioning. | 2 weeks |
| **M3: Simulation Environment & Baselines** | Implement environment, reward functions, baseline models; unit tests. | 2 weeks |
| **M4: RL Algorithms Implementation** | Implement Q‑Learning, DQN, PPO training and hyperparameter tuning. | 3 weeks |
| **M5: Evaluation & Metrics** | Build evaluation service; compute metrics; create dashboards. | 2 weeks |
| **M6: User Interface & Visualisation** | Develop Streamlit pages; integrate services; ensure accessibility. | 3 weeks |
| **M7: Security & Privacy** | Implement authentication, role management, secure file handling. | 1 week |
| **M8: Testing & QA** | Unit/integration/E2E tests; performance profiling. | 2 weeks |
| **M9: Deployment & Operations** | Create CI/CD pipeline; containerisation; staging & production rollout. | 2 weeks |
| **M10: Pilot & Feedback** | Conduct pilot with selected building managers; collect feedback; iterate. | 2 weeks |

Assume a team of four (product manager, data scientist, backend engineer, front‑end/DevOps engineer). Slack time allocated for risk mitigation and documentation.

### Risks & mitigations

| Risk | Likelihood | Impact | Mitigation |
| --- | --- | --- | --- |
| Data quality issues (missing or inconsistent data) | Medium | High | Robust preprocessing; data validation scripts; early detection during M2. |
| Model convergence failure (RL unstable) | Medium | High | Hyperparameter tuning; early experimentation; fallback to DQN baseline. |
| Performance bottlenecks (slow training/inference) | Medium | Medium | Profiling; optional GPU; code optimisation. |
| User adoption low due to complexity | Medium | Medium | Intuitive UI; onboarding tutorials; training sessions for managers. |
| Regulatory changes (data privacy) | Low | Medium | Monitor regulations; design for GDPR compliance; provide data deletion. |

## 15. Traceability & Coverage

### Coverage matrix

| Source requirement (doc) | PRD section(s) | Feature IDs | Test cases |
| --- | --- | --- | --- |
| Need for adaptive control strategies to handle fluctuating occupancy/weather | Executive Summary, Scope, FR‑004 | FR‑004 | TC‑ENV‑1..4 |
| Develop RL‑based system to optimise building energy consumption | Executive Summary, Goals, FR‑002 | FR‑002, FR‑004 | TC‑RL‑1..5 |
| Preprocess BDGP2 dataset (cleaning, imputation, normalisation, feature engineering) | Scope, FR‑001 | FR‑001 | TC‑PRE‑1..5 |
| Evaluate RL algorithms (Q‑Learning, DQN, PPO) and compare to baselines | Goals, FR‑002, FR‑003, FR‑005 | FR‑002, FR‑003, FR‑005 | TC‑EVAL‑1..4 |
| Use reward function balancing energy savings and comfort | FR‑002, FR‑004 | FR‑002, FR‑004 | TC‑ENV‑2, TC‑RL‑1 |
| Compute metrics: energy savings, thermal comfort, accuracy, reward and convergence【687278560629580†L2893-L2944】 | Success metrics, FR‑005 | FR‑005 | TC‑EVAL‑1..4 |
| Three‑tier architecture (IoT, edge, cloud) | FR‑007, Implementation Plan | FR‑007 | TC‑ARCH‑1..3 |
| Ethical considerations: privacy, local processing, transparency | FR‑008, UX, Security & Privacy | FR‑008 | TC‑ETHICS‑1..3 |

### Gaps & assumptions

| Gap / assumption | Owner | Due date |
| --- | --- | --- |
| Clarify whether baseline models beyond Random Forest, SVM, ANN are needed. | Product Manager | Before M3 |
| Decide if hyperparameter search should be user‑configurable or fixed grid. | Data Scientist | Before M4 |
| Determine integration with price signals or renewable generation. | Stakeholders | Future release |
| Confirm requirement for multi‑language interface and dark mode. | UX Designer | M6 |
| Data retention period and deletion mechanism for user data. | Legal | M7 |

## 16. Appendix

### Glossary

| Term | Definition |
| --- | --- |
| **BDGP2** | Building Data Genome Project 2 dataset; hourly energy/environmental data from over 3,000 buildings. |
| **HVAC** | Heating, Ventilation, and Air Conditioning system. |
| **RL (Reinforcement Learning)** | Machine learning paradigm where an agent learns by interacting with an environment via states, actions, and rewards. |
| **Q‑Learning** | Model‑free RL algorithm using Q‑table and Bellman updates. |
| **DQN (Deep Q‑Network)** | Extension of Q‑Learning using neural network approximator and experience replay. |
| **PPO (Proximal Policy Optimisation)** | Policy gradient RL algorithm optimising policy within a trust region. |
| **PMV (Predicted Mean Vote)** | Thermal comfort metric with scale −3 to +3; values between −0.5 and +0.5 considered comfortable. |
| **IQR** | Interquartile range; statistical measure used to detect outliers. |
| **Experience replay** | Buffer storing past transitions to decorrelate training samples. |
| **Target network** | Secondary network in DQN updated periodically to stabilise learning. |

### ASCII wireframe sketches

Home Page (wireframe):  
┌──────────────────────────────────────┐  
│ RL Building Energy Optimisation │  
├───────────────────────┬─────────────┤  
│ Intro text │ File upload│  
│ │ (Drop zone) │  
├──────────────────────────────────────┤  
│ Get Started → Preprocessing │  
└──────────────────────────────────────┘  
  
Training Page (wireframe):  
┌──────────────────────────────────────────────┐  
│ Select Model: [Q‑Learning ▼] │  
│ Hyperparameters: │  
│ α: [0.1] γ: [0.95] ε: [0.2] │  
│ [Start Training] [Cancel] │  
├──────────────────────────────────────────────┤  
│ Progress Chart (Episodes vs Reward) │  
│ Log Panel │  
└──────────────────────────────────────────────┘  
  
Simulation Page (wireframe):  
┌──────────────────────────────────────────────┐  
│ Comfort Range: [20]–[24] °C │  
│ Date Range: [Start Date][End Date] │  
│ [Run Simulation] │  
├──────────────────────────────────────────────┤  
│ Line Chart: Predicted Energy vs Time │  
│ Line Chart: Comfort Index vs Time │  
│ Summary Metrics │  
└──────────────────────────────────────────────┘

### Example config files

.streamlit/config.toml:

[server]  
headless = true  
port = 8501  
enableCORS = false  
  
[theme]  
base = "light"  
primaryColor = "#008080"  
secondaryBackgroundColor = "#F7F7F7"  
textColor = "#202020"

requirements.txt (partial):

pandas>=2.1.0  
numpy>=1.23  
scikit-learn>=1.3  
stable-baselines3>=2.1.0  
gymnasium>=0.28.1  
streamlit>=1.29  
sqlalchemy>=2.0  
alembic>=1.12  
pytest>=7.4  
playwright>=1.39  
bcrypt>=4.0

.env sample keys (no values):

DATABASE\_URL=  
SECRET\_KEY=  
POSTHOG\_API\_KEY=

## 17. Final Checklist

* [ ] All source‑doc requirements are represented and traceable.
* [ ] All features have acceptance criteria and telemetry.
* [ ] Python/Streamlit plan is production‑ready (state, caching, secrets, tests, deploy).
* [ ] Security, privacy, accessibility, and performance targets are specified.
* [ ] Risks, open questions, and owners are listed.