Software Requirements Specification (SRS)

for

Hybrid IRCTC Bot-Detection And Train Search Web App

Prepared By: Subradeep Das

Institute: CDAC CINE, IIT Guwahati Research Park, Guwahati, Assam

Course: Advanced Certificate Course in HPC-AI

Date: 11/08/2025

Version: 1.0

This document specifies software and system requirements for the Hybrid IRCTC Bot-Detection + Train Search Web App. It is intended for developers, testers, evaluators, and stakeholders responsible for design, implementation and validation.

Contents

1. Introduction

1.1 Purpose

This SRS describes in detail the functional and non-functional requirements for the Hybrid IRCTC Bot-Detection + Train Search Web App. It provides a clear, testable and traceable specification to guide implementation, testing and acceptance.

1.2 Scope

The system is a web application with two primary capabilities: *

- Authentication using an image-based CAPTCHA plus behavioral analysis to distinguish humans, bots and human-like bots. Only humans proceed.
- Train search and ticket confirmation prediction. After login, users can search trains by source, destination, journey date and class; results include train details and a predicted confirmation probability. Visual demand heatmaps per train are provided.

1.3 Definitions, Acronyms and Abbreviations

IRCTC Indian Railway Catering and Tourism Corporation

SRS Software Requirements Specification

FR Functional Requirement

NFR Non-Functional Requirement

CSV Comma Separated Values

API Application Programming Interface

UI User Interface

ML Machine Learning

TF TensorFlow

RF Random Forest

1.4 References

- Project repository - local
- GitHub repository: `Github Repo`
- Model file: `model/bot_detector_model.h5`
- Login attempts log: `login_attempts.csv`

1.5 Overview

This SRS describes system context, requirements (functional and non-functional), interfaces, data definitions, system models and acceptance criteria.

2. Overall Description

2.1 Product Perspective

The product is a standalone web application integrating:

- Frontend: HTML / CSS / JavaScript (login, search, admin dashboard)
- Backend: Flask (Python)
- ML components: TensorFlow behavioral classifier (bot detection); Scikit-learn Random Forest (ticket prediction)
- Database: SQLite (development) / PostgreSQL (production)
- Model storage: `model/` directory (e.g., `tf_bot_model.h5`, `rf_ticket_predictor.joblib`)
- Logs: `login_attempts.csv` used to trigger retraining after every 10 new entries

2.2 Product Functions

High-level functions:

1. Secure user registration and login with image-based CAPTCHA.
2. Behavioral capture during login (mouse movement, typing speed, scroll, focus changes, idle time, key timings).
3. Real-time classification: human, bot, human-like bot.
4. Access control: only humans access train search and predictions.
5. Train search by source, destination, date and class.
6. Display train details, punctuality rate, demand heatmap.
7. Predict ticket confirmation probability using Random Forest.
8. Admin dashboard: logs, filters, charts, manual retrain trigger, CSV download.

2.3 User Classes and Characteristics

End User (Passenger) Non-technical; needs straightforward UI for search and prediction.

Admin Technical; monitors logs, triggers retrain, downloads data.
Developer/Maintainer Implements and deploys the system.

2.4 Operating Environment

- Server: Linux (Ubuntu recommended), Python 3.8+
- Optional GPU for training (NVIDIA drivers + CUDA)
- Browsers: Latest Chrome, Firefox, Edge
- ML libraries: TensorFlow 2.x, scikit-learn, joblib

2.5 Design and Implementation Constraints

- Behavioral data collection must avoid storing raw PII.
- Bot-detection model retrains automatically after every 10 new entries in `login_attempts.csv`.
- Preserve the nine behavioral features currently used in the system.
- Minimize false positives (blocking real humans).
- Persist and version models in `model/`.

2.6 Assumptions and Dependencies

- Users have JavaScript enabled.
- Dataset for ticket prediction is representative.
- Sufficient compute available for inference within the latency targets.

3. Specific Requirements

This chapter lists detailed functional and non-functional requirements with acceptance criteria.

3.1 Functional Requirements

FR1 – User Registration and Login The system shall allow users to register (email + password) and login using an image-based CAPTCHA.

Acceptance: A new user can register and login within 2 minutes; CAPTCHA is generated and verified server-side.

FR2 – Behavioral Data Capture The system shall capture these behavioral features during login: mouse deltas/paths, mouse velocity/acceleration, typing speed, inter-key intervals, scroll events, focus in/out events, idle times, click coordinates, and CAPTCHA interaction metrics.

Acceptance: Each login attempt writes one CSV row with the required features and timestamp to `login_attempts.csv`.

FR3 – Bot Classification The system shall run a TensorFlow model to classify attempts into: human, bot, human-like bot.

Acceptance: Inference completes within the documented latency; predictions are returned with a confidence score.

FR4 – Access Control Only attempts labeled `human` shall be allowed to proceed to the search page; others are denied and logged.

Acceptance: Denied attempts show a clear message; no access to search pages.

FR5 – Retraining Trigger The system shall automatically retrain the bot-detection model when `login_attempts.csv` accrues 10 new entries since last training.

Acceptance: On trigger, backend starts a retrain job, evaluates metrics (accuracy/precision/recall), archives previous model and saves new model with metadata.

FR6 – Train Search Logged-in users can search trains by source, destination, date

and class. Results include train number, name, departure/arrival times, punctuality rate, and predicted confirmation probability.
Acceptance: Standard searches return within 2 seconds under baseline load.

FR7 – Ticket Confirmation Prediction The system shall use a Random Forest model (`model/rf_ticket_predictor.joblib`) with features: days_before_journey, weekday, month, avg_waitlist, punctuality_rate, class, encoded source/destination, and train_no features when available.
Acceptance: Prediction displayed with probability and model version.

FR8 – Heatmap Visualization Provide an interactive heatmap per train: avg_waitlist vs days_before_journey.
Acceptance: Heatmap is responsive, can be exported as PNG.

FR9 – Admin Dashboard Admin can view logs, filter by date/user/class, download CSV, trigger manual retrain, view model metrics and denied attempts.
Acceptance: Admin operations are available after admin authentication.

FR10 – Logging and Auditing Maintain audit logs for login attempts (IP, timestamp, UA, behavioral features) and retraining events. Logs are exportable and retained per policy.
Acceptance: Logs are queryable and downloadable by admin.

3.2 Non-Functional Requirements

3.2.1 Performance

- NFR-P1: Search response time < 2s (baseline).
- NFR-P2: Bot-classification inference < 500 ms on server (documented if different).
- NFR-P3: Baseline support for 100 concurrent users; provide scaling plan.

3.2.2 Security

- NFR-S1: Passwords hashed with bcrypt or Argon2.
- NFR-S2: HTTPS (TLS 1.2+) enforced.
- NFR-S3: Role-based access to model files and logs.
- NFR-S4: Behavioral data stored without raw PII.

3.2.3 Reliability and Availability

- NFR-R1: Target uptime 99% (deployment dependent).

- NFR-R2: Retrain jobs run as background tasks (Celery/RQ) and do not block web workers.

3.2.4 Maintainability

- NFR-M1: Modular code separation: frontend / API / ML.
- NFR-M2: Model metadata: training date, metrics, hyperparameters saved with each model version.

3.2.5 Usability

- NFR-U1: Registration and search flow can be completed by a new user within 3 minutes.
- NFR-U2: Responsive UI for mobile and desktop.

4. External Interface Requirements

4.1 User Interfaces

Key UI pages (mockups stored in repository):

- Login Page (image CAPTCHA + behavioral capture)
- Search Page (source/destination/date/class inputs + results)
- Train Details (heatmap + metrics)
- Admin Dashboard (charts, filters, logs)

4.2 Hardware Interfaces

Server exposes standard HTTP/HTTPS. If using GPU, specify model (e.g., NVIDIA RTX series), drivers and CUDA versions.

4.3 Software Interfaces

- REST endpoints (Flask): `POST /api/login`, `GET /api/search`, `POST /api/retrain`, `GET /api/admin/logs`
- Models: TensorFlow (HDF5/SavedModel) and scikit-learn joblib
- DB: SQL accessible via SQLAlchemy
- Optional: Celery + Redis (background jobs)

4.4 Communication Interfaces

- HTTPS (TLS), JSON payloads for APIs.
- Optional internal message queue for long-running tasks.

5. Software Requirements

This chapter lists every software-related requirement, packages, system dependencies, file expectations, deployment snippets and operational checklist needed to run, test, and operate the project in development and production. The content below is intentionally exhaustive and exact so you can copy/paste the commands and files directly.

5.1 Overview

The application is a Flask-based web app using TensorFlow (Keras) for behavioral bot detection and scikit-learn Random Forest for ticket confirmation prediction. It requires specific system libraries for image handling and plotting, Python packages for ML and web serving, and optional services (Redis, PostgreSQL) for scale.

5.2 Python runtime

- Python version: **3.8 – 3.11** (recommended: 3.10).
- Use a virtual environment for isolation: `python3 -m venv .venv` then `source .venv/bin/activate`.

5.3 Required Python packages (exact requirements.txt)

Save the following block as `requirements.txt` at project root and install with `pip install -r requirements.txt`.

```
# Web & server
Flask==2.2.5
gunicorn==20.1.0

# ML & data
tensorflow==2.11.0
```

```
scikit-learn==1.2.2
joblib==1.2.0
pandas==1.5.3
numpy==1.23.5

# Image & fonts
Pillow==9.4.0

# Plotting for heatmap
matplotlib==3.7.1

# Optional background worker
celery==5.2.7
redis==4.5.1

# DB adapter for Postgres (optional)
psycopg2-binary==2.9.6

# Utilities
python-dotenv==0.21.0
bcrypt==4.0.1
```

5.3.1 Notes on TensorFlow

- The line above installs the CPU TF wheel. For GPU, install the TF GPU wheel compatible with your CUDA/cuDNN — consult TensorFlow's compatibility matrix before pinning.
- If you will only run inference and not train large models, CPU TF is usually acceptable.

5.4 System / OS packages (Ubuntu/Debian)

Run before installing Python packages to ensure native dependencies build correctly:

```
sudo apt update
sudo apt install -y build-essential python3-dev python3-venv \
  libssl-dev libffi-dev libjpeg-dev zlib1g-dev libpng-dev \
  libfreetype6-dev pkg-config fonts-dejavu-core git curl
# For matplotlib and font rendering
```

```
sudo apt install -y libfreetype6-dev libpng-dev
# Optional: for postgres & redis
sudo apt install -y postgresql postgresql-contrib redis-server
```

5.5 GPU (optional)

If GPU acceleration is required:

- Install NVIDIA driver for your GPU.
- Install CUDA and cuDNN versions compatible with your TensorFlow version.
- Use an official NVIDIA CUDA base image for Docker if you containerize GPU workloads.

5.6 Required files, models and folders

Your application expects the following files and directories in exact structure. Create them before first run.

```
project-root/
 app.py
 requirements.txt
 Dockerfile # optional
 docker-compose.yml # optional
 train_search_dataset.csv
 data/
   train_availability_data.csv
 model/
   bot_detector_model.h5
   bot_scaler.pkl
   rf_ticket_predictor.joblib
 templates/
   index.html
   welcome.html
   access_denied.html
   train_results.html
 static/
   fonts/
      DejaVuSans.ttf
```

```
      Poppins-Regular.ttf
      Poppins-Bold.ttf
 logs/
    login_attempts.csv # created automatically if missing
    debug_predictions.log
    retraining_history.csv
```

5.6.1 Model metadata (recommended)

Keep a JSON metadata file next to each model:

```
model/bot_detector_model_metadata.json
{
  "version": "v1.0",
  "trained_on": "2025-08-01",
  "train_samples": 5000,
  "train_accuracy": 0.96,
  "val_accuracy": 0.93,
  "features": ["mouse_movement_units","typing_speed_cpm","click_pattern_score", "..."]
  "notes": "Initial training with synthetic augmentations"
}
```

5.7 Environment variables and secrets

Do **not** hardcode secrets in code. Use environment variables or a .env file (with gitignore).

```
# minimal .env
SECRET_KEY=your_strong_hex_or_base64_key_here
FLASK_ENV=production
DATABASE_URL=postgresql://user:pass@host:5432/dbname # optional
REDIS_URL=redis://localhost:6379/0                   # optional
LOG_DIR=/full/path/to/project/logs                   # optional override
```

5.8 Retraining policy and thresholds

- Retrain triggers: automatic when $\texttt{len(login}_{a}ttempts.csv)isdivisibleby10(i.e., every10newrecords$ $checks(fromcode)$:
- - Minimum total rows to retrain: **100**

- Minimum samples per class: **10**

Retrain acceptance: replace model only when:

- new validation accuracy > old validation accuracy
- AND new validation loss < old validation loss

On success: backup old model (`bot`$_d$*etector*$_m$*odel*$_b$*ackup.h*5), *savenewscaler*`bot`$_s$*caler.pkl*, *appendretrain*

5.9 Production services and recommended components

- WSGI: `gunicorn` for production process management.
- Reverse proxy / TLS: `nginx` in front of gunicorn (TLS termination).
- Background tasks: `Celery` with `Redis` broker for retraining and long-running tasks (recommended).
- Database: `PostgreSQL` for storing users, model metadata, and logs at scale (optional but recommended).
- Containerization: `Docker` for reproducible deployments; `docker-compose` for local multi-service setups.

5.10 Dockerfile (example)

Place this in `Dockerfile` (CPU TF example):

```
FROM python:3.10-slim

WORKDIR /app
COPY requirements.txt .

RUN apt-get update && apt-get install -y build-essential libfreetype6-dev libpng-dev l
 && pip install --no-cache-dir -r requirements.txt \
 && apt-get clean && rm -rf /var/lib/apt/lists/*

COPY . /app

ENV PYTHONUNBUFFERED=1
EXPOSE 5000
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "app:app"]
```

5.11 docker-compose.yml (example)

A simple compose file that includes Redis and Postgres (optional):

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./:/app
    environment:
      - SECRET_KEY=${SECRET_KEY}
      - DATABASE_URL=postgresql://postgres:postgres@db:5432/appdb
      - REDIS_URL=redis://redis:6379/0
    depends_on:
      - db
      - redis

  redis:
    image: redis:7

  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: appdb
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

5.12 systemd unit (example)

Use this to run gunicorn as a systemd service on Ubuntu:

```
# /etc/systemd/system/hybrid-app.service
```

```
[Unit]
Description=Hybrid IRCTC Bot-Detection + Train Search
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/path/to/project
Environment=PATH=/path/to/project/.venv/bin
Environment=SECRET_KEY=your_secret_here
ExecStart=/path/to/project/.venv/bin/gunicorn -w 4 -b 127.0.0.1:5000 app:app

[Install]
WantedBy=multi-user.target
```

5.13 Security and operational notes

- **SECRET_KEY** must be set via env and not hardcoded. Replace any `app.secret_key='your_se` in production.
- Enforce HTTPS and secure cookies: `SESSION_COOKIE_SECURE=True` and `SESSION_COOKIE_HTTPONL`
- Do not store raw passwords or PII in logs. If usernames are sensitive, hash them before writing to logs.
- Apply rate limiting on `/predict` endpoint (nginx or middleware) to prevent abuse.
- Use least-privilege for filesystem and DB access.

5.14 Testing and CI

- Unit tests with `pytest` (cover Flask routes and model inference wrappers).
- Integration tests using Flask test client (end-to-end login/predict/search flows).
- CI pipeline (GitHub Actions/GitLab CI) to run lint, tests, build docker image.
- Use `black` for formatting and `flake8` for linting (optional).

5.15 Monitoring and logging

- Rotate logs (logrotate) for `logs/`.
- Optional: forward logs to ELK/Fluentd; collect metrics with Prometheus + Grafana.
- Optional: Sentry for error reporting (DSN via env var).

5.16 Operational checklist (pre-deploy)

```
- [ ] Confirm SECRET_KEY is set via env and not hardcoded.
- [ ] Ensure model/ contains bot_detector_model.h5 and bot_scaler.pkl.
- [ ] Ensure model/rf_ticket_predictor.joblib exists.
- [ ] Place fonts in static/fonts/ (DejaVuSans.ttf, Poppins-Regular.ttf, Poppins-Bold
- [ ] Place train_search_dataset.csv and data/train_availability_data.csv.
- [ ] Ensure logs/ directory exists and is writable.
- [ ] If using Postgres/Redis, ensure services are up and DATABASE_URL/REDIS_URL are s
- [ ] Configure gunicorn + nginx for TLS in production.
- [ ] Run unit tests and integration tests.
```

5.17 Acceptance targets (software)

- Bot inference latency: < 500 ms on server for single prediction (document observed latency).
- Search latency: < 2 s under baseline load.
- Retrain minimum dataset: 100 rows; minimum per-class: 10 samples.
- Model backup and metadata must be present when new model is promoted.

6. System Models

6.1 Use Case Descriptions

UC1 – User Login Actor: End User. **Precondition:** Registered. **Main flow:** Enter credentials, solve CAPTCHA, behavioral payload sent; backend classifies; if human, grant access; else deny and log.

6.2 Workflow / Architecture Diagram

Figure 6.1 shows the system workflow and architecture.

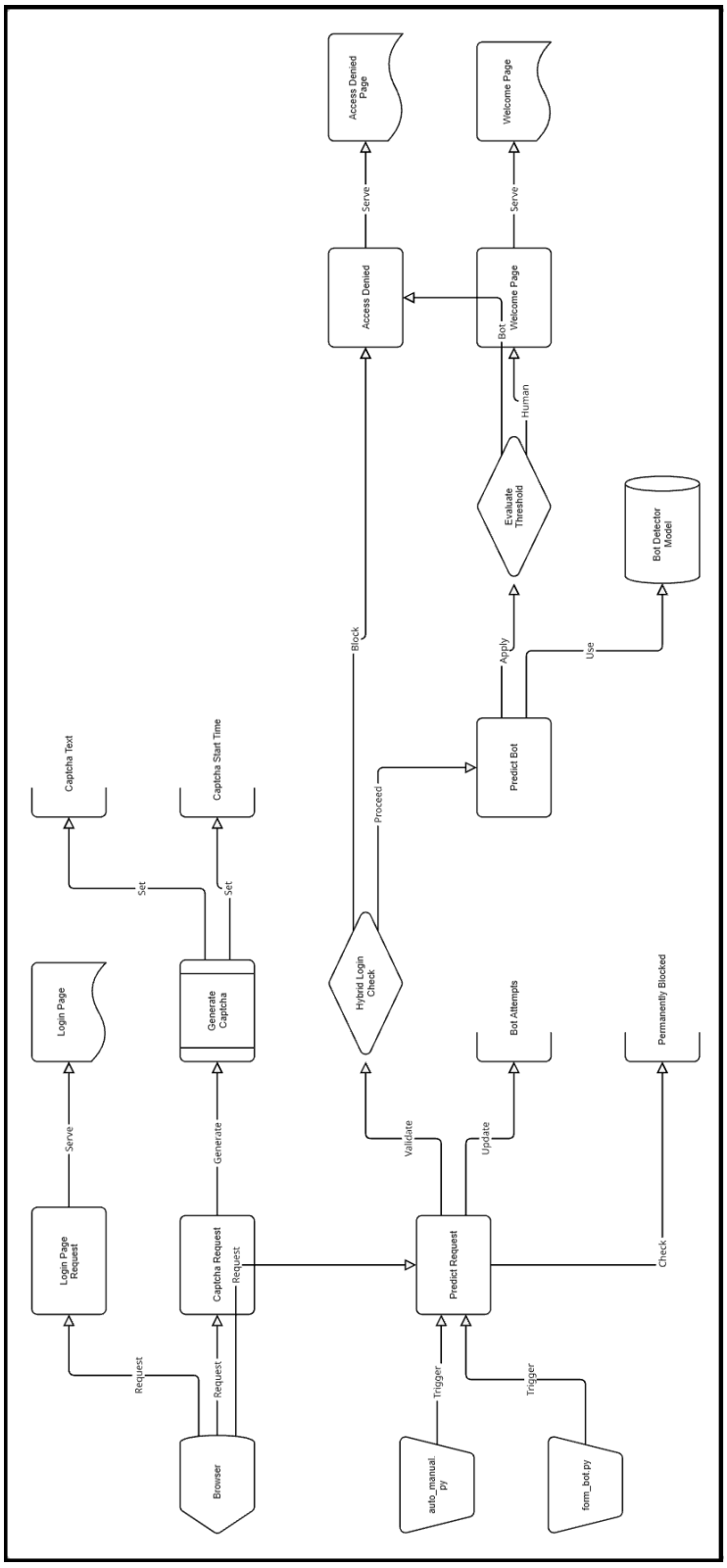

Figure 6.1: System workflow and architecture (full-page landscape).

6.3 Data Flow Diagram

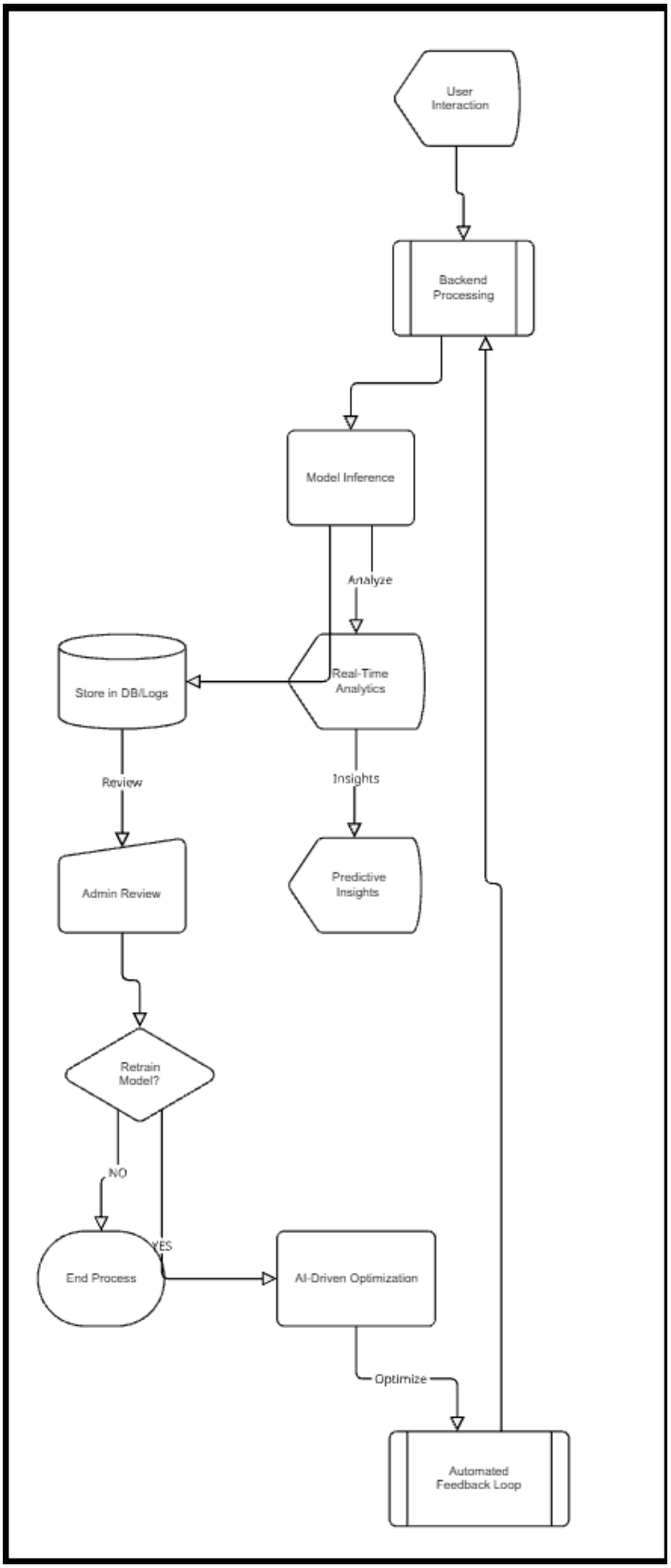

Figure 6.2: High-level data flow (top-to-bottom).

7. Data Definitions and Database Schema

7.1 Major Data Entities

- **User** { user_id, name, email, password_hash, created_at, role }
- **LoginAttempt** { attempt_id, user_id (nullable), timestamp, ip, user_agent, features_json, label, model_version }
- **Train** { train_no, train_name, source, destination, departure_time, arrival_time, punctuality_rate }
- **Prediction** { prediction_id, train_no, timestamp, features_json, predicted_prob, model_version }

7.2 Sample Table Schemas (SQL)

```
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY,
  name TEXT,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role TEXT DEFAULT 'user',
  created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE login_attempts (
  attempt_id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(user_id),
  timestamp TIMESTAMP DEFAULT now(),
  ip TEXT,
  user_agent TEXT,
  features_json JSONB,
  label TEXT,
```

```
  model_version TEXT
);

CREATE TABLE trains (
  train_no TEXT PRIMARY KEY,
  train_name TEXT,
  source TEXT,
  destination TEXT,
  departure_time TIME,
  arrival_time TIME,
  punctuality_rate REAL
);

CREATE TABLE predictions (
  prediction_id SERIAL PRIMARY KEY,
  train_no TEXT REFERENCES trains(train_no),
  timestamp TIMESTAMP DEFAULT now(),
  features_json JSONB,
  predicted_prob REAL,
  model_version TEXT
);
```

8. Validation, Verification and Acceptance Criteria

8.1 Unit and Integration Testing

Unit tests should cover backend routes, data validation, ML wrappers. Integration tests should cover login->classification->search flows.

8.2 Model Evaluation

Record per-class metrics (accuracy, precision, recall, F1) for bot-detection. New models replace production only if they meet baseline or after manual review.

8.3 Acceptance Tests

- AT1: Human login succeeds and grants access.
- AT2: Automated bot attempt is detected and denied.
- AT3: Prediction shown with confidence in search results.
- AT4: Admin can download logs and trigger retrain.

9. Requirements Traceability Matrix

Requirement ID	Description	Verification Method
FR1	User Registration and Login	Unit + Integration + Manual
FR2	Behavioral Data Capture	Review CSV + automated tests
FR3	Bot Classification	Model tests + acceptance tests
FR5	Retraining Trigger	Integration test with synthetic entries
FR6	Train Search	Performance and functional tests

10. Appendix

10.1 Glossary

Domain-specific terms.

10.2 Future Enhancements

- OTP-based secondary authentication
- Model serving (TensorFlow Serving / TorchServe)
- Multi-language UI
- A/B experiments for model variants

10.3 Document Revision History

Date	Change	Author
11/08/2025	1) Updated workflow (landscape full-page) 2) Added vertical Dataflow diagram 3) Added GitHub repo hyperlink under References 4) Other miscellaneous formatting fixes	Subradeep Das