Non-Functional Requirements and Design Solutions

1. Performance

Requirement:

The system should process requests quickly, handle large CSV uploads, and respond efficiently for concurrent users.

How the Design Addresses It:

Asynchronous CSV Processing:

- The Celery worker with a Redis broker allows file uploads to be processed in the background without blocking the main thread.
- This ensures that the API remains responsive, even for large files.

Pagination:

- The PageNumberPagination in get_records ensures that large datasets are delivered in manageable chunks, preventing slow response times.

Caching:

- Results of search and get_records are cached in Redis . This reduces database load for frequently requested data.

Query Optimization:

- Querysets uses .order_by('id') to ensure consistent and optimized database queries.

2. Scalability

Requirement:

The system should scale horizontally to accommodate increasing data volume, user load, and file sizes.

How the Design Addresses It:

Background Task Scaling:

- Celery workers can be scaled horizontally by adding more workers to process CSV files concurrently.

Database Optimization:

- The database (MySQL) can handle large datasets efficiently with proper indexing.

Stateless API Design:

- The RESTful APIs are stateless, allowing deployment behind a load balancer for horizontal scaling.

Caching with Redis:

- Cached responses reduce load on the database, improving scalability as traffic increases.

3. Availability

Requirement:

The system must ensure high availability with minimal downtime.

How the Design Addresses It:

Fault Tolerance:

- Redis for caching and as a Celery broker ensures reliability for background task management.

Separation of Concerns:

- Background tasks (CSV processing) are decoupled from the main API server, ensuring the API remains available even if a background task fails.

Logging:

- Application logs (using logging.FileHandler) capture errors and debug information to quickly identify and resolve issues.

4. Security

Requirement:

Ensure data security during transmission, storage, and access.

How the Design Addresses It:

Authentication and Authorization:

- JWT-based authentication (rest_framework_simplejwt) ensures that only authenticated users can access the APIs.

- Endpoints enforce the IsAuthenticated permission class to restrict unauthorized access.

File Validation:

- Uploaded files are validated for content and headers (validate_csv), ensuring invalid files are not processed.

Input Validation:

- The system validates query parameters (e.g., strip() on search parameters) and file format before processing.

Role-Based Access Control (potential future improvement):

- While not implemented, RBAC can be added to restrict certain actions (e.g., editing records) to admins.

5. Maintainability

Requirement:

The system should be easy to maintain, debug, and extend.

How the Design Addresses It:

Separation of Concerns:

- Views, serializers, models, and tasks are properly separated into their respective files.

Logging:

- Logs capture user actions, file uploads, and errors to aid debugging (logger.info() and logger.error()).

Modular Design:

- Each endpoint performs a single responsibility (e.g., upload, search, edit), making it easier to maintain.

Testability:

- Unit tests are provided for all endpoints (e.g., uploading, searching, editing) to ensure reliability during changes.

6. Reliability

Requirement:

The system should handle errors gracefully and ensure task completion.

How the Design Addresses It:

Graceful Error Handling:

- Each view has try-except blocks to catch and log errors while returning appropriate status codes (500, 400, 404).

Logging:

- Errors are logged with stack traces for easier troubleshooting (logger.error()).

CSV Validation:

- Files are validated for correct headers before processing, reducing the risk of processing invalid data.

7. Usability

Requirement:

The system should be easy to use and interact with.

How the Design Addresses It:

Clear Error Messages:

- API responses provide descriptive error messages like "Invalid CSV: headers not matched" or "No match found."

Consistent API Design:

- RESTful endpoints (/, /upload/, /records/, /records/<pk>/) follow a clear structure.

Pagination:

- Results are paginated, making it easier for users to navigate through large datasets.

Summary Table

Non-Functional Requirements	How the Design Addresses It
Performance	Asynchronous Celery tasks, caching with Redis, query optimizations, pagination.
Scalability	Horizontal scaling with Celery workers, stateless REST APIs, caching.
Availability	Fault tolerance, logging, retry mechanisms for task processing.
Security	JWT authentication, file validation, HTTPS enforcement, input validation.
Maintainability	Modular code, logging for debugging, separation of concerns, test cases.
Reliability	Error handling, logging, and validation for input files and parameters.
Usability	Descriptive error messages, pagination, clean RESTful API design.