1. Technology Stack

Component	Choice	Reason for Choosing
Frontend	React.js	React provides a fast, component-based UI with reusable components for scalability.
Backend	Django REST Framework (DRF)	Django handles API requests efficiently, and DRF simplifies API development.
Database	MySQL	MySQL is a robust, relational database suited for structured data like pricing feeds.
Caching	Redis	Redis improves performance by caching frequently queried search results.
Task Queue	Celery + Redis	Celery processes large CSV files asynchronously, preventing system slowdowns.
API Communication	Axios (React)	Axios simplifies API requests with clean error handling and support for interceptors.
Deployment	Docker (Optional)	Ensures consistent development and production environments.

---

2. Backend Design Decisions

- **2.1. Django and DRF**
- **Why Django?**
    - Django provides a robust framework for rapid development.
    - It follows the **MVT (Model-View-Template)** architecture, ensuring separation of concerns.
- **Why Django REST Framework (DRF)?**
    - DRF simplifies API development for CRUD operations (upload, search, edit).
    - Built-in serialization reduces manual code.
- **Key API Endpoints**:
    - POST /upload/: Upload CSV file.
    - GET /records/: Fetch paginated records.
    - GET /search/: Search records using filters.
    - PUT /records/<id>/: Edit an existing record.

- **2.2. Celery for Background Tasks**
- **Why Celery?**
    - Processing large CSV files can block the server.
    - Celery offloads file processing to a background worker to improve responsiveness.
- **Redis as Broker**:
    - Redis acts as the message broker for Celery tasks, ensuring quick communication between workers and the backend.
- **Flow**:
    - User uploads a CSV → Task is sent to Celery → Celery processes the file → Parsed data is stored in **MySQL**.

- **2.3. Redis Caching**
- **Why Redis?**
    - Searching large data repeatedly can slow the database.
    - Redis caches frequently queried search results to improve performance.
- **Cache Invalidation**:
    - When a record is edited, the cache is cleared using a **key-based invalidation** strategy.

3. Frontend Design Decisions

- **3.1. React.js for the UI**
- **Why React?**
    - React provides a fast and scalable solution for creating dynamic and interactive UIs.
    - Component-based architecture allows reusability (e.g., UploadCSV, SearchRecords, RecordsList).
- **Key Components**:
    - uploadCSV.js: Allows CSV uploads.
    - searchRecords.js: Provides search functionality.
    - recordsList.js: Displays records with pagination.
    - editRecord.js: Allows editing of specific records.

- **State Management**:
    - **useState and useEffect hooks** are used for managing local component state (e.g., search results, current page, and loading states).
- **API Communication**:
    - **Axios** is used for API calls because of its clean syntax and better error handling.

- **3.2. Pagination**
- **Why Pagination?**
    - Large datasets need to be broken into manageable chunks to improve performance.
    - Pagination reduces the payload size and enhances responsiveness.
- **Backend Pagination**:
    - Implemented using Django's **PageNumberPagination**.
- **Frontend Integration**:
    - React dynamically updates the page content using the **Next** and **Previous** buttons.

- **3.3. Edit Workflow**
- **Why Client-Side Edit Form?**
    - Users can update specific records dynamically without reloading the entire page.
- **How it Works**:
    - The **Edit button** triggers a modal or form to edit a record.
    - The updated data is sent to the backend using a **PUT request**.
    - Once updated, the frontend refreshes the records list.

4. Database Design Decisions

- **4.1. MySQL for Storage**
- **Why MySQL?**
    - MySQL is a relational database that is well-suited for structured data like pricing feeds.
- **Table Structure**:
  The PriceData table stores:

- id: Auto-incremented primary key.
- store_id: Unique identifier for a store.
- sku: Stock Keeping Unit.
- name: Product name.
- price: Product price.
- date: Date of the record.

5. Security Design Decisions

- **Authentication**:
  - JWT (JSON Web Token) authentication ensures secure access to the backend APIs.
  - Tokens are validated with each request.
- **File Upload Validation**:
  - CSV files are validated for required headers: Store ID, SKU, Product Name, Price, and Date.
- **Input Validation**:
  - User inputs (search queries, edits) are sanitized to prevent injection attacks.
- **Error Logging**:
  - Implemented using Python's **logging** module to handle and track errors.

7. Assumptions

- MySQL database is pre-configured.
- Redis and Celery are correctly set up.
- CSV files adhere to a specific format.
- Users have valid JWT tokens to access the APIs.

Summary of Design Decisions

Layer	Choice	Reason
Frontend	React.js, Axios	Fast, modular UI and efficient API calls.
Backend	Django REST Framework	Robust API handling and structured design.
Database	MySQL	Structured data storage.
Caching	Redis	Improves search performance.
Task Queue	Celery + Redis	Handles background processing.
Security	JWT Authentication	Secure API access.