Data Management System

**Design Document**

**Architectural Decisions**

1. **Django with PostgreSQL**: Chosen for scalability and schema flexibility.
2. **JWT Authentication**: Secure API endpoints with token-based authentication.
3. **Celery for Asynchronous Processing**: Handles large CSV imports without blocking requests.
4. **Django REST Framework**: Simplifies API development and integration.
5. **React JS**: Chosen for its efficient rendering and component-based architecture, as it allows for building dynamic and responsive user interfaces

**Anticipated Challenges & Solutions**

|  |  |
| --- | --- |
| **Challenge** | **Solution** |
| Handling large CSV imports efficiently | Used Celery for background processing |
| Ensuring schema flexibility | Implemented dynamic model creation at runtime |
| Pagination for large datasets | Implemented Django's pagination framework |
| Ensuring data integrity | Used PostgreSQL constraints and validations |

**Future Expansion Considerations**

* **Role-based access control (RBAC)** for more granular security.
* **Improved Frontend Dashboard** using React for a better user experience.
* **GraphQL API Support** for more efficient data fetching.
* **Automated Backups** to ensure data persistence in case of failure.

**Contribution**

1. Fork the repository.
2. Create a new branch: git checkout -b feature-branch.
3. Commit changes: git commit -m 'message'.
4. Push to GitHub: git push origin feature-branch.
5. Create a pull request.