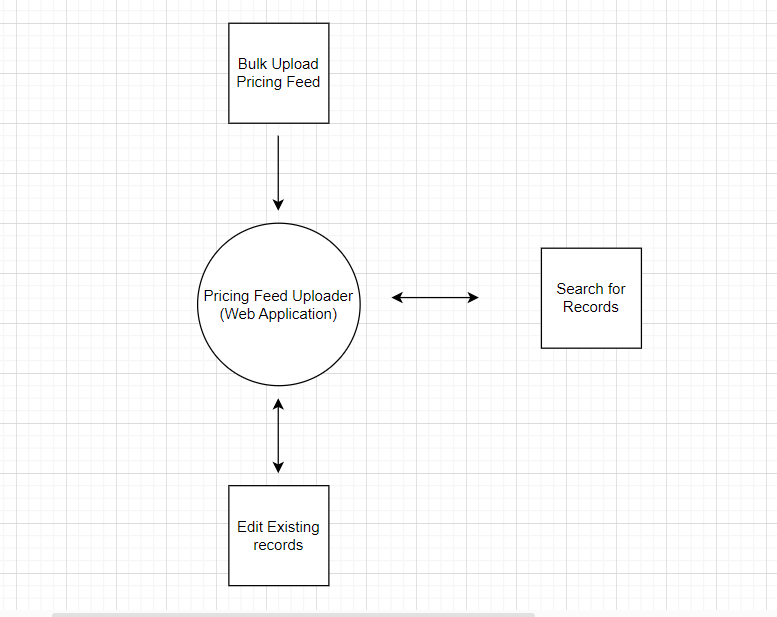
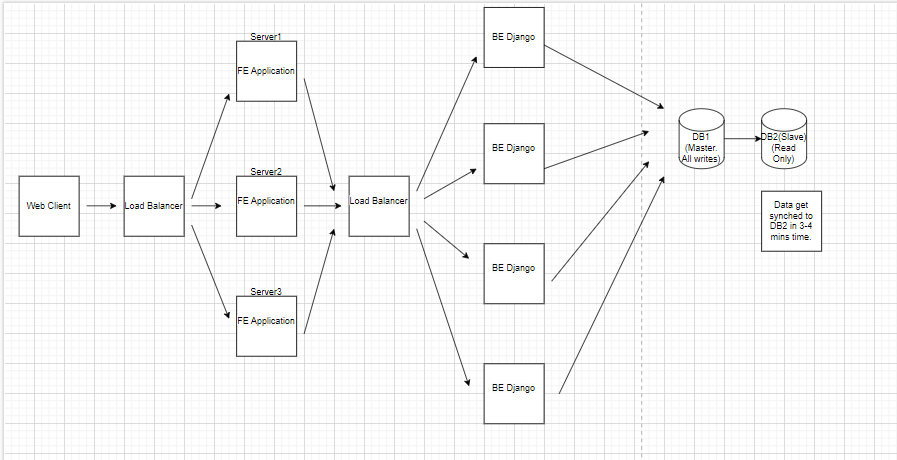
# Price Feeder

**Context Diagram**



**Architecture Diagram**

****

**Application Flow (Developed as part of the submission)**

* User will be able to upload the csv excel file with 5 fields i.e StoreId, Product name, Product SKU, Price and Date in the excel sheet.
* User will be able to see all the records irrespective of the region/country/ user who has uploaded the same.
* Pagination functionality is given so that application handles large data.
* We can filter the records by StoreId, Product Name, Product SKU and the Price Range.
* On click of any any Product feed, User can update the corresponding details.

**Application flow that we can include as part of Enhancements**

* Authentication mechanism and showing records only for the specific storeId which is mapped to the logged in user.
* Date filter in the pagination API.
* Storing data in more efficient way i.e dividing between different databases or different tables within same database.
* Admin view to see all the records with country filter.

**Design Designs**

* Application FE is made using React JS.
  + Used by Big technology firms. Hence proven track record to design high complexity application.
  + Large Developer community
  + Can be easily ported to Mobile devices by making PWA or Using similar framework React Native.
  + Client side rendering. Good for Single Page Application.
* Application Backend is made using Django.
  + Python – widely used language
  + Objective is to make Analytics Application and Python library supports such things.
  + Django – Used by large Technology firms.
  + Large Developer community.
  + Easy to learn and master.
* Database
  + Used Mysql (Relational Db)
  + The data is sales data and needed for many calculations and other purposes. Hence Relational DB.
  + DB – We can divide based on larger regions. i.e Asia , North America etc. There will be separate structure similar to shown above for specific regions.
  + In one region, In order to scale the DB we can follow Master Slave and hence can help us in case the read requests are on the higher side.
  + Will have different sales table in the DB to store information specific to one country in one region.
  + Index are applied on Column which are more prone for searching.
* Cloud Service Provider
  + Can go with Azure/AWS. Azure is the preferred choice because of the ease to uuse as experienced in past projects.
  + Provides Azure Active directory and other setup.
  + Provided Azure Devops board for running the project development.
  + Provided Azure WIKI to document the development.
  + Provided the Infrastructure and other things.
  + Maintained by Microsoft and gained popularity in recent times.
  + Application server, Db can be hosted in region specific to our division strategy to improve application performance.

**Non-Functional requirement**

* **Scalabilty :** Many FE and BE server help maintain the scalable requirements. AlsoScale of the application will not drastically increase at the onset of some event and hence we can go with fixed number of servers. These are configurable from cloud system and can be easily scaled whenever needed. We can also go for Auto scaling plan by many cloud service provider.
* **Reliabiltiy:** We are using many app servers which provides the reliable connection. There is no single point of failure and hence system remain up most of time. MYSQL Db further is ACID complaints and is being used from long time. Hence provides a reliable setup.
* **Maintainability/ Serviceability :** System uses cloud infrastructure and hence easy to maintain. The cloud provider provides good support whenever needed. The cloud provider has good uptime history and provides support whenever needed.
* **Security :** Authentication is included in the designwhich will prevent undue data access. Also the application would be using https so that the data flow is encrypted and is secure.
* **Economic:** We are using SQL db and no big analytics platform like Synapse and other. This helps us in minimizing the cost.

**Assumptions**

* Assumptions taken in the application code:
  + The application connects to single DB. Simplified case. For now code connects to inmemory DB
  + There is no validation done w.r.t to any data while inserting the records. Ideally we would validate the storeId and can use the mapping between storeId and Country within some region to insert data in different countrywise table. This would help in efficient data management and would optimize the get queries.
  + Indexing is not included in any tables that we have used in the application.
  + No authentication mechanism is used in the code. Also we are not maintaining the user who is inserting the records.
  + Ideally, User will be able to see his/her own records. This assumption is helpful in designing the Database table.
  + Date Range filter is not included in the code.
  + The application shows minimal flows which handles all the business requirement as mentioned in the problem statement.
* Assumption used in design
  + SQL db is used since assumption is sale data is needed for various calculations/other use case.
  + The application is provided with different setup within some continent so as to handle large load across the globe.
  + Data within one region is divided into country-wise tables to improve the application performance.
  + The store are uniformly divided across continents. The differential application setup will help us in case the store are not divided in similar way.
  + Within one region, Data is not very high which will need more scaling at Database side.
  + 3000 store with 2 users using the application. Users will update daily feed twice daily. On Average, there will be 2000 active stores. Each user will do 30 request on average case in single use.

Total request

* 2000 \* 30 \*2 \*2 per day
* 2000 \*30\*2\*2/100k per second
* 2.4 request per second across globe
* 2.4 /7 since we are dividing the system across continents
* .5 request per second
* Average request takes 1 second to serve
* Hence we need 4 machines so that our system does not breaks on limits.