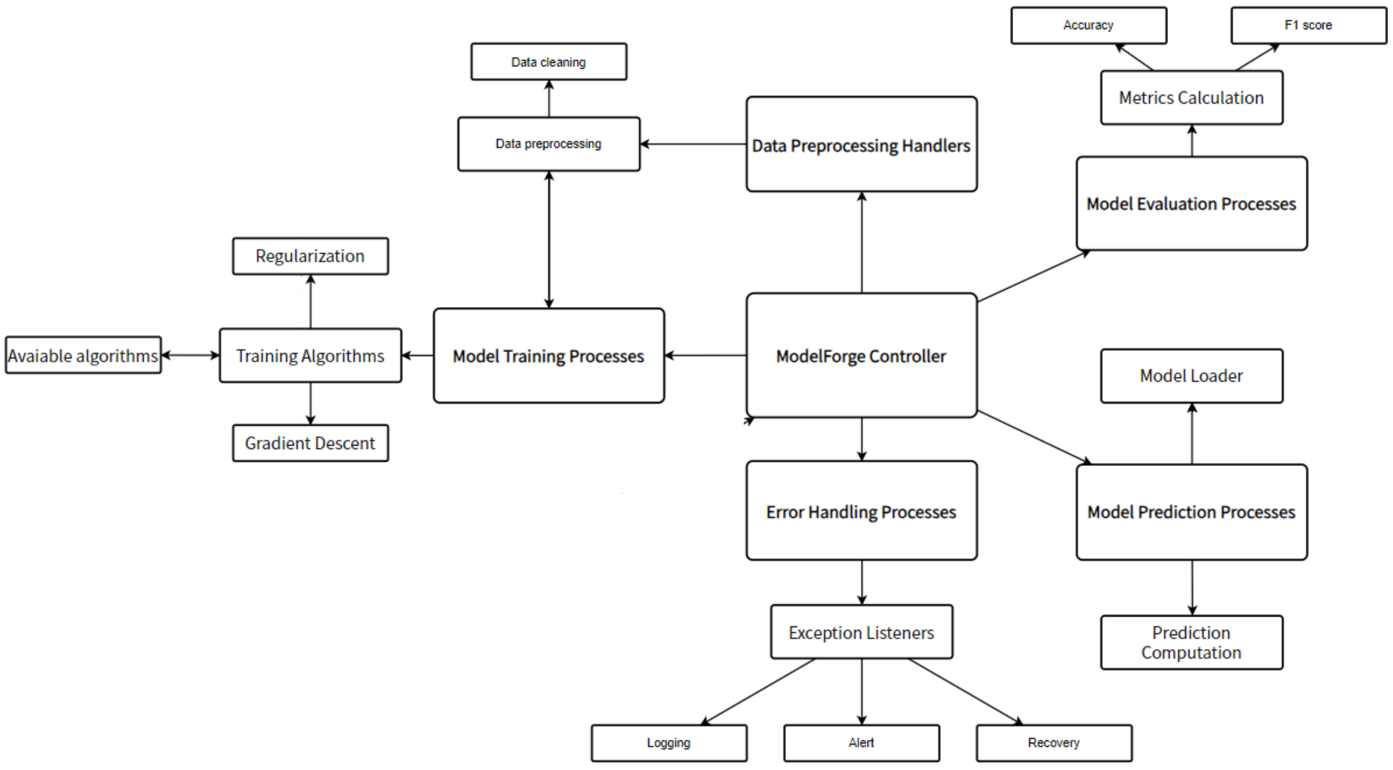
Prepared by Yukta, Brij, Mayank, Nimish

Design Document

SE Lab - Deliverable 2

1. Architecture

Diagram: Data-Centric Architecture



Components and Connectors

ModelForge Controller: Serves as the central management system orchestrating the ModelForge library operations.

Model Training Processes: Executes algorithms to create machine learning models from training data.

-Data Preprocessing Handlers: Prepares raw data for analysis, ensuring it is clean and formatted correctly.

-Data Cleaning: Removes inaccuracies and inconsistencies from data to improve quality.

-Training Algorithms: Mathematical procedures used to teach models how to make predictions.

-Gradient Descent: An optimization algorithm that minimizes the cost function in learning.

-Regularization: Methods used to prevent overfitting by penalizing complex models.

Model Prediction Processes: These processes handle the generation of predictions based on the trained models.
-Model Loader: Loads the trained model into memory for prediction tasks.

-Prediction Computation: Calculates the predicted values using the loaded model.
Model Evaluation Processes: These are responsible for assessing the performance of models using various metrics.
-Metrics Calculation: Quantifies the accuracy and effectiveness of the predictive models.

-Accuracy, F1-Score: Classification metrics used to measure the quality of predictions.
Data Preprocessing Handlers: Components that handle the preparation and validation of input data for training models.

Error Handling Processes: Processes designed to handle and report errors during model training, prediction, and evaluation.
-Exception Listeners: Monitors for any irregularities or runtime exceptions to handle them appropriately.

-Logging: Records events, especially errors, for later review.

-Alert: Creates notifications when errors or significant events occur.

-Error Recovery: Defined steps to recover from errors and restore normal operations.

Mapping to NFR’s and FR’s

Connectors and Nodes Mapping to **FR**s:

- **ModelForge Controller**:
  - Manages system operations, covering FR1 (System Initialization), FR2 (User Session Management), and FR3 (System Configuration Management).
- **Model Training Processes**:
  - Data Preprocessing and Training Algorithms like Gradient Descent and Regularization map to FR4 (Model Training) and FR6 (Model Learning).
- **Model Prediction Processes**:
  - The process of loading models and computing predictions corresponds to FR7 (Generate Predictions).
- **Model Evaluation Processes:**

    - The activities for calculating performance metrics relate to FR8 (Model Evaluation).

Connectors and Nodes Mapping to **NFR**s:

- **Error Handling Processes:**
    - Exception Listeners and Logging are designed for system robustness, mapping to NFR1 (Error Detection), NFR2 (Error Logging), and NFR3 (Error Recovery).
- **Modularity (Design Decision):**
    - Allows for independent development and maintenance, impacting NFRs related to maintainability and scalability.
- **Scalability (Design Decision):**
    - The system's ability to handle more extensive data sets or a higher number of requests impacts NFRs related to performance and scalability.
- **Performance Optimization (Design Decision):**
    - Selection of specific algorithms for efficiency maps to NFRs related to system performance and accuracy.

Each node and connector in the ModelForge architecture is designed with specific FRs and NFRs in mind, ensuring that the system not only meets its current requirements but is also poised for future expansion and integration.

Why we chose this design over others

A data-centric architecture is ideally suited for ModelForge due to its intrinsic ability to prioritize the data—which is the lifeblood of any machine learning system. This approach ensures the utmost data quality and integrity, which are paramount for the accuracy of predictions and model reliability.

It offers scalability to handle large and expanding datasets, which is essential for machine learning applications that typically process vast amounts of information.

Additionally, it provides robust data security, a must-have when dealing with sensitive or proprietary data. The modularity inherent in a data-centric design facilitates easier maintenance and agile adaptation to emerging requirements without overhauling the system.

2. Detailed Design

Data Structures and Algorithms

General overview of the types of data structures and algorithms commonly used in the context of linear regression, logistic regression, and basic ML:

Data Structures:

1. Arrays and Matrices: Arrays and matrices are fundamental for storing and manipulating data. Features and datasets are often represented using arrays or matrices.
2. Lists or Collections: Used for dynamic storage of data during various stages of data processing.
3. Graphs (for model representation): In some cases, graphs or data structures representing mathematical models might be used.

Algorithms:

1. Linear Regression Algorithm: The core algorithm for linear regression involves mathematical operations to find the best-fitting linear model for the training data. This often involves matrix operations and optimization algorithms like gradient descent.
2. Logistic Regression Algorithm: Logistic regression similarly involves mathematical operations, typically using the logistic function. Optimization algorithms like gradient descent are also employed for model training.
3. Data Loading and Parsing Algorithms: Algorithms for efficiently loading and parsing data from various sources, such as CSV files or databases.

API's

Our ML library will be implemented using a collection of modules, functions, and classes that need to be imported into the code at the start. Hence it does not involve making API calls to any external servers.

3. UI Design

User Study

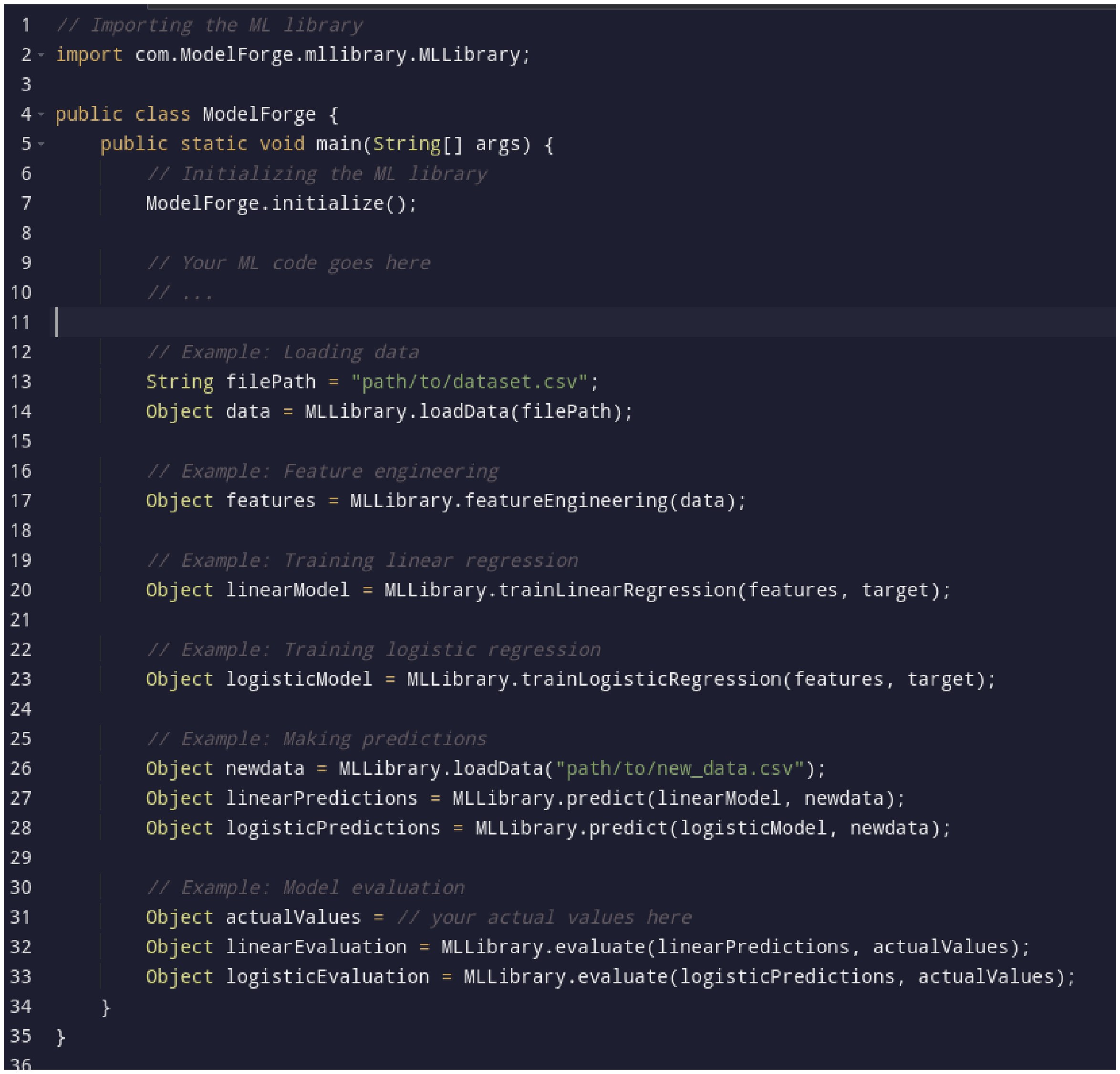
Here the main user groups will be ML-experienced users who have the technical skills to use libraries on their local system. Since this is a software library no explicit UI is required, it is purely based on syntax. Users will have domain knowledge as well as experience with Java.

The user group is primarily students who have already worked with similar tools or working professionals in the applied ML field.

Screen and Navigation design

As mentioned above there is no explicit UI design necessary. The main interaction will be through a programming library rather than a graphical user interface (GUI).

This is a sample snippet of the library functions and how it will be displayed in any code editor.



```java
// Importing the ML library
import com.ModelForge.mllibrary.MLLibrary;

public class ModelForge {
    public static void main(String[] args) {
        // Initializing the ML library
        ModelForge.initialize();

        // Your ML code goes here
        // ...

        // Example: Loading data
        String filePath = "path/to/dataset.csv";
        Object data = MLLibrary.loadData(filePath);

        // Example: Feature engineering
        Object features = MLLibrary.featureEngineering(data);

        // Example: Training linear regression
        Object linearModel = MLLibrary.trainLinearRegression(features, target);

        // Example: Training logistic regression
        Object logisticModel = MLLibrary.trainLogisticRegression(features, target);

        // Example: Making predictions
        Object newdata = MLLibrary.loadData("path/to/new_data.csv");
        Object linearPredictions = MLLibrary.predict(linearModel, newdata);
        Object logisticPredictions = MLLibrary.predict(logisticModel, newdata);

        // Example: Model evaluation
        Object actualValues = // your actual values here
        Object linearEvaluation = MLLibrary.evaluate(linearPredictions, actualValues);
        Object logisticEvaluation = MLLibrary.evaluate(logisticPredictions, actualValues);
    }
}
```