Form 3: Methodology

1. Team No: 9

2. Project Title:

Loan Approval Estimation Deploying Deep Learning and Blockchain Technologies

3. Proposed Method:

The proposed model aims to optimize and enhance the lower approval process by combining the power of deep learning for accurate predictions with tamper proof and transparency afforded by blockchain technology. The integration of user friendly interface ensures accessibility and Usability for fostering trust, confidence in the financial decision making process.

4. Proposed Method illustration:

In the rapidly evolving landscape of financial technology, The integration of deploying models with secure and transparent data storage systems such as blockchain wolves great promise this model aims to leverage deep learning techniques for accurate loan predictions while security and immutability of the underlining data through blockchain technology.

The main objectives are to develop a deployed model for loan approval protection.
Implement a blockchain based data storage system to enhance transparency and guarantees the tamper proof integrity of the data.
Create a user friendly interface for seamless interaction and accessibility.

<u>Deep learning model:</u>

- ➔ Train the model on diverse data sets uprising historical known data including approved and denied cases.
- ➔ Employed techniques such as mean functions and dropouts and batch normalization to enhance the model generalization.
- ➔ Utilize a neural network architecture with multiple layers for feature extraction and decision making.
- ➔ Finetune the hyperparameters through rigorous experimentation which gives an improved accuracy.

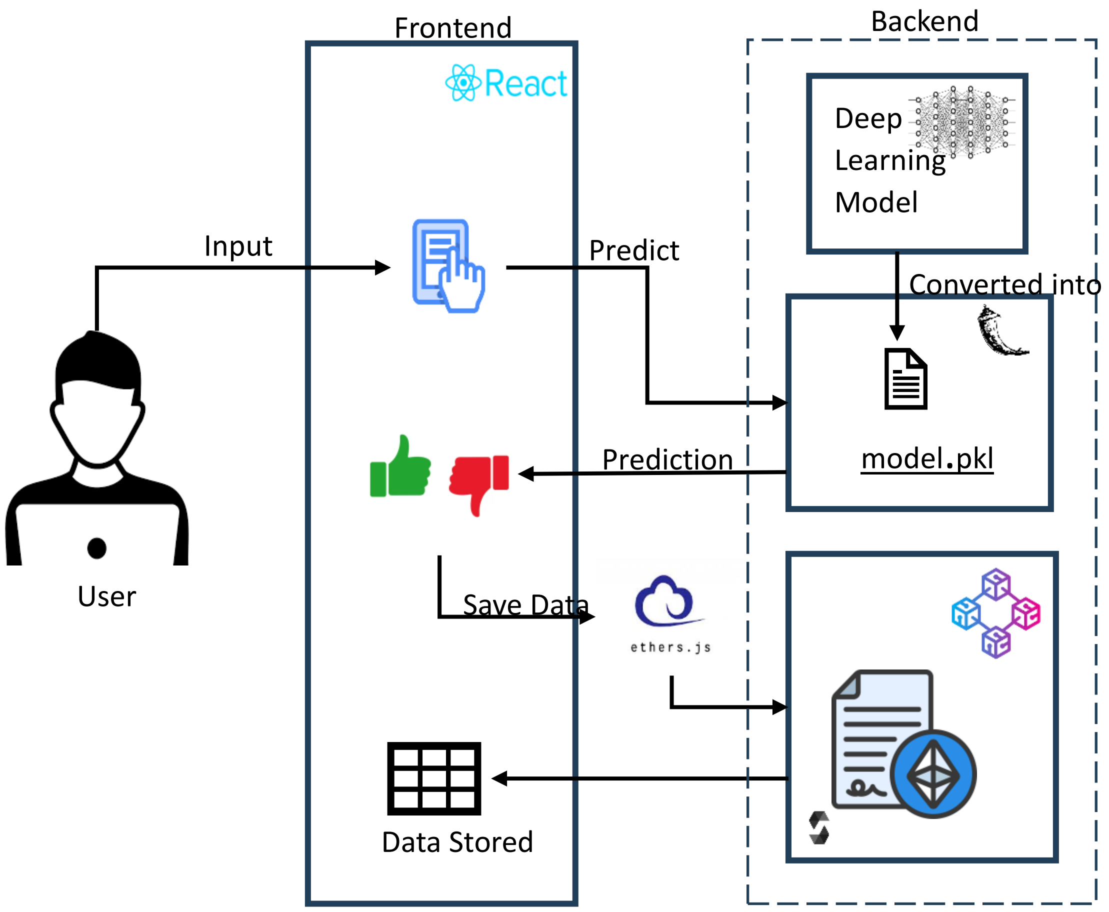
<u>Blockchain integration:</u>

- ➔ Employer permission public blockchain to ensure the data privacy and access control.
- ➔ utilize smart contracts for automating and enforcing the loan approval process on the Brock chain and store the data of the applicant information and approval decisions on the blockchain for transparency and immutability.

<u>User friendly interface:</u>

- ➔ Develop an intuitive web based interface accessible from various phases
- ➔ include a user friendly form for loan application approval and storage.

<u>Architecture</u>:



Smart Contract :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract data_storage {
    uint256 public blockCount; // Add blockCount variable
      string[]  public  applicationNumbers;  //  Declare  and  initialize  the
      applicationNumbers array

    struct Block {
        string person_name;
        string person_income;
        string person_emp_length;
        string loan_grade;
        string application_no;
        string loan_amnt;
        string loan_int_rate;
        string loan_percent_income;
        string cb_person_cred_hist_length;
        string approval;
    }

    mapping(string => Block) public blocks; // Map application number to Block

    function addInformation(
        string memory _person_name,
        string memory _person_income,
        string memory _person_emp_length,
        string memory _loan_grade,
        string memory _application_no,
        string memory _loan_amnt,
        string memory _loan_int_rate,
        string memory _loan_percent_income,
        string memory _cb_person_cred_hist_length,
        string memory _approval
    ) external {
        // Create a new Block with the provided information
        Block memory newBlock = Block({
            person_name: _person_name,
            person_income: _person_income,
            person_emp_length: _person_emp_length,
            loan_grade: _loan_grade,
            application_no: _application_no,
            loan_amnt: _loan_amnt,
            loan_int_rate: _loan_int_rate,
            loan_percent_income: _loan_percent_income,
            cb_person_cred_hist_length: _cb_person_cred_hist_length,
            approval: _approval
        });
        blocks[_application_no] = newBlock;
        blockCount++; // Increment the blockCount
        applicationNumbers.push(_application_no); // Add the application number to
the applicationNumbers array
    }

    function GetAllInformation() external view returns (Block[] memory) {
        // Create an array to store all Blocks
        Block[] memory allBlocks = new Block[](blockCount);

        // Iterate through all application numbers and retrieve the corresponding
Blocks
        for (uint256 i = 0; i < blockCount; i++) {
            allBlocks[i] = blocks[applicationNumbers[i]];
        }

        return allBlocks;
    }

    // Get information for a specific loan application by application number
    function  GetInformationByApplicationNumber(string  memory  _application_no)
external view returns (Block memory) {
        // Retrieve the Block using the provided application number
        return blocks[_application_no];
    }

    // Function to get the total number of stored loan applications
    function GetNumberOfBlocks() external view returns (uint256) {
        return blockCount;
    }
}
```

5. Parameter formulas:

- ➔ Mean of all the columns to fill NA values
- ➔ Implementation of co-relation matrix to check the dependencies of other columns
- ➔ Handling the outliers
- ➔ Normalizing the data to split and train the data.

Guide Signature