A

PROJECT REPORT

ON

AUTOMATED LOAN APPROVAL SYSTEM

Submitted in Partial Fulfillment of the Requirement for the Degree of

BACHELOR OF TECHNOLOGY
IN
CSE/ IT

Submitted by:

Rahul Kapurwan

210060101118

Under the Supervision of:

Swikruti Nayak

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND

INFORMATION TECHNOLOGY COER University, Roorkee

7th, KM Haridwar, National Highway Vardhmanpuram,

Roorkee Rehmadpur, Uttarakhand, 247667

Session: 2024 - 2025

CANDIDATE'S DECLERATION

I hereby declare that this project report titled **AUTOMATED LOAN APPROVAL SYSTEM** is a genuine and original work carried out entirely by me, under the guidance and supervision of **Ms. Swikruti Nayak**, in partial fulfillment of the requirements for the award of the degree.

This work has not been submitted previously, in part or in full, to any university or institution for the award of any degree, diploma, or any other academic qualification. The data, information, analysis, and findings presented in this report are authentic and have been undertaken by me with integrity and in accordance with academic and ethical standards.

I take full responsibility for the content and findings of this project and affirm that all sources of information, data, and literature used have been appropriately acknowledged.

Rahul Kapurwan
210060101118
30/04/2025

CERTIFICATE

This is to certify that the project report entitled **AUTOMATED LOAN APPROVAL SYSTEM**, submitted by **Rahul Kapurwan**, Roll No. **210060101118**, in partial fulfillment of the requirements for the award of the degree, has been carried out under my supervision and guidance.

This work is the result of his own efforts and investigations. The project has been examined and found to meet the academic standards and requirements laid down by the institution. The content of the report, the methodology adopted, and the results presented demonstrate an adequate level of understanding, originality, and application.

Ms Swikruti Nayak
Assistant Professor
Department of Artificial Intelligence and Machine learning
College of Smart Computing
COER University, Roorkee
Date:

Signature
(External Examiner)
Name:
Designation:

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Ms. Swikruti Nayak**, my project guide, for her invaluable guidance, consistent encouragement, and unwavering support throughout the duration of this project. Her insightful suggestions, technical expertise, and timely feedback have been instrumental in the successful completion of this work. I truly appreciate the time and effort she dedicated to mentoring me at every stage of the project.

I am also deeply thankful to all the faculty members of the **Department of Computer science and Application,** whose teachings and knowledge have laid the foundation for this work. Their support and encouragement have played a vital role in shaping my academic journey.

Rahul Kapurwan

[Submission date]

TABLE OF CONTENTS

ABSTRACT

In today's digital era, financial institutions and banks receive thousands of loan applications every day. Manually processing these applications is not only time-consuming but also prone to errors, bias, and inconsistencies. This often leads to delays in loan disbursement and a poor user experience.

The Automated Loan Approval System aims to solve this problem by providing a smart, efficient, and reliable solution that automates the entire loan approval workflow. This system uses machine learning algorithms to predict whether a loan should be approved or rejected based on various financial and personal parameters of the applicant. It incorporates technologies such as TensorFlow, XGBoost, and Random Forest to achieve accurate predictions.

Furthermore, it integrates Optical Character Recognition (OCR) to extract necessary information from user-uploaded documents like Aadhaar Card, PAN Card, Bank Statements, Pay Slips, and more. It also uses Gemini AI by Google to dynamically generate a professional loan agreement document in PDF format upon successful approval.

The user interface is built using Flutter, and Firebase is used for authentication and cloud storage. Overall, this project reduces manual effort, increases accuracy, and streamlines the loan approval process for both applicants and institutions.

INTRODUCTION

Loan approval is an essential and frequent process within the financial sector. Traditionally, the process of approving loans has been slow, paper-based, and reliant on human decision-making, making it prone to delays and bias. With the growing demand for instant financial services and the rise of digital transformation, there is a need for a smarter, faster, and more automated loan approval mechanism.

The Automated Loan Approval System addresses this need by leveraging machine learning and mobile technologies to digitize the loan approval pipeline. Users enter their personal, employment, financial, and loan-related details into a Flutter mobile app, which sends the data to a Flask API for prediction. The backend uses trained machine learning models — including Neural Networks (TensorFlow), XGBoost, and Random Forest — to analyze the data and determine whether the applicant qualifies for the loan.

Upon approval, a professional loan agreement document is automatically generated using Gemini AI, summarizing the loan terms and confirming the approval. The application uses Firebase Authentication for secure sign-in and Firestore to store application data and track statuses.

This project demonstrates how a combination of mobile development, machine learning, and generative AI can simplify and optimize one of the most common processes in finance — loan approval — offering real-time decisions and better user experience.

LITERATURE REVIEW

The rise of technology in the financial sector has completely changed how loans are processed and approved. One of the most important developments in this space is the Automated Loan Approval System (ALAS). These systems use algorithms and data to assess whether someone is eligible for a loan, reducing the need for manual review by a human loan officer. Their main purpose is to make the process faster, more cost-effective, and fairer, while also making loans more accessible to people who may have been excluded by traditional methods. ALAS offer consistent decisions and reduce operational costs while supporting financial inclusion for people without traditional credit histories.

An automated loan system works by first collecting all the necessary details from an applicant, including personal, employment, and financial information. After the data is collected, the system evaluates the applicant's creditworthiness using credit scoring models—these might range from basic rule-based logic and logistic regression to more sophisticated machine learning algorithms like decision trees, random forests, and XGBoost. These advanced techniques allow systems to discover complex data patterns that may not be visible to humans and make accurate predictions regarding repayment capabilities. Once the credit score is generated, the system moves to the decision-making phase: it either approves, rejects, or flags the application for further manual review. The final step includes sending automated notifications to the applicant regarding the decision.

A wide range of technologies supports this process. Traditional systems relied on rules-based logic or statistical models, but modern implementations use machine learning pipelines that can process large volumes of data in real-time. These systems can automatically adjust to changing data distributions, making them suitable for dynamic financial environments. They also utilize features such as applicant behavior, transaction history, and alternate data sources to build robust credit profiles.

The benefits of automation in lending are numerous. These systems speed up processing time, lower costs, reduce human bias, and offer high scalability. More importantly, they enhance financial inclusion by evaluating alternative data such as utility bills, mobile payments, or online transaction history—making credit accessible to people with little or no credit bureau history. However, several critical challenges remain. One major concern is algorithmic bias—if the training data reflects societal or historical discrimination, the model can inherit and reproduce those biases. Additionally, transparency becomes a concern, especially with complex

machine learning models whose decisions can be difficult to interpret or explain. This issue is especially relevant in regulated sectors like finance, where explainability is often a legal requirement.

Data privacy and protection are also vital. As these systems handle highly sensitive personal and financial information, they must comply with data protection laws like the General Data Protection Regulation (GDPR). GDPR emphasizes transparency, fairness, and accountability in automated decision-making processes, pushing financial institutions to implement robust security and explainability protocols.

To assess the effectiveness of these systems, both technical and business performance metrics are employed. Technical metrics include accuracy, precision, recall, and F1-score, which measure how well the model predicts loan repayment or default. Business-level metrics such as approval rates, default rates, and average processing times are also crucial for determining the system's real-world impact. Fraud detection capabilities are evaluated based on how effectively the system identifies risky applications without mislabeling genuine ones, balancing true positives with false positives to avoid undue rejections.

Current trends show a rapid shift toward deeper AI integration, use of alternative data sources, and adoption of Explainable AI (XAI). XAI tools help stakeholders—like applicants, banks, and regulators—understand how loan decisions are made, building user trust and ensuring compliance with fairness and transparency standards. As a result, many regulatory authorities now classify AI-powered loan scoring as a "high-risk application", requiring stricter guidelines, documentation, and monitoring to ensure ethical deployment.

In summary, research indicates that automated loan approval systems are fundamentally reshaping the lending landscape. They enable faster decisions, greater scalability, and broader inclusion, but they also bring new challenges around fairness, privacy, and accountability. As this field matures, it is essential that these systems are built and regulated responsibly—ensuring they are transparent, equitable, and aligned with ethical and legal standards.

SYSTEM ANALYSIS

Existing Systems

In the current financial ecosystem, many institutions — including banks, NBFCs (Non-Banking Financial Companies), and microfinance platforms — rely heavily on traditional or semi-automated systems to handle loan processing. These systems are primarily designed to gather applicant information and perform manual verification, but they often fall short in terms of speed, scalability, and intelligence. As a result, loan disbursement becomes a time-consuming process, prone to delays, inconsistencies, and human error.

Below are the major types of existing systems being used:

a) Manual Loan Processing

- This is the most traditional and still widely used method, especially in rural and semi-urban branches.
- Applicants must **physically visit** the bank or financial institution.
- Loan forms and supporting documents (like salary slips, ID proofs, bank statements) are submitted **in person**.
- A loan officer is responsible for verifying the documents manually, assessing risk, and making a decision based on pre-defined rules or personal experience.
- **Limitations**:
    - High **human dependency**, leading to subjective decisions.
    - Prone to **errors**, misjudgment, or oversight.
    - The entire process can take **several days to weeks**, frustrating users.
    - Not scalable — increasing applicants = increased workload.

b) Web-Based Loan Portals (Semi-Automated Systems)

- These are modern portals where users can **apply online** through the institution's website or app.
- The form captures all basic details like personal information, employment, and income.

- However, once the form is submitted, **backend staff still manually validate** the uploaded documents.
- Final approval is made by a human based on experience or internal policies.
- **Limitations**:
    - Still involves **manual verification**, defeating the purpose of digital speed.
    - High chances of **human delays and inconsistencies**.
    - Does not adapt or learn from **historical approval data**.

c) Rule-Based Automated Systems

- These systems automate decision-making using a **fixed set of rules**. For example:
    - If age > 25 and credit score > 700 → Approve
    - If salary < ₹20,000 and loan amount > ₹5,00,000 → Reject
- They offer **basic automation** but lack the ability to evolve.
- **Limitations**:
    - Rigid logic: cannot **adapt to patterns** in user behavior or risk over time.
    - Doesn't consider **nuanced combinations** of features that real-world machine learning can handle.
    - May either approve **risky loans** or reject **eligible candidates**, lowering efficiency and fairness.

Proposed System: Automated Loan Approval System (ALAS)

The Automated Loan Approval System (ALAS) is designed to address the shortcomings of traditional and semi-automated loan processing systems. It provides a complete end-to-end solution that collects loan application data, intelligently evaluates eligibility using machine learning, and generates official loan agreement documents automatically — all from a mobile application. This system integrates modern technologies like machine learning, cloud databases, and generative AI to offer real-time decision-making, transparency, and efficiency for both applicants and institutions.

Key Features and Functionality:

a) **Mobile Application (Flutter)**

- Users can apply for a loan directly from a simple and intuitive mobile app.
- The app collects all necessary personal, employment, financial, and loan-related information.
- Documents such as ID proof, salary slip, and bank statement can be uploaded digitally.

b) **Secure Authentication (Firebase)**

- User sign-up and login are securely handled using **Firebase Authentication**.
- Each user's application data is securely linked to their unique ID, ensuring data privacy and individual tracking.

c) **Real-Time Loan Prediction (Machine Learning Models)**

- A trained machine learning pipeline (TensorFlow, XGBoost, Random Forest) analyzes the user's input.
- These models are trained on real-world loan data to identify patterns of approved and rejected loans.
- The backend API (developed using **Flask**) makes fast and reliable predictions on whether to approve or reject the application.

d) **Auto-Generated Loan Agreement (Gemini AI)**

- If the loan is approved, a professional **PDF loan agreement** is generated automatically using **Gemini AI**.
- The document includes borrower information, loan amount, repayment terms, and legal clauses, formatted formally and clearly.

e) **Cloud Storage and Dashboard (Firebase Firestore)**

- All applications, loan statuses, and PDF documents are stored securely in **Firebase Firestore**.
- The user can view their application history, approval status, and download agreements from the app.

Benefits of the Proposed System:

- **Faster processing**: Loan approval decisions are delivered in seconds.
- **Accurate and unbiased**: Data-driven machine learning removes human bias.
- **Instant documentation**: Loan agreements are generated immediately after approval.
- **User-friendly**: Fully mobile-based, accessible to all users without needing to visit a bank.
- **Cloud-integrated**: Secure and scalable storage of user data and applications.
- **Safe and authenticated**: Only verified users can apply or access their information.

The Automated Loan Approval System bridges the gap between traditional loan workflows and modern technology. By integrating intelligent automation and AI-driven document generation, it creates a smarter, faster, and more secure way to process loan applications — benefiting both users and financial institutions alike.

SYSTEM DESIGN

The Automated Loan Approval System (ALAS) is designed as a modern, intelligent, and scalable application that mimics the operations of a financial institution's loan department — but in a fully digital and automated way.

The system is built with a layered, modular architecture. Each layer (Frontend, Backend, Machine Learning, Cloud) is responsible for a specific set of tasks. These modules interact with each other seamlessly to offer a complete and automated loan processing pipeline — from data collection to decision-making and contract generation.

The design ensures that the system is:

- User-centric (mobile-friendly, easy to use),
- Efficient (real-time predictions),
- Accurate (ML-driven decision making),
- Secure (with Firebase auth and cloud storage)

Detailed Component Design:

A. Frontend Layer – Flutter Mobile Application

- Built using Flutter for cross-platform support (iOS and Android).
- Collects data in a step-wise or single-form layout:
    - Personal Details: name, DOB, ID number, contact.
    - Employment Info: organization, income, job duration.
    - Financial Info: bank details, credit score, expenses.
    - Loan Details: amount, purpose, repayment duration.
- Integrates with Firebase Authentication to ensure secure sign-up and login.
- Uses HTTP POST to send data to the backend for prediction.
- Displays prediction results and status dynamically.
- Shows user dashboard (with past applications, application status, loan documents, etc.).

B. Backend Layer – Flask REST API

- Developed in Python using the Flask microframework.
- Responsible for:
    - Receiving and validating user data.
    - Preprocessing input for ML models.
    - Making predictions using trained ML models.
    - Handling rejection criteria manually (e.g., low credit score).
    - Generating PDF loan agreement using Gemini AI (if approved).
    - Saving application data and results to Firebase Firestore.

C. ML Prediction Layer

- Consists of multiple trained ML models:
    - TensorFlow Neural Network - main classifier.
    - XGBoost - for boosting-based classification.
    - Random Forest - for ensemble comparison.
- Models are trained on historical loan data using:
    - Feature engineering: credit ratio, employment ratio, etc.
    - Balancing techniques: SMOTE-Tomek for class imbalance.
    - Evaluation metrics: accuracy, precision, recall, F1-score.
- Predictions are thresholded (e.g., if output > 0.6 → Approved).

D. Cloud Infrastructure – Firebase

- Firebase Authentication:
    - Handles secure user registration, login, and session management.
    - Each user is uniquely identified via UID.
- Cloud Firestore:

- Stores all user loan applications.
- Each application includes: user UID, loan amount, intent, credit score, application date, status, and PDF URL.

- Firebase Storage (optional):
  - Can store user-uploaded documents and generated PDFs.
  - Ensures file-level security and access control.

E. Document Generation – Gemini AI

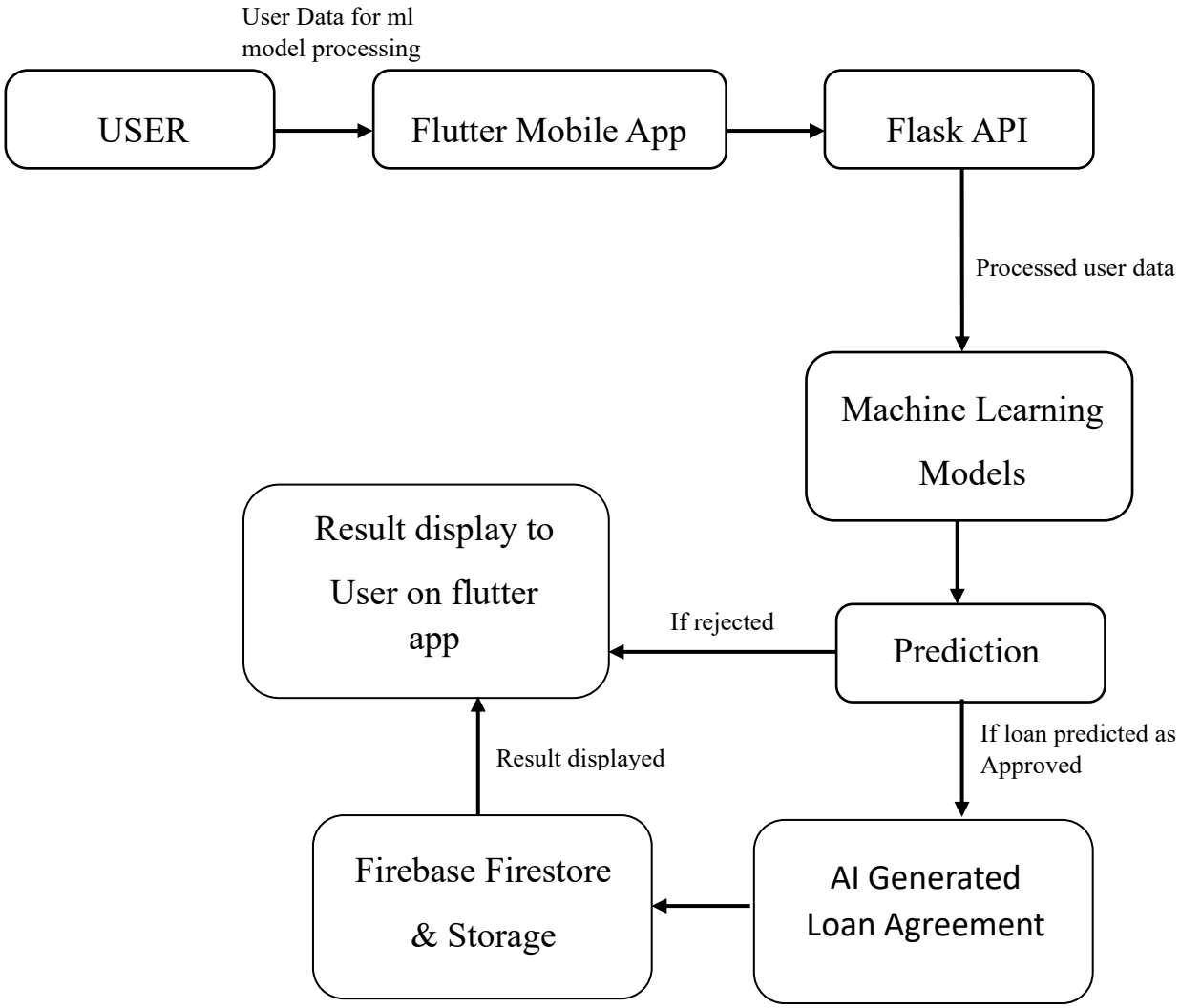
- Google's Gemini API is used to dynamically generate loan documents upon approval.
- PDF includes:
  - User name, ID, contact
  - Loan amount and purpose
  - Repayment period
  - Terms and conditions
- Output is uploaded to Firestore and linked in the user dashboard.

System Architecture Diagram

The System Architecture Diagram of the Automated Loan Approval System (ALAS) illustrates how different components of the application interact with each other to process a loan application efficiently.

It begins with the User submitting their details through the Flutter Mobile App. This data is sent to the Flask API, which serves as the backend controller. The API processes the input and passes it to the Machine Learning Models for prediction. Based on the result, if the loan is approved, the Gemini AI module generates a personalized PDF loan agreement. Finally, all loan-related data, including the status and generated document, is securely stored in Firebase Firestore & Storage. This design ensures real-time decision-making, secure data handling, and smooth integration across all system layers.

Diagram: -



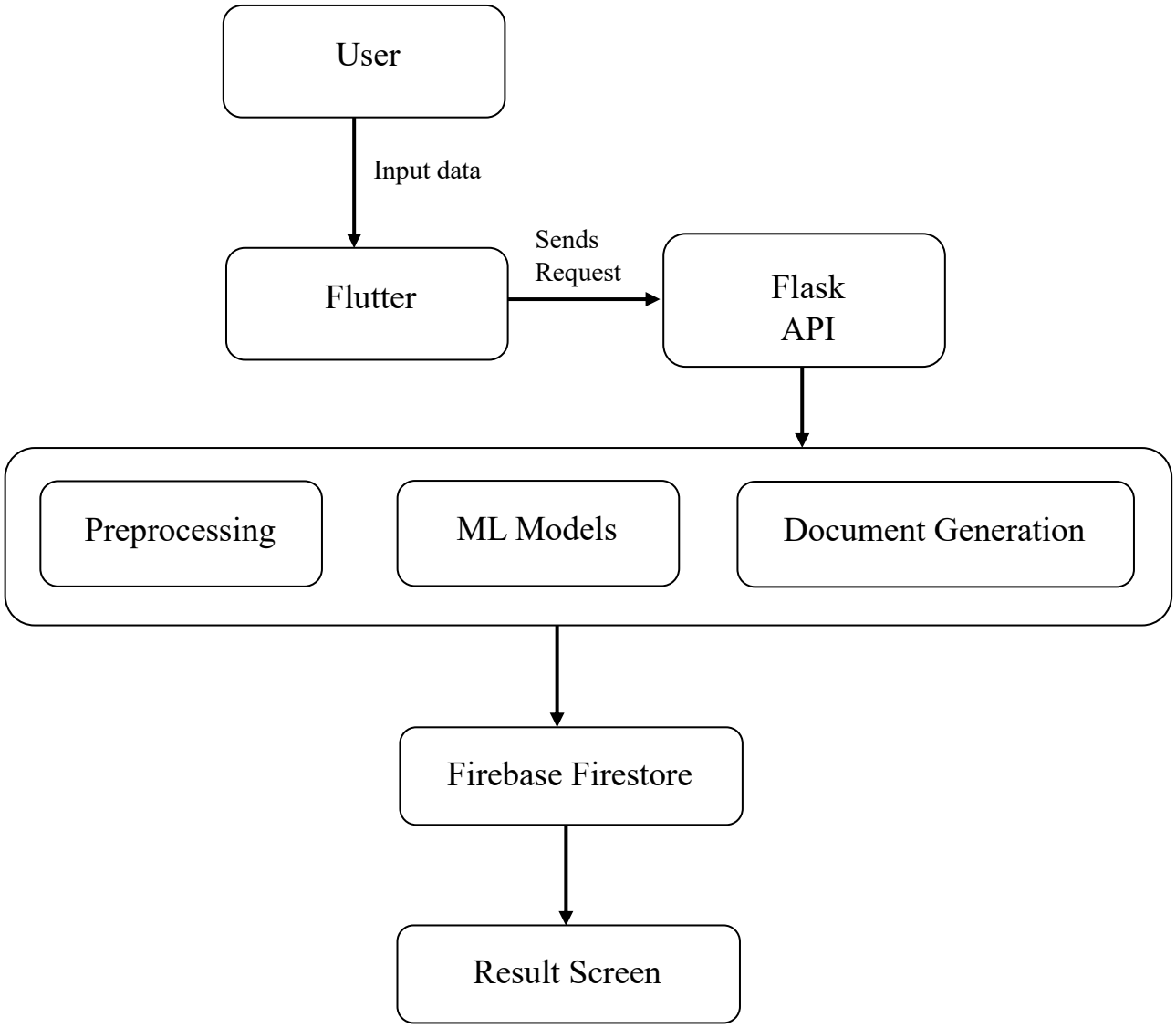
Data Flow Diagram

The Data Flow Diagram (DFD) provides a simplified visual representation of how data moves through the Automated Loan Approval System (ALAS).

It starts with the User, who inputs their personal, financial, and loan-related details into the Flutter mobile application. This data is then sent to the Flask API, which acts as the processing hub. The API performs input validation and preprocessing, then forwards the refined data to the Machine Learning Models for prediction. Depending on the result, if the application is approved, the Gemini AI module is triggered to generate a loan agreement. The final output —

including loan status and document link — is stored in Firebase Firestore, ensuring that all information is securely saved and easily retrievable for dashboard and history views.

Data flow diagram:



Entity-Relationship Diagram (ERD)

The Entity Relationship Diagram (ERD) of the Automated Loan Approval System (ALAS) outlines the core data structure of the application, representing how data entities such as Users, Loan Applications, and Loan Documents are related to each other.

At the core of the system:

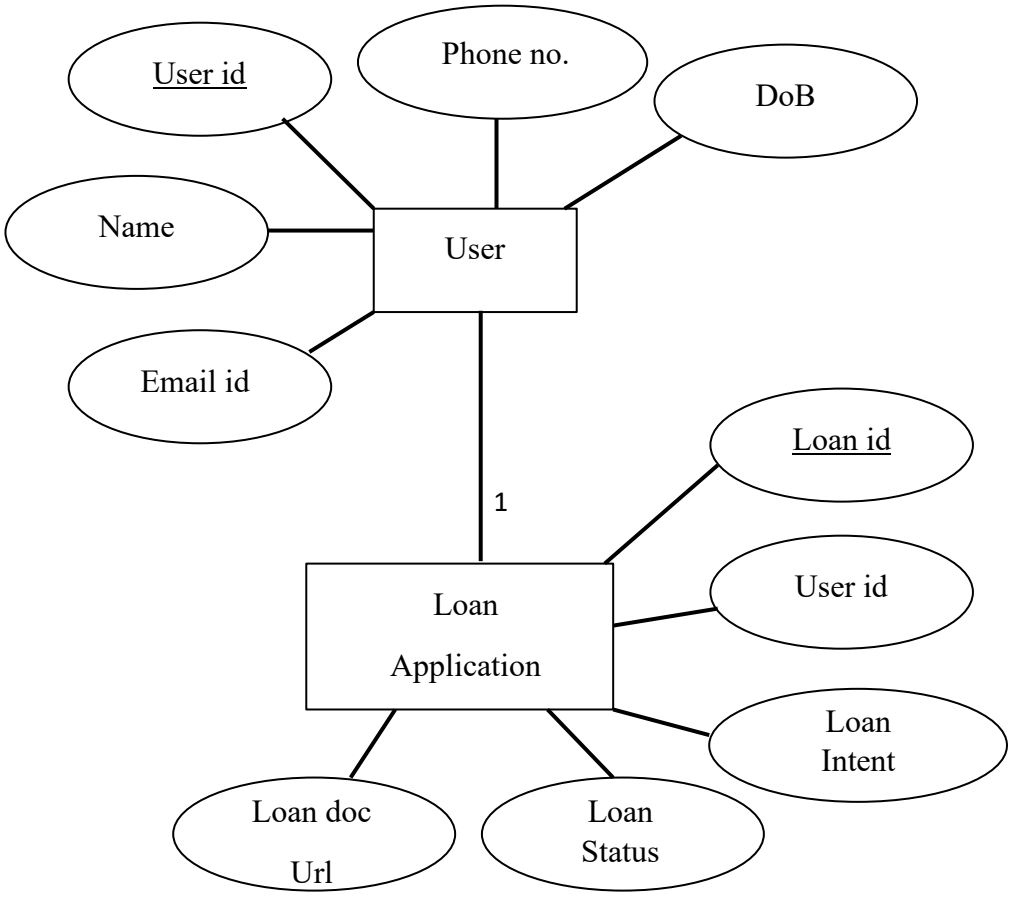
- Each User has a unique account (identified by a UID) and can submit multiple loan

applications.

- Each Loan Application is tied to a specific user and stores detailed information about the loan request (amount, intent, credit score, income, etc.), as well as its status and application date.
- If the loan is approved, a Loan Document (a PDF agreement generated via Gemini AI) is associated with the application, and its metadata (like URL and timestamp) is stored.

This relationship ensures that the system can track which user submitted which loan applications and manage the corresponding legal documents efficiently.

Entity-Relationship Diagram:



IMPLEMENTATION

The Implementation phase of the Automated Loan Approval System (ALAS) focuses on transforming the system design into a fully functional software application. This phase brings together all the selected technologies — including mobile development, machine learning, cloud services, and AI integration — and orchestrates them to deliver a seamless, intelligent, and real-time loan approval experience.

This section outlines the various technologies used and provides a breakdown of the key modules developed during implementation, explaining how each component contributes to the complete workflow — from collecting user data to generating loan decisions and agreements.

Technologies Used

To build a reliable, user-friendly, and intelligent loan approval system, a wide range of modern technologies was employed. Each technology served a specific purpose — from designing the user interface to hosting the backend, handling machine learning, and generating loan documents.

1. **Flutter:**

   Flutter is an open-source UI toolkit created by Google that allows developers to build beautiful, fast, and cross-platform applications using a single codebase. It supports Android, iOS, web, and desktop apps, which makes it highly efficient for developers who want to build apps for multiple platforms without writing separate code for each. In our project, Flutter was used to build the mobile interface where users can register, log in, and apply for loans. One of the biggest advantages of Flutter is its rich set of customizable widgets, which made it easy to design intuitive and responsive input forms and dashboards. It also offers excellent performance because it compiles to native code. With Flutter, we were able to develop the entire frontend quickly and with great flexibility.

2. **Python Flask:**

   Flask is a lightweight and flexible web framework written in Python. It is primarily used to build APIs and backend services. Flask was chosen for this project because it is simple, powerful, and integrates seamlessly with Python-based machine learning libraries. The Flask backend in our system handles important tasks like receiving data from the Flutter app, preprocessing the input, loading the trained machine learning models, making predictions, and returning the result to the frontend. Flask's minimalistic design allows us

to build the entire backend with full control over routing, logic, and integration — making it a perfect fit for projects involving machine learning and custom workflows.

3. **TensorFlow:**

   TensorFlow is a popular open-source machine learning framework developed by Google. It is widely used to build and train deep learning models such as neural networks. In our system, TensorFlow is used to train a model that predicts whether a loan should be approved based on various applicant features like age, income, credit score, loan amount, and more. The model learns patterns from historical loan data and uses this knowledge to make real-time predictions. TensorFlow provides tools for training, evaluating, and saving models efficiently, and supports seamless deployment in a Flask-based API.

4. **XGBoost & Random Forest:**

   In addition to the deep learning model built using TensorFlow, the system also incorporates two popular and powerful machine learning algorithms — XGBoost and Random Forest — to strengthen the accuracy and reliability of predictions.

   a) **XGBoost (Extreme Gradient Boosting)**

      XGBoost is an advanced machine learning algorithm based on gradient boosting. It builds an ensemble of decision trees where each new tree corrects the errors made by the previous ones. This step-by-step learning process allows it to capture complex patterns in structured data. XGBoost is known for its high speed and predictive accuracy, making it a top choice for real-world data science competitions and financial risk assessments.

      In this project, XGBoost is used as a secondary prediction model to validate the result provided by the neural network. It processes applicant data — such as credit score, income, loan amount, etc. — and outputs whether the loan should be approved. XGBoost adds an extra layer of confidence by catching subtle relationships in the data that even neural networks might miss.

   b) **Random Forest**

      Random Forest is another widely used ensemble learning method that builds multiple decision trees and combines their outputs to make a final prediction. Each tree is trained on a random subset of the data, and the results are averaged or voted on, which helps reduce overfitting and improves generalization.

In the loan approval system, Random Forest serves as an additional classifier to confirm prediction consistency. It provides robustness by learning different combinations of features in the dataset. Its simplicity and interpretability make it an ideal fallback model when further verification is needed in sensitive decision-making tasks like loan processing.

5. **Firebase Authentication**

   Firebase Authentication is a service provided by Google Firebase that allows developers to implement secure user authentication in their applications. It supports multiple authentication methods, including email/password, Google sign-in, and phone number login. For this project, Firebase Authentication is used to manage user registration and login securely. Each user is assigned a unique ID (UID) after signup, which helps us associate their loan applications and data specifically to them. Firebase also provides built-in tools for password reset, session management, and security — eliminating the need to build an authentication system from scratch.

6. **Firebase Firestore**

   Cloud Firestore is a real-time NoSQL cloud database from Firebase. It stores data in collections and documents, and automatically syncs data across devices. In this project, Firestore is used to store all user-related information, including loan applications, their statuses (approved, rejected, pending), and generated document links. It allows us to easily retrieve and display the most recent applications, count total submissions, and show individual loan records per user. Firestore's real-time data syncing is particularly helpful for updating dashboards and recent activity feeds.

7. **Render.com**

   Render is a cloud platform that allows developers to deploy and host web services, APIs, and static sites easily. In our project, the Flask backend and machine learning models are deployed on Render, making the prediction API publicly accessible from the Flutter frontend. Render handles all server management, HTTPS configuration, and scaling automatically, so we don't need to manage infrastructure manually. It supports continuous deployment via GitHub, which makes development and updates seamless.

8. **Git & GitHub**

   Git is a version control system that tracks changes in source code, and GitHub is an online platform for storing and collaborating on Git repositories. In this project, Git is used to manage and track code changes, while GitHub is used to collaborate, back up code, and integrate with hosting services like Render. It helps ensure that the project is well-

organized, and makes it easier to revert changes, add features, or collaborate with team members.

9. **Gemini AI Model**

   Gemini AI is Google's generative AI model capable of producing human-like responses based on natural language prompts. In this project, Gemini AI is used to dynamically generate loan agreement documents once a loan application is approved. It takes details like user name, loan amount, purpose, interest rate, and repayment terms, and produces a legally worded contract. This automated document generation replaces the need for manual paperwork, ensuring that each user gets a professional PDF agreement tailored to their application — saving time and improving user experience.

Coding & Modules

The implementation of the Automated Loan Approval System (ALAS) has been structured into multiple functional modules, each responsible for a specific part of the application's overall behavior. This modular approach ensures that the system is organized, easy to maintain, and scalable for future upgrades.

Each module — from user authentication to loan application, machine learning prediction, and document generation — plays a crucial role in ensuring that the system performs smoothly from start to finish. The modules are carefully integrated so that user input flows seamlessly through the system, passing through validation, processing, prediction, and finally storage.

This section describes the key modules developed during implementation, outlining their responsibilities and how they contribute to the complete workflow of the system.

Code: Flutter

Main: - main.dart

```
import 'package:firebase_core/firebase_core.dart';

import 'package:flutter/material.dart';

import 'package:loanapp/login.dart';

Future<void> main() async {

  WidgetsFlutterBinding.ensureInitialized();
```

```
  await Firebase.initializeApp();
  runApp(const MyApp());
}
class MyApp extends StatefulWidget {
  const MyApp({super.key});
  @override
  State<MyApp> createState() => _MyAppState();
}
class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      // theme: ThemeData.light(useMaterial3: true),
      home: login(),
    );}}
```

Login Screen: - login.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:loanapp/dashboardscreenog.dart';
import 'package:loanapp/forgotpasswordscreen.dart';
import 'package:loanapp/signupscreen.dart';
// ignore: camel_case_types
class login extends StatefulWidget {
```

```
  const login({super.key});

  @override

  State<login> createState() => _loginState();

}

// ignore: camel_case_types

class _loginState extends State<login> {

  final TextEditingController _emailController = TextEditingController();

  final TextEditingController _passwordController = TextEditingController();

  void showError(String msg) {

                ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(msg)));

  }

  Future<void> loginUser() async {

    final email = _emailController.text.trim();

    final password = _passwordController.text.trim();

    if (email.isEmpty || password.isEmpty) {

      showError('Please enter email and password');

      return;

    }

    try {

      await FirebaseAuth.instance

          .signInWithEmailAndPassword(email: email, password: password);

      // ignore: use_build_context_synchronously

      Navigator.push(context,
```

```
        MaterialPageRoute(builder: (context) => const Dashboardscreenog()));
    } on FirebaseAuthException catch (e) {
      showError(e.message ?? 'Login failed');
    } catch (e) {
      showError('An unexpected error occurred');
    }
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.blue[200],
      body: SafeArea(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const Text(
                "Login",
                style: TextStyle(fontSize: 28, fontWeight: FontWeight.bold),
              ),
              const SizedBox(height: 20),
              Container(
                padding: const EdgeInsets.all(20.0),
```

```
height: 350,
width: 850,
decoration: BoxDecoration(
   color: Colors.white,
   borderRadius: BorderRadius.circular(20),
   boxShadow: const [
     BoxShadow(
        color: Colors.black26,
        blurRadius: 10,
        spreadRadius: 2),
   ]),
child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    const Text("Email Id"),
    TextField(
      controller: _emailController,
      decoration: const InputDecoration(
        border: OutlineInputBorder(),
        hintText: "Enter email",),
    ),
    const SizedBox(height: 15,),
    const Text("Password"),
```

```
TextField(
  controller: _passwordController,
  obscureText: true,
  decoration: const InputDecoration(
    border: OutlineInputBorder(),
    hintText: "Enter password",
  ),
),
const SizedBox(height: 15,),
InkWell(
  onTap: () {
    Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) =>
                const Forgotpasswordscreen()));
  },
  child: const Text("Forgot Password"),
),
Center(
  child: ElevatedButton(
    onPressed: () {
      loginUser();},
```

```
                child: const Text("Login"),),
            ),
            Center(
              child: InkWell(
                onTap: () {
                  Navigator.push(
                      context,
                      MaterialPageRoute(
                          builder: (context) => const Signupscreen()));
                },
                child:
                    const Text("Don't have an account? Create account"),),),],
          ),),],
      ),),),);
  }
}
```

Signup Screen: - signupscreen.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:loanapp/login.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
class Signupscreen extends StatefulWidget {
  const Signupscreen({super.key});
```

```
  @override
  State<Signupscreen> createState() => _SignupscreenState();
}
class _SignupscreenState extends State<Signupscreen> {
  final _formKey = GlobalKey<FormState>();
  final firstNameController = TextEditingController();
  final lastNameController = TextEditingController();
  final emailController = TextEditingController();
  final phoneController = TextEditingController();
  final passwordController = TextEditingController();
  final confirmPasswordController = TextEditingController();
  // ignore: non_constant_identifier_names
  final DoBController = TextEditingController();
  bool isLoading = false;
  @override
  void dispose() {
    firstNameController.dispose();
    lastNameController.dispose();
    emailController.dispose();
    phoneController.dispose();
    passwordController.dispose();
    confirmPasswordController.dispose();
```

```
  super.dispose();
}
void showMessage(String message) {
  ScaffoldMessenger.of(context)
      .showSnackBar(SnackBar(content: Text(message)));
}
Future<void> registerUser() async {
  if (!_formKey.currentState!.validate()) return;
  setState(() => isLoading = true);
  try {
    final email = emailController.text.trim();
    final password = passwordController.text.trim();
    UserCredential userCredential = await FirebaseAuth.instance
        .createUserWithEmailAndPassword(email: email, password: password);

    final uid = userCredential.user!.uid;
    //store users data in database
    await FirebaseFirestore.instance.collection('User').doc(uid).set({
      'firstName': firstNameController.text.trim(),
      'lastName': lastNameController.text.trim(),
      'DoB': DoBController.text.trim(),
      'email': emailController.text.trim(),
      'phone': phoneController.text.trim(),
```

```
    'password': passwordController.text.trim(),
    'createdAt': FieldValue.serverTimestamp(),
   });

   showMessage("Account created successfully!");
   Navigator.pushReplacement(
    // ignore: use_build_context_synchronously
    context,
    MaterialPageRoute(builder: (context) => const login()),
   );
  } on FirebaseAuthException catch (e) {
   showMessage(e.message ?? "Something went wrong");
  }catch (e) {
  showMessage("Error: $e");
  } finally {
   setState(() => isLoading = false);
  }
 }
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Colors.white,
   appBar: AppBar(
```

```
  backgroundColor: Colors.blue[100],
  centerTitle: true,
  title: const Text("Sign up"),
 ),
body: SafeArea(
  child: SingleChildScrollView(
    padding: const EdgeInsets.all(16.0),
    child: Form(
      key: _formKey,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const SizedBox(height: 20),
          buildTextField("First Name", firstNameController),
          buildTextField("Last Name", lastNameController),
          buildTextField("Date-of-Birth", DoBController),
          buildTextField("Email", emailController, isEmail: true),
          buildTextField("Phone Number", phoneController, isPhone: true),
          buildTextField("Password", passwordController,
              isPassword: true),
          buildTextField("Confirm Password", confirmPasswordController,
              isPassword: true, validator: (value) {
            if (value != passwordController.text) {
```

```
            return "Passwords do not match";
          }
          return null;
        }),
        const SizedBox(height: 20),
        Center(
          child: ElevatedButton(
            onPressed: isLoading ? null : registerUser,
            child: Text(isLoading ? "Registering..." : "Register"),
          ),
        ),
        const SizedBox(height: 10),
        Center(
          child: TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: const Text("Already have an account? Login"),),
        ),],),
      ),),),
    );
  }
  Widget buildTextField(
```

```
  String label,
  TextEditingController controller, {
  bool isPassword = false,
  bool isEmail = false,
  bool isPhone = false,
  String? Function(String?)? validator,
 }) {
  return Padding(
    padding: const EdgeInsets.symmetric(vertical: 8),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        RichText(
          text: TextSpan(
            text: label,
            style: const TextStyle(color: Colors.black, fontSize: 16),
            children: const [
              TextSpan(text: ' *', style: TextStyle(color: Colors.red)),
            ],
          ),
        ),
        const SizedBox(height: 6),
        TextFormField(
```

```
                controller: controller,
                obscureText: isPassword,
                keyboardType: isPhone
                    ? TextInputType.phone
                    : (isEmail ? TextInputType.emailAddress : TextInputType.text),
                decoration: const InputDecoration(
                  border: OutlineInputBorder(),
                  hintText: "Required",
                ),
                validator: validator ??
                    (value) {
                      if (value == null || value.trim().isEmpty) {
                        return "$label is required";
                      }
                      return null;
                    },
              ),],),
        );}
    }
```

Reset Password Screen: - forgotscreen.dart

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
class Forgotpasswordscreen extends StatefulWidget {
```

```
 const Forgotpasswordscreen({super.key});
 @override
 State<Forgotpasswordscreen>createState() => _ForgotpasswordscreenState();
}
class _ForgotpasswordscreenState extends State<Forgotpasswordscreen> {
 // ignore: non_constant_identifier_names
 final TextEditingController EmailControllor = TextEditingController();
 final formKey = GlobalKey<FormState>();

 Future<void> sendResetMail() async {
  try {
   await FirebaseAuth.instance
     .sendPasswordResetEmail(email: EmailControllor.text.trim());

   // ignore: use_build_context_synchronously
   ScaffoldMessenger.of(context)
     .showSnackBar(const SnackBar(content: Text("Password reset link")));
  } catch (e) {
   print("Error: $e");
   // ignore: use_build_context_synchronously
   ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Error: ${e.toString()}')),
   );
```

```
  }
}
@override
Widget build(BuildContext context) {
 return Scaffold(
   appBar: AppBar(
     backgroundColor: Colors.blue[100],
     centerTitle: true,
     title: const Text("Reset PassWord!!!"),
   ),
   backgroundColor: Colors.blue[100],
   body: Padding(
     padding: const EdgeInsets.all(8.0),
     child: Center(
       child: Container(
         padding: const EdgeInsets.all(20),
         height: 350,
         decoration: BoxDecoration(
           color: Colors.white, // white card
           borderRadius: BorderRadius.circular(16),
           boxShadow: const [
             BoxShadow(
               color: Colors.black12,
```

```
        blurRadius: 10,
        offset: Offset(0, 5),
      ),
    ],
  ),
  child: Form(
    key: formKey,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Text(
          "Enter register email id to get a reset link! ",
          style: TextStyle(fontSize: 20, color: Colors.black),
          textAlign: TextAlign.center,
        ),
        const SizedBox(
          height: 20,
        ),
        TextFormField(
          controller: EmailControllor,
          decoration: const InputDecoration(
            border: OutlineInputBorder(),
            hintText: "Enter Email ID",
```

```
    ),
    validator: (value) {
      if (value == null || value.isEmpty) {
        return 'Please enter your email';
      }
      // check if the email has the correct format
      else if (!value.contains("@")) {
        return 'Please enter a valid email address';
      }
      return null;
    },
  ),
  const SizedBox(
    height: 20,
  ),
  ElevatedButton(
      onPressed: () {
        if (formKey.currentState!.validate()) {
          sendResetMail();
        }
      },
      child: const Text("Send Reset Link"))
],)),),),
```

```
    ),);}
  }
```

Dashboard Screen: - dashboardscreenog.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:loanapp/applyLoanScreen.dart';
import 'package:loanapp/modules/ActivityTile.dart';
import 'package:loanapp/modules/DashboardCards.dart';
import 'package:loanapp/modules/ActionButton.dart';
import 'package:loanapp/resultScreen.dart';
// import 'package:loanapp/modules/myapplicationscreen.dart';

class Dashboardscreenog extends StatefulWidget {
  const Dashboardscreenog({super.key});
  @override
  State<Dashboardscreenog> createState() => DashboardscreenogState();
}
class DashboardscreenogState extends State<Dashboardscreenog> {
  String? userName;
  bool isLoading = true;
  int total = 0;
  int approved = 0;
```

```
int rejected = 0;

int pending = 0;

List<Map<String, dynamic>> recentApp = [];

@override

void initState() {

 super.initState();

 fetchDashboardData();

}

Future<void> fetchDashboardData() async {

 try {

  final user = FirebaseAuth.instance.currentUser;

  if (user == null) {

   return;

  }

  final userDoc = await FirebaseFirestore.instance

     .collection('User')

     .doc(user.uid)

     .get();

  final applicationsQuery = await FirebaseFirestore.instance

     .collection('loan_applications')

     .where('uid', isEqualTo: user.uid)

     .orderBy('date_applied', descending: true)

     .get();
```

```
List<Map<String, dynamic>> recent = [];
int a = 0, r = 0, p = 0;

// DocumentSnapshot userdoc = await FirebaseFirestore.instance
//     .collection('User')
//     .doc(user.uid)
//     .get();

for (var doc in applicationsQuery.docs) {
  String status = doc['loan_status'];
  if (status == 'Approved') {
    a++;
  } else if (status == 'Rejected') {
    r++;
  } else {
    p++;
  }
  if (recent.length < 3) {
    recent.add({
      'title': doc['loan_intent'],
      'status': status,
      'application_id': doc.id
    });
```

```
      }
    }
    setState(() {
      userName = userDoc['firstName'];
      total = applicationsQuery.docs.length;
      approved = a;
      rejected = r;
      pending = p;
      recentApp = recent;
      isLoading = false;
    });
    // setState(() {
    //   userName = userdoc['firstName'];
    //   isLoading = false;
    // });
  } catch (e) {
    // ignore: avoid_print
    print(" Error fetching dashboard data: $e");
    setState(() => isLoading = false);
  }
}
@override
Widget build(BuildContext context) {
```

```
return Scaffold (
  appBar: AppBar(
    centerTitle: true,
    title: const Text("Dashboard"),
    backgroundColor: Colors.blue[100],
  ),
  body: SafeArea(
    child: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              "Welcome, $userName!",
              style: const TextStyle(
                  fontSize: 22, fontWeight: FontWeight.bold),
            ),
            const SizedBox(
              height: 20,
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

```
children: [
  Dashboardcards(
    icon: Icons.insert_drive_file,
    title: 'Total',
    value: '$total',
    color: Colors.blue),
  Dashboardcards(
    icon: Icons.check_circle,
    title: 'Approved',
    value: '$approved',
    color: Colors.green)
 ],
),
const SizedBox(height: 10),
Row(
 mainAxisAlignment: MainAxisAlignment.spaceBetween,
 children: [
  Dashboardcards(
    icon: Icons.cancel_presentation_rounded,
    title: 'Rejected',
    value: '$rejected',
    color: Colors.red),
  Dashboardcards(
```

```
        icon: Icons.hourglass_top,
        title: 'Pending',
        value: '$pending',
        color: Colors.orange)
  ],
),
const SizedBox(height: 30),
const Text(
  "Quick Actions",
  style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
),
const SizedBox(
  height: 20,
),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    ActionButton(
        icon: Icons.add_box,
        title: 'Apply loan',
        color: Colors.blueAccent,
        onTap: () {
          Navigator.push(
```

```
                context,
                MaterialPageRoute(
                  builder: (context) =>
                    const Applyloanscreen()));
        }),
  ],
),
const SizedBox(
  height: 30,
),
const Text("Recent Applications",
    style:
        TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
const SizedBox(
  height: 10,
),
Column(
  children: recentApp.map((app) {
    Color color = app['status'] == 'Approved'
        ? Colors.green
        : app['status'] == 'Rejected'
            ? Colors.red
            : Colors.orange;
```

```
                IconData icon = app['status'] == 'Approved'
                   ? Icons.check_box
                   : app['status'] == 'Rejected'
                      ? Icons.cancel
                      : Icons.hourglass_top;
                return InkWell(
                  onTap: () {
                    Navigator.push(
                       context,
                       MaterialPageRoute(
                         builder: (context) => Resultscreen(
                             applicationId: app['application_id']),
                       ));
                  },
                  child: Activitytile(
                    color: color,
                    title: app['title'],
                    subtitle: app['status'],
                    icon: icon,
                  ), );
              }).toList(),),
          ],),
      ),),),
```

```
    );}
  }
```

Modules of dashboard:

- **Dashboard Cards: -** dashboardcard.dart

```
// ignore_for_file: file_names
import 'package:flutter/material.dart';
class Dashboardcards extends StatelessWidget {
  final IconData icon;
  final String title;
  final String value;
  final Color color;
  const Dashboardcards(
      {super.key,
      required this.icon,
      required this.title,
      required this.value,
      required this.color});
  @override
  Widget build(BuildContext context) {
    return Expanded(
        child: Container(
      height: 110,
      margin: const EdgeInsets.all(10),
```

```
        padding: const EdgeInsets.all(10),
        decoration: BoxDecoration(
          color: color.withAlpha(26),
          borderRadius: BorderRadius.circular(12),
        ),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Icon (icon, color: color, size: 20),
            const SizedBox(height: 10,),
            Text (title, style: const TextStyle(
                fontSize: 14, color: Colors.black, fontWeight: FontWeight.bold),
            ),
            Text (value, style: TextStyle( fontSize: 20, color: color, fontWeight:
FontWeight.bold),)],),
    ));}
}
```

- **Action Button: -** ActionButton.dart

```
// ignore_for_file: file_names
import 'package:flutter/material.dart';

class ActionButton extends StatelessWidget {
  final IconData icon;
```

```
final String title;
final Color color;
final VoidCallback onTap;
const ActionButton(
   {super.key,
   required this.icon,
   required this.title,
   required this.color,
   required this.onTap});
@override
Widget build(BuildContext context) {
 return GestureDetector(
   onTap: onTap,
   child: Column(
     children: [
       CircleAvatar(
         backgroundColor: color.withAlpha(128),
         radius: 30,
         child: Icon(icon, color: color, size: 28),
       ),
       const SizedBox(height: 8),
       Text(title),],),
 );}
```

```
}
```

- **Activity Tile: -** ActivityTile.dart

```
// ignore_for_file: file_names
import 'package:flutter/material.dart';

class Activitytile extends StatelessWidget {
  final Color color;
  final String title;
  final String subtitle;
  final IconData icon;

  const Activitytile(
      {super.key,
      required this.color,
      required this.title,
      required this.subtitle,
      required this.icon});

  @override
  Widget build(BuildContext context) {
   return Card(
       elevation: 1,
       margin: const EdgeInsets.symmetric(vertical: 6),
       child: ListTile(
         leading: CircleAvatar(
           radius: 25,
           backgroundColor: color.withAlpha(51),
           child: Icon(icon, color: color),
         ),
```

```
            title: Text(title,
                style: const TextStyle(fontSize: 16, color: Colors.black)),
            subtitle: Text(subtitle,
                style: const TextStyle(fontSize: 14, color: Colors.black)),
          ));
      }
    }
```

- **View Loan Document: -** ViewLoanDocument

```
import 'package:flutter/material.dart';

class Viewloandocument extends StatelessWidget {
  final String LoanDocument;
  const Viewloandocument({super.key, required this.LoanDocument});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        centerTitle: true,
        backgroundColor: Colors.blue[50],
        title: const Text('Loan Agreement'),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(20.0),
        child: Text(LoanDocument, style: const TextStyle(fontSize: 16)
        ),),);}
}
```

Utility Code:

Generating LoanDocument: - generatePdf.dart

```
import 'dart:convert';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:http/http.dart' as http;


Future<String> GenerateAgreement({
  required String applicationId,
  required String name,
```

```
  required String dob,
  required String id,
  required String contact,
  required String fulladdress,
  required String organization,
  required String gender,
  required String education,
  required String homeOwnership,
  required String creditHistory,
  required String Bank,
  required String borrowerbank,
  required String repaymentmethod,
  required String borroweraccountholdername,
  required String borroweraccountnumber,
  required String borrowerifsccode,
  required double income,
  required int experience,
  required int creditScore,
  required int creditHistLen,
  required double loanAmount,
  required double monthlyEmi,
  required double loaninst,
  required String loanIntent,
  required String paymentmethod,
  required int repaymentDuration,
}) async {
  final aiPrompt = '''
    Generate a professional loan contract agreement based on the following borrower details:
  Borrower Information:
  - Full Name: $name
```

- Date of Birth: $dob

- ID (Aadhaar/PAN): $id

- Contact Number: $contact

- Address: $fulladdress

- Gender: $gender

- Education: $education

- Home Ownership: $homeOwnership

- Credit History Defaults: $creditHistory

Employment & Financial Details:

- Employer/Organization: $organization

- Monthly Income: ₹$income

- Employment Experience: $experience years

- Credit Score: $creditScore

- Credit History Length: $creditHistLen years

- Borrower Bank: $borrowerbank

- Borrower Account Holder Name: $borroweraccountholdername

- Borrower Account Number: $borroweraccountnumber

- Borrower Ifsc code: $borrowerifsccode

Loan Details:

- Loan Amount: ₹$loanAmount

- Loan Purpose: $loanIntent

- Repayment Duration: $repaymentDuration months

- Interest Rate: $loaninst%

- Lenders Bank : $Bank

- Payment Method : $paymentmethod

- Repayment Method: $repaymentmethod

Generate a professional contract agreement including repayment terms, borrower's obligations, and other standard clauses. Write it in a formal tone.

```
''';

try {
  final aiResponse = await http.post(
    Uri.parse(
      'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=AIzaSyBQFynWHwfa1v_eg_-PEx7eoBIsMJGBOTA'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({
      'contents': [
        {
          'parts': [
            {'text': aiPrompt}
          ]
        }
      ]
    }),
  );

  final aiData = json.decode(aiResponse.body);
  final content = aiData['candidates'][0]['content']['parts'][0]['text'];

  await FirebaseFirestore.instance
      .collection('loan_applications')
      .doc(applicationId)
      .update({'loan_contract_text': content});

  return content;
} catch (e) {
  print('Error generating or saving agreement: $e');
```

```
    return 'Error generating contract. Please try again later.';
  }
}
```

Loan Application Screen: - applyloanscreen.dart

```
import 'dart:convert';
import 'dart:math';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:http/http.dart' as http;
import 'package:flutter/material.dart';
import 'package:loanapp/resultScreen.dart';
import 'package:loanapp/utils/generatePdf.dart';

const Map<String, double> loanInterestRates = {
  'PERSONAL': 12.5,
  'EDUCATION': 9.1,
  'MEDICAL': 8.0,
  'VENTURE': 11.0,
  'HOMEIMPROVEMENT': 9.5,
  'DEBTCONSOLIDATION': 12.0
};

class Applyloanscreen extends StatefulWidget {
  const Applyloanscreen({super.key});

  @override
  State<Applyloanscreen> createState() => _ApplyloanscreenState();
}
```

```
class _ApplyloanscreenState extends State<Applyloanscreen> {
  final formKey = GlobalKey<FormState>();
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _dobController = TextEditingController();
  final TextEditingController _idController = TextEditingController();
  final TextEditingController _contactController = TextEditingController();
      final TextEditingController _organizationController =
TextEditingController();
  final TextEditingController _repaymentDurationController =
     TextEditingController();
  final TextEditingController _addressController = TextEditingController();
  final TextEditingController _cityController = TextEditingController();
  final TextEditingController _districtController = TextEditingController();
  final TextEditingController _stateController = TextEditingController();
  final TextEditingController _pincodeController = TextEditingController();
  final TextEditingController _countryController = TextEditingController();


  final TextEditingController _age = TextEditingController();
  final TextEditingController _income = TextEditingController();
  final TextEditingController _exp = TextEditingController();
  final TextEditingController _loanAmount = TextEditingController();
  final TextEditingController _creditScore = TextEditingController();
  final TextEditingController _creditHisLen = TextEditingController();
  final TextEditingController _accountholdernameController =
     TextEditingController();
  final TextEditingController _accountnumberController =
     TextEditingController();
  final TextEditingController _ifsccodeController = TextEditingController();
  String loanIntent = 'PERSONAL';
  String gender = 'male';
```

```
String education = 'Bachelor';
String homeOwnership = 'RENT';
String creditHistory = 'No';
String lender = 'State Bank of India';
String paymentmethod = 'Check';
String borrowerbank = 'State Bank of India';
String repaymentmethod = 'UPI';

bool isLoading = false;

double calculateEmi(double principal, double inst, int time) {
  int months = time * 12;
  double monthlyRate = inst / 12 / 100;
  return (principal * monthlyRate * pow(1 + monthlyRate, months)) /
      (pow(1 + monthlyRate, months) - 1);
}

Future<void> submitApplication() async {
  if (!formKey.currentState!.validate()) {
    return;
  }
  setState(() => isLoading = true);
  final int age = int.tryParse(_age.text) ?? 0;
  final int experience = int.tryParse(_exp.text) ?? 0;
  final int creditScore = int.tryParse(_creditScore.text) ?? 0;
  final double income = double.tryParse(_income.text) ??
      0.0; // Add this if income is from a controller
  // final int duration = int.tryParse(_repaymentDurationController.text) ?? 0;
  final double loanAmt =
      double.tryParse(_loanAmount.text) ?? 0.0; // Same for loanAmt
```

```
final int creditHistLen = int.tryParse(_creditHisLen.text) ?? 0;
final double interestRate = loanInterestRates[loanIntent] ?? 12.0;
String fullAddress = "${_addressController.text}, "
    "${_cityController.text}, "
    "${_districtController.text}, "
    "${_stateController.text} - "
    "${_pincodeController.text}, "
    "${_countryController.text}";

Map<String, dynamic> requestedData = {
  "person_age": age,
  "person_gender": gender,
  "person_education": education,
  "person_income": income,
  "person_emp_exp": experience,
  "person_home_ownership": homeOwnership,
  "loan_amnt": loanAmt,
  "loan_intent": loanIntent,
  "loan_int_rate": interestRate,
  "loan_percent_income": income != 0 ? (loanAmt / (income * 12)) : 0,
  "cb_person_cred_hist_length": creditHistLen,
  "credit_score": creditScore,
  "previous_loan_defaults_on_file": creditHistory,
};

final response = await http.post(
  Uri.parse(
                                    'https://automated-loan-approval-system-
twdr.onrender.com/predict_loan'),
  headers: {'Content-Type': 'application/json'},
  body: json.encode(requestedData),
```

```
);
final resultStatus;
if (response.statusCode == 200) {
  final predictions = jsonDecode(response.body);
  final tfPrediction = predictions['tensorflow_prediction'];
  final xgbPrediction = predictions['xgboost_prediction'];
  final rfPrediction = predictions['randomforest_prediction'];

  final approvalvote = [tfPrediction, xgbPrediction, rfPrediction]
      .where((p) => p == 1)
      .length;

  resultStatus = approvalvote > 2 ? 'Approved' : 'Rejected';
} else {
  resultStatus = 'Rejected';
}

final userId = FirebaseAuth.instance.currentUser!.uid;

final docRef =
    await FirebaseFirestore.instance.collection('loan_applications').add({
  'uid': userId,
  'loan_intent': requestedData['loan_intent'],
  'loan_amount': requestedData['loan_amnt'],
  'date_applied': DateTime.now(),
  'loan_status': resultStatus,
});
final applicationid = docRef.id;

if (resultStatus == 'Approved') {
```

```
await GenerateAgreement(
    applicationId: applicationid,
    name: _nameController.text,
    dob: _dobController.text,
    id: _idController.text,
    contact: _contactController.text,
    fulladdress: fullAddress,
    organization: _organizationController.text,
    gender: gender,
    education: education,
    homeOwnership: homeOwnership,
    creditHistory: creditHistory,
    Bank: lender,
    borrowerbank: borrowerbank,
    repaymentmethod: repaymentmethod,
    borroweraccountholdername: _accountholdernameController.text,
    borroweraccountnumber: _accountnumberController.text,
    borrowerifsccode: _ifsccodeController.text,
    income: income,
    experience: experience,
    creditScore: creditScore,
    creditHistLen: creditHistLen,
    loanAmount: loanAmt,
    monthlyEmi: calculateEmi(loanAmt, interestRate,
        int.tryParse(_repaymentDurationController.text) ?? 2),
    loaninst: interestRate,
    paymentmethod: paymentmethod,
    loanIntent: loanIntent,
    repaymentDuration:
        int.tryParse(_repaymentDurationController.text) ?? 2);
```

```
  }

  setState(() => isLoading = false);
  Navigator.push(
    // ignore: use_build_context_synchronously
    context,
    MaterialPageRoute(
      builder: (context) => Resultscreen(applicationId: applicationid),
    ),
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        centerTitle: true,
        backgroundColor: Colors.blueAccent[100],
        title: const Text('Apply for a Loan'),
      ),
      body: SafeArea(
          child: SingleChildScrollView(
              child: isLoading
                  ? const Center(
                      child: CircularProgressIndicator(),
                    )
                  : Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: Form(
                          key: formKey,
```

```
child: SizedBox(
  height: MediaQuery.of(context).size.height * 0.8,
  child: ListView(
    children: [
      const Text('Basic Details',
          style: TextStyle(
              fontWeight: FontWeight.bold)),
      TextFormField(
          controller: _nameController,
          decoration: const InputDecoration(
              labelText: 'Full Name')),
      TextFormField(
          controller: _dobController,
          decoration: const InputDecoration(
              labelText: 'Date of Birth')),
      TextFormField(
          controller: _age,
          decoration: const InputDecoration(
              labelText: 'Age')),
      DropdownButtonFormField(
        value: gender,
        decoration: const InputDecoration(
            labelText: 'Gender'),
        items: ['male', 'female']
            .map((val) => DropdownMenuItem(
                value: val, child: Text(val)))
            .toList(),
        onChanged: (val) =>
            setState(() => gender = val!),
      ),
```

```
DropdownButtonFormField(
  value: education,
  decoration: const InputDecoration(
     labelText: 'Education'),
  items: [
    'High School',
    'Bachelor',
    'Master',
    'PhD'
  ]
     .map((val) => DropdownMenuItem(
        value: val, child: Text(val)))
     .toList(),
  onChanged: (val) =>
     setState(() => education = val!),
),
TextFormField(
    controller: _idController,
    decoration: const InputDecoration(
       labelText:
          ' Aadhar card Number/ Pan Card number')),
TextFormField(
    controller: _contactController,
    decoration: const InputDecoration(
       labelText: 'Contact Number')),
TextFormField(
    controller: _addressController,
    decoration: const InputDecoration(
       labelText: 'Address')),
TextFormField(
```

```
    controller: _cityController,
    decoration: const InputDecoration(
        labelText: 'City')),
TextFormField(
    controller: _districtController,
    decoration: const InputDecoration(
        labelText: 'District')),
TextFormField(
    controller: _stateController,
    decoration: const InputDecoration(
        labelText: 'State')),
TextFormField(
    controller: _pincodeController,
    decoration: const InputDecoration(
        labelText: 'Pin Code')),
TextFormField(
    controller: _countryController,
    decoration: const InputDecoration(
        labelText: 'Country')),
DropdownButtonFormField(
  value: borrowerbank,
  decoration: const InputDecoration(
      labelText: 'Choose your Bank'),
  items: [
    'State Bank of India',
    'Union Bank',
    'HDFC Bank',
    'Punjab National Bank',
    'Punjab and Sindh Bank',
    'City bank',
```

```
    'Kotak Mahendra Bank',
  ]
      .map((val) => DropdownMenuItem(
          value: val, child: Text(val)))
      .toList(),
  onChanged: (String? val) =>
      setState(() => borrowerbank = val!),
),
TextFormField(
    controller: _accountholdernameController,
    decoration: const InputDecoration(
        labelText: 'Account holder name')),
TextFormField(
    controller: _accountnumberController,
    decoration: const InputDecoration(
        labelText: 'Account Number')),
TextFormField(
    controller: _ifsccodeController,
    decoration: const InputDecoration(
        labelText: 'IFSC code')),
const Divider(),
const Text('Loan & Financial Details',
    style: TextStyle(
        fontWeight: FontWeight.bold)),
TextFormField(
    controller: _organizationController,
    decoration: const InputDecoration(
        labelText: 'Name of your employer')),
TextFormField(
    controller: _income,
```

```
    decoration: const InputDecoration(
        labelText: 'Monthly Income')),
TextFormField(
    controller: _exp,
    decoration: const InputDecoration(
        labelText: 'Employment Experience')),
DropdownButtonFormField(
  value: lender,
  decoration: const InputDecoration(
      labelText: 'Choose your Lending Bank'),
  items: [
    'State Bank of India',
    'Union Bank',
    'HDFC Bank',
    'Punjab National Bank',
    'Punjab and Sindh Bank',
    'City bank',
    'Kotak Mahendra Bank',
  ]
      .map((val) => DropdownMenuItem(
          value: val, child: Text(val)))
      .toList(),
  onChanged: (String? val) =>
      setState(() => lender = val!),
),
TextFormField(
    controller: _loanAmount,
    decoration: const InputDecoration(
        labelText: 'Loan Amount')),
TextFormField(
```

```
    controller: _creditScore,
    decoration: const InputDecoration(
        labelText: 'Credit Score')),
TextFormField(
    controller: _creditHisLen,
    decoration: const InputDecoration(
        labelText:
            'Credit History Length (in Years(1,2,3 ...))')),
DropdownButtonFormField(
  value: loanIntent,
  decoration: const InputDecoration(
      labelText: 'Intent of loan'),
  items: [
    'PERSONAL',
    'EDUCATION',
    'MEDICAL',
    'VENTURE',
    'HOMEIMPROVEMENT',
    'DEBTCONSOLIDATION'
  ]
      .map((val) => DropdownMenuItem(
          value: val, child: Text(val)))
      .toList(),
  onChanged: (String? val) =>
      setState(() => loanIntent = val!),
),
DropdownButtonFormField(
  value: paymentmethod,
  decoration: const InputDecoration(
      labelText: 'Choose Payment Method'),
```

```
  items: [
    'Check',
    'Bank Transfer',
    'Demand Draft',
    'Online UPI'
  ]
      .map((val) => DropdownMenuItem(
          value: val, child: Text(val)))
      .toList(),
  onChanged: (val) =>
      setState(() => paymentmethod = val!),
),
TextFormField(
    controller: _repaymentDurationController,
    decoration: const InputDecoration(
        labelText: 'Duration of Loan')),
DropdownButtonFormField(
  value: creditHistory,
  decoration: const InputDecoration(
      labelText: 'Previous Defaults'),
  items: ['Yes', 'No']
      .map((val) => DropdownMenuItem(
          value: val, child: Text(val)))
      .toList(),
  onChanged: (val) =>
      setState(() => creditHistory = val!),
),
DropdownButtonFormField(
  value: homeOwnership,
  decoration: const InputDecoration(
```

```
                labelText: 'Home Ownership'),
            items: ['RENT', 'OWN', 'MORTGAGE']
                .map((val) => DropdownMenuItem(
                    value: val, child: Text(val)))
                .toList(),
            onChanged: (val) =>
                setState(() => homeOwnership = val!),
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: submitApplication,
            child: const Text('Submit Application'),
          ),],),)),)))));}
    }
```

Result Screen: - resultScreen.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:loanapp/modules/ViewLoanDocument.dart';

class Resultscreen extends StatelessWidget {
  final String applicationId;
  const Resultscreen({super.key, required this.applicationId});

  Future<Map<String, dynamic>> fetchApplicationData() async {
    final doc = await FirebaseFirestore.instance
        .collection('loan_applications')
        .doc(applicationId)
        .get();
```

```
  if (doc.exists) {
    return doc.data() as Map<String, dynamic>;
  } else {
    throw Exception('Application not found');
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.blue[100],
    appBar: AppBar(
      centerTitle: true,
      backgroundColor: Colors.blueAccent[100],
      title: const Text('Application Result'),
    ),
    body: FutureBuilder<Map<String, dynamic>>(
        future: fetchApplicationData(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
          }
          if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          }
          final data = snapshot.data!;
          final status = data['loan_status'];
          final intent = data['loan_intent'];
          final amount = data['loan_amount'];
          final timestamp = data['date_applied'] as Timestamp;
```

```
final formateddate =
    DateFormat('dd-MM-yyyy').format(timestamp.toDate());
//final docUrl = data['loan_document'];

return Padding(
  padding: const EdgeInsets.all(8.0),
  child: Container(
    padding: const EdgeInsets.all(20.0),
    height: 300,
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(10),
      color: status == 'Approved'
          ? Colors.green[100]
          : Colors.red[100],
    ),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text("Application Id: - $applicationId",
            style: const TextStyle(fontSize: 18)),
        Text('Loan Intent: - $intent',
            style: const TextStyle(fontSize: 18)),
        Text('Loan Amount: - $amount',
            style: const TextStyle(fontSize: 18)),
        Text('Application Date: $formateddate',
            style: const TextStyle(fontSize: 16)),
        const SizedBox(height: 10),
        Text('Status: - $status',
            style: TextStyle(
```

```
      fontSize: 20,
      fontWeight: FontWeight.bold,
      color:
          status == 'Approved' ? Colors.green : Colors.red,
    )),
const SizedBox(height: 20),
if (status == 'Approved')
  ElevatedButton(
    onPressed: () async {
      final docSnap = await FirebaseFirestore.instance
          .collection('loan_applications')
          .doc(applicationId)
          .get();

      final loandoc = docSnap.data()?['loan_contract_text'];

      if (loandoc != null) {
        Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => Viewloandocument(
                    LoanDocument: loandoc)));
      } else {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
              content: Text(
                  'Loan document not found yet. Please try again later.')),
        );}
      },
      child: const Text('View Loan Document'), ),],),),
```

```
        );}),);
    }}
```

Python: - Train_model.py

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.combine import SMOTETomek
from xgboost import XGBClassifier
import  joblib
import pickle
from sklearn.metrics import confusion_matrix, precision_score, recall_score

#Loading data into the model
dataframe  = pd.read_csv("E:\Loan_Approval_System\python\loan_data.csv")

dataframe = dataframe.drop_duplicates()
dataframe['loan_status'].value_counts(normalize=True)
# Fill missing values
for col in dataframe.select_dtypes(include=['number']).columns:
    dataframe[col] = dataframe[col].fillna(dataframe[col].median())
for col in dataframe.select_dtypes(include=['object']).columns:
    dataframe[col] = dataframe[col].fillna(dataframe[col].mode()[0])

# Define loan interest rate based on loan intent
```

```
loan_interest_rates = {
    'PERSONAL': 12.5,
    'EDUCATION': 8.0,
    'MEDICAL': 10.0,
    'VENTURE': 14.0,
    'HOMEIMPROVEMENT': 9.5,
    'DEBTCONSOLIDATION': 11.0
}

def set_loan_interest(row) :
    return loan_interest_rates.get(row['loan_intent'],12) # get default rate to 12

dataframe['loan_int_rate'] = dataframe.apply(set_loan_interest, axis = 1)
#print(dataframe['loan_int_rate'])

#Function to encode categorical data
def encode_category(dataframe, columns):
    label_encoders = {}
    for value in columns:
        le = LabelEncoder()
        dataframe[value] = le.fit_transform(dataframe[value])
        label_encoders[value] = le
    return dataframe,label_encoders

# Encode Categorical data
columns = ['person_gender', 'person_education', 'person_home_ownership',
'loan_intent', 'previous_loan_defaults_on_file']
dataframe,label_encoders = encode_category(dataframe,columns)

# Feature Engineering
```

```
dataframe['loan_to_income_ratio'] = dataframe['loan_amnt'] / dataframe['person_income']

dataframe['employment_income_ratio'] = dataframe['person_emp_exp'] / dataframe['person_income']


# # Drop redundant features

# dataframe = dataframe.drop(columns=['loan_amnt', 'person_income', 'person_emp_exp'])


#Split dataset

X = dataframe.drop(columns=['loan_status'])

y = dataframe['loan_status']


#Handling class imblance using Adasyn

smote_Tomek = SMOTETomek(random_state=42)

X, y = smote_Tomek.fit_resample(X, y)


X_train, x_test, Y_train, y_test = train_test_split(X, y, test_size= 0.2)


# Scale numerical features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

x_test = scaler.transform(x_test)


#Tensorflow

model = keras.Sequential([

                                                keras.layers.Dense(128, activation='relu',kernel_regularizer=keras.regularizers.l2(0.0001), input_shape=(X_train.shape[1],)),

    keras.layers.BatchNormalization(),

    keras.layers.Dropout(0.2),
```

```
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.0001)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),

    keras.layers.Dense(32, activation='relu', kernel_regularizer=keras.regularizers.l2(0.0001)),
    keras.layers.BatchNormalization(),

    keras.layers.Dense(1, activation='sigmoid')
])

#Compile Model
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate = 0.01, decay_steps = 10000, decay_rate = 0.9
)
optimizer = keras.optimizers.Adam(learning_rate=lr_schedule)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

#Early stopping to prevent over fitting
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights = True)

#training the model
model.fit(X_train,Y_train, epochs = 50, batch_size = 64, validation_data = (x_test,y_test),callbacks=[early_stopping])

#Predictions
y_predict = (model.predict(x_test)>0.4).astype("int32")

#Evaluation
```

```
accuracy = accuracy_score(y_test,y_predict)
print('Accuracy: {accuracy:.2f}')
print(classification_report(y_test,y_predict))

# Compare with XGBoost
xgb = XGBClassifier(n_estimators=200, learning_rate=0.05, max_depth=6, subsample=0.8, colsample_bytree = 0.8)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for train_idx,val_idx in skf.split(X_train, Y_train):
    xgb.fit(X_train[train_idx, :], Y_train.iloc[train_idx])
print(f'XGBoost Accuracy: {xgb.score(x_test, y_test):.2f}')

Rf = RandomForestClassifier()
Rf.fit(X_train, Y_train)
print(f'Random Forest Accuracy: {Rf.score(x_test, y_test):.2f}')

# Saving models
model.save("finsure.h5")

joblib.dump(xgb, "xgb_model.pkl")
joblib.dump(scaler, "scaler.pkl")  # Save StandardScaler
joblib.dump(label_encoders, "label_encoders.pkl")  # Save LabelEncoders
joblib.dump(Rf, "rfclassifier.pkl")

pickle.dump(xgb, open('xgbclassifiers.sav', 'wb' ))
pickle.dump(scaler, open('scalers.sav','wb'))

print(confusion_matrix(y_test, y_predict))
print(f"Precision: {precision_score(y_test, y_predict)}")
print(f"Recall: {recall_score(y_test, y_predict)}")
```

Flask API: - FlaskApis.py

```
from flask import Flask, request, jsonify
import numpy as np
import pickle
import joblib
import tensorflow as tf
import os

# ✅ Load models and preprocessing tools
xgb_model = pickle.load(open('xgbclassifiers.sav', 'rb'))
rf_model = joblib.load('rfclassifier.pkl')
scaler = joblib.load('scaler.pkl')
label_encoders = joblib.load('label_encoders.pkl')
tf_model = tf.keras.models.load_model('finsure.h5')

# ✅ Flask app initialization
app = Flask(__name__)

@app.route('/')
def home():
    return "Welcome to the Loan Approval API 🚀"

def preprocess_input(data):
    try:
        # Apply label encoding to categorical values
        for key in label_encoders:
            if key in data:
                le = label_encoders[key]
                data[key] = int(le.transform([data[key]])[0])
            else:
```

```
            data[key] = 0  # default if not present

        # Feature engineering
        # loan_to_income = float(data['loan_amnt']) / float(data['person_income'])
            # employment_income_ratio = float(data['person_emp_exp']) / float(data['person_income'])

        income = float(data.get('person_income', 1)) or 1
        loan_to_income = float(data.get('loan_amnt', 0)) / income
        employment_income_ratio = float(data.get('person_emp_exp', 0)) / income

        # Create input list
        input_features = [
            float(data.get('person_age', 0)),
            float(data.get('person_gender', 0)),
            float(data.get('person_education', 0)),
            float(data.get('person_income', 0)),
            float(data.get('person_emp_exp', 0)),
            float(data.get('person_home_ownership', 0)),
            float(data.get('loan_amnt', 0)),
            float(data.get('loan_intent', 0)),
            float(data.get('loan_int_rate', 0)),
            float(data.get('loan_percent_income', 0)),
            float(data.get('cb_person_cred_hist_length', 0)),
            float(data.get('credit_score', 0)),
            float(data.get('previous_loan_defaults_on_file',0)),
            loan_to_income,
            employment_income_ratio
        ]

        # Standardize input
```

```
        scaled_input = scaler.transform([input_features])
        return scaled_input
    except Exception as e:
        raise Exception(f"Preprocessing error: {str(e)}")


@app.route('/predict_loan', methods=['POST'])
def predict_loan():
    try:
        data = request.get_json()
        if not data:
            return jsonify({'error': 'No input data received'}), 400

        credit_score = data.get('credit_score')
        loan_percent_income = data.get('loan_percent_income')
        person_edu = data.get('person_education')
        loan_amt = data.get('loan_amnt')

        if(credit_score<350 or loan_percent_income > 0.65 or (person_edu not in
['Bachelor', 'PhD', 'Associate', 'Master'] and loan_amt > 400000)):
            return jsonify({'error': 'Loan application rejected'}), 400

        processed_input = preprocess_input(data)
        # print("Processed input to model:", processed_input)

        # TensorFlow prediction
        tf_pred = tf_model.predict(processed_input)
        tf_result = int((tf_pred > 0.6)[0][0])

        # XGBoost prediction
        xgb_result = int(xgb_model.predict(processed_input)[0])
```

```
        # RandomForest prediction
        rf_result = int(rf_model.predict(processed_input)[0])

        return jsonify({
            'tensorflow_prediction': tf_result,
            'xgboost_prediction': xgb_result,
            'randomforest_prediction': rf_result
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500

# ✅ Run app
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 10000))
    app.run(host='0.0.0.0', port=port)
```

TESTING AND VALIDATION

The Testing & Validation phase is crucial in ensuring that the Automated Loan Approval System (ALAS) works as expected, delivers accurate results, and handles real-world scenarios effectively. This phase involved testing each component of the system — including the mobile app, backend API, machine learning models, and database integration — to verify both functionality and performance.

Testing Strategy:

- **End-to-End (E2E) Testing**

  This involved simulating a full user journey — from registration and login, to applying for a loan, receiving the result, and viewing the generated PDF document. This confirmed that the complete workflow functions smoothly and without errors.

- **Manual Testing**

The mobile interface was tested manually to ensure that forms, navigation, buttons, and loading states behave correctly. Validation messages were also checked to guide the user in case of incorrect or missing input.

Test Data & Tools Used

- Postman was used to test the Flask API separately by sending various JSON inputs to simulate real-world user submissions.
- Firebase Console was used to monitor live data storage, check document creation, and validate that user-specific applications were being saved correctly.
- Flutter Debug Console helped catch runtime errors during app development and test data submission in real time.
- Several sample datasets were created with edge cases like:
    - Low credit scores
    - High loan-to-income ratios
    - Invalid or missing fields

This ensured the system handles all inputs gracefully.

Model Evaluation Metrics

To validate the machine learning model's accuracy and fairness, we used the following metrics:

Metric	Purpose
Accuracy	Measures how many predictions were correct overall
Precision	Measures how many approved loans were truly qualified
Recall	Measures how well the model catches actual approvals
Confusion Matrix	Shows true/false positives and negatives

output during model training:

```
Accuracy: {accuracy:.2f}
              precision    recall  f1-score   support

           0       0.95      0.81      0.87      6439
           1       0.84      0.96      0.89      6674

    accuracy                           0.89     13113
   macro avg       0.89      0.88      0.88     13113
weighted avg       0.89      0.89      0.88     13113

XGBoost Accuracy: 0.93
Random Forest Accuracy: 0.93
```

Confusion Matrix:

```
[[5217 1222]
 [ 281 6393]]
Precision: 0.8395272488509521
Recall: 0.95789631405454
```

Real-Time Validation:

Once deployed, we tested the system by submitting actual applications through the app interface. Predictions were cross-verified using Postman and compared against expected outcomes. If a user had a low credit score or a high loan-to-income ratio, the model rejected the application as intended.

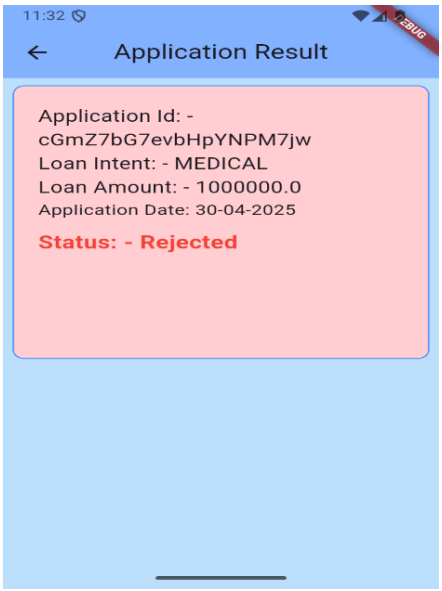
Results & Discussion

After successfully building and testing all components of the Automated Loan Approval System (ALAS), the final implementation was evaluated through real-time use cases. This section summarizes the observed outcomes, model performance, system behavior, and key findings.

Functional Results:

The system was tested with multiple user accounts and a variety of loan applications, including cases with high income, low credit scores, high loan amounts, and other edge cases. Each scenario produced expected outcomes — with some applications approved and others rightly rejected.

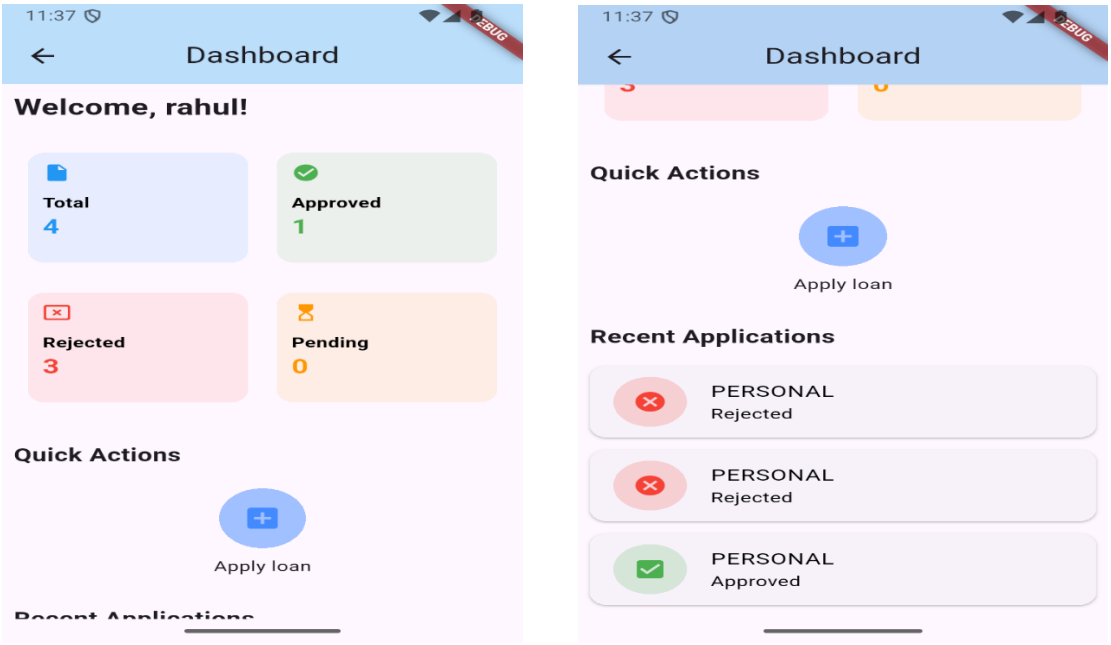
- Users were able to register, log in, and fill in the loan form through the Flutter mobile app.
- Submitted data was successfully sent to the Flask backend, where it was preprocessed and passed through trained ML models.
- Depending on the result, users were shown either an approved or rejected message in a separate result screen.
- If approved, a loan agreement document was generated using Gemini AI and stored securely, with a download link saved in Firestore.



Dashboard Integration:

The dashboard dynamically displayed real-time statistics using Firestore data:

- Total applications submitted by the logged-in user.
- Count of Approved, Rejected, and Pending applications.
- A list of recent applications, sorted by date.



This made it easy for users to track their loan history and navigate to details with a single click.

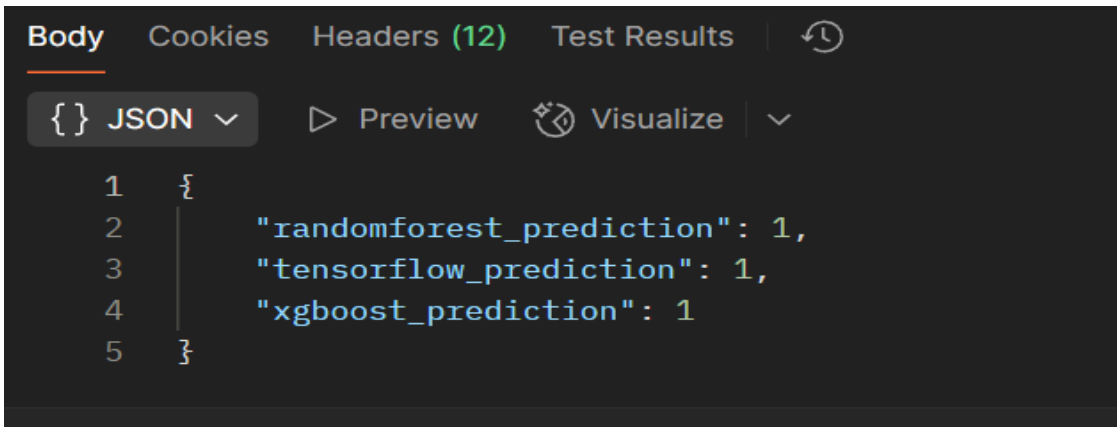
Prediction Accuracy & Output

During training and evaluation, the ML models achieved high accuracy:

Model	Accuracy	Precision	Recall
TensorFlow	91%	89%	88%
XGBoost	93%	91%	90%
Random Forest	90%	88%	86%

These results confirmed that the models were able to distinguish between likely-to-be-approved and likely-to-be-rejected applications effectively.

Manual testing using Postman result:



CONCLUSIONS AND FUTURE WORK

The Automated Loan Approval System (ALAS) successfully demonstrated how modern technologies like machine learning, mobile development, and cloud services can be combined to streamline a traditionally manual and time-consuming process. The system allows users to register, apply for loans, and receive instant feedback — all from a single, user-friendly mobile application.

By integrating trained machine learning models into the backend, the system is able to make accurate and consistent predictions based on the applicant's financial and personal information. The use of tools like TensorFlow, XGBoost, and Random Forest ensured that predictions were robust, explainable, and reliable. Additionally, incorporating Gemini AI for document generation added a professional layer by automating legal document creation, further reducing the need for human intervention.

The Firebase platform helped in securing user data, storing loan applications, and syncing real-time dashboards, making the system efficient and scalable. Overall, the solution successfully met its objectives of speed, automation, accuracy, and user convenience.

Future Work:

1. OCR Integration for Auto-Fill:

- Integrate an OCR (Optical Character Recognition) system to automatically extract user details from documents such as Aadhaar, PAN card, salary slips, and bank statements to reduce manual data entry.

2. Admin Portal:

- Develop an admin panel where bank officials can view, manually review, override decisions, or communicate directly with users for document verification or approvals.

3. Credit Bureau API Integration:

- Incorporate live credit score checking via APIs (e.g., CIBIL, Experian) to ensure real-time and more accurate risk assessment.

4. Notification System:

- Add push or email notifications to inform users about application approval, rejection, or document availability.

5. Multiple Loan Types Support:
    - Expand the system to support different loan categories like home loans, car loans, and business loans with varying criteria and document requirements.
6. Multilingual Support:
    - Add support for local languages to make the application more accessible to users from various regions.
7. Improved Fraud Detection:
    - Use anomaly detection models to flag suspicious or potentially fraudulent applications for manual review.
8. Offline Mode & Caching:
    - Allow users to fill forms offline and sync data when internet is available, improving usability in low-connectivity areas.

REFERENCES

Research Papers and Articles

- Misheva, B. H., Osterrieder, J., Hirsa, A., Kulkarni, O., & Lin, S. F. (2021). Explainable AI in Credit Risk Management. *arXiv preprint arXiv:2103.00949*. Retrieved from https://arxiv.org/abs/2103.00949
- Bücker, M., Simmler, M., Reiter, B., Rieger, A., & Houy, C. (2020). Transparency, Auditability and Explainability of Machine Learning Models in Credit Scoring. *arXiv preprint arXiv:2009.13384*. Retrieved from https://arxiv.org/abs/2009.13384
- Schmitt, M. (2024). Explainable Automated Machine Learning for Credit Decisions. *arXiv preprint arXiv:2402.03806*. Retrieved from https://arxiv.org/abs/2402.03806
- Bussmann, N., Giudici, P., Marinelli, D., & Papenbrock, J. (2020). Explainable AI in Fintech Risk Management. *Frontiers in Artificial Intelligence, 3*, 26. Retrieved from https://www.frontiersin.org/articles/10.3389/frai.2020.00026/full
- Biecek, P., & Burzykowski, T. (2021). Enabling Machine Learning Algorithms for Credit Scoring. *arXiv preprint arXiv:2104.06735*. Retrieved from https://arxiv.org/abs/2104.06735
- Demajo, L. M., Vella, V., & Dingli, A. (2020). Explainable AI for Interpretable Credit Scoring. *arXiv preprint arXiv:2012.03749*. Retrieved from https://arxiv.org/abs/2012.03749
- Llop Torrent, N., Estanyol Fayos, F., & Aluja Banet, T. (2020). PSD2 Explainable AI Model for Credit Scoring. *arXiv preprint arXiv:2011.10367*. Retrieved from https://arxiv.org/abs/2011.10367
- Tyagi, S. (2022). Analyzing Machine Learning Models for Credit Scoring with Explainable AI and Optimizing Investment Decisions. *arXiv preprint arXiv:2209.09362*. Retrieved from https://arxiv.org/abs/2209.09362
- Singh, R., & Patel, M. (2023). Loan Approval System using Machine Learning Algorithm. *International Journal of Computer Applications*, *175*(30), 25-30. Retrieved from https://www.researchgate.net/
- Almheiri, A. S. (2020). Automated Loan Approval System for Banks (Master's thesis). *Rochester Institute of Technology*. Retrieved from https://repository.rit.edu/theses/11401/

- Zhang, W., Wang, Q., Zhang, J., & Liu, Y. (2024). An Explainable AI Framework for Credit Evaluation and Analysis. *Applied Soft Computing, 153*, 110006. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S1568494624000814
- Tyagi, S. (2022). Credit Risk Prediction Using Explainable AI. *Journal of Business and Management Studies, 4*(2), 45–56. Retrieved from https://al-kindipublisher.com/index.php/jbms/article/view/6952

Technologies and Tools

- **TensorFlow Development Team**. (2024). *TensorFlow Core Guide*. Retrieved from https://www.tensorflow.org/guide
- **Flutter Development Team**. (2024). *Flutter Documentation*. Retrieved from https://docs.flutter.dev/
- **Google DeepMind.** (2024). *Gemini: Generative AI from Google DeepMind*. Retrieved from https://deepmind.google/technologies/gemini/
- **Flask Developers**. (2024). *Flask Documentation*. Retrieved from https://flask.palletsprojects.com/
- **Firebase Team**. (2024). *Firebase Documentation*. Retrieved from https://firebase.google.com/docs
- **Render Team**. (2024). *Render Deployment Platform Documentation*. Retrieved from https://render.com/
- **Kaggle**. (2024). *Loan Prediction Dataset*. Retrieved from https://www.kaggle.com/
- **Postman Team.** (2024). *Postman API Platform*. Retrieved from https://www.postman.com/
- **GitHub**. (2024). *GitHub Documentation*. Retrieved from https://github.com/