📄 Automated Personal Loan Document Processing

📘 Comprehensive Project Report

🔗 GitHub: https://github.com/Amanraj0149/loan-ocr-app

🔗 Live App: https://loan-ocr-app.onrender.com

🔗 Demo Recording: Video Presentation of Project

🔍 Problem Statement

Banks manually handle thousands of personal loan applications daily. This involves extracting key information such as Name, Address, Income, and Loan Amount from scanned documents like ID proofs and salary slips.

Challenges:
- ⏱️ Time-consuming
- ❌ Prone to human error
- 📈 Difficult to scale

🎯 Objective

To develop a full-stack OCR-based web application that automates document processing using:
- Image preprocessing
- Text recognition
- Field extraction
- Manual correction
- Validation
- Backend storage

🧩 Solution Features

- ✔️ Upload scanned loan documents
- ✔️ Image preprocessing with Sharp for better OCR
- ✔️ Text extraction using Tesseract.js

- ✔️ Regex-based field detection
- ✔️ Manual correction interface
- ✔️ Field validation
- ✔️ MongoDB integration for persistence

🛠️ Technology Stack

Technology	Usage
Node.js + Express	Backend server & routing
EJS	Dynamic templates for UI
Tesseract.js	OCR for extracting text
Sharp	Image preprocessing (grayscale etc)
Multer	File upload middleware
MongoDB Atlas	Cloud-based database
express-validator	Server-side validation

⚙️ Methodology

1. 🖼️ Image Preprocessing

Uploaded images are normalized and gray scaled using Sharp:

```
await sharp(originalPath).grayscale().normalize().toFile(processedPath);
```

✅ Helps reduce OCR noise and improves accuracy.

2. 🔍 OCR Text Extraction

OCR is performed using Tesseract.js to extract raw text:

```
const result = await Tesseract.recognize(processedPath, "eng");
const text = result.data.text;
```

✅ Converts scanned images into usable text.

3. 🧠 Field Extraction with Regex

Regex identifies fields like Name, Address, Income, Loan Amount:

```
if (/loan\s*amount\s*[:\-]/i.test(line)) {
  loanAmount = line.split(/[:\-]/)[1]?.trim() || loanAmount;
}
```

✅ Detects values even in noisy/unstructured text.

4. ✅ Field Validation

Validation ensures format integrity:

```
body("income").matches(/^\d[\d,]*$/).withMessage("Invalid income format");
```

✅ Prevents invalid entries from being submitted.

5. 🖼️ Manual Correction UI ✍️

If any field is "Not found", it is flagged in red and submission is blocked until corrected.

They can:
- Manually edit fields
- Upload a new document

EJS Example:

```
<input name="address"
  value="<%= extractedData.address %>"
  class="<%= extractedData.address === 'Not found' ? 'input-error' : '' %>"/>
```

✅ Ensures accuracy before database entry.

6. ☁️ Backend Image Optimization with Sharp

In the /upload route, Sharp preprocesses the uploaded image to enhance OCR accuracy and improve field detection.

```
// Inside the /upload route
const originalPath = req.file.path;
const processedPath = `processed/processed_${Date.now()}.png`;

// Image preprocessing
await sharp(originalPath)
  .grayscale()          // Convert to grayscale for clarity
  .normalize()          // Normalize brightness/contrast
  .toFile(processedPath);  // Save processed image for OCR
```

7. ☁️ Backend Integration

Validated data is stored in MongoDB Atlas:

```
const appData = new Application({ name, address, income, loanAmount });
await appData.save();
```

✅ Enables persistent, scalable storage.

✅ Final Feature Implementation

Feature	Status	Description
Manual correction UI ✍️	✅ Done	result.ejs allows manual review & edit of OCR-extracted fields
Image preprocessing 🧼	✅ Done	sharp improves OCR results by cleaning the image
Field-level validation ✅	✅ Done	express-validator ensures valid and clean data
Backend integration ☁️	✅ Done	MongoDB Atlas stores form data securely

🧪 Results

- System tested with varied image inputs (clear, noisy, angled)
- Successfully extracted key data
- Manual override improved form accuracy
- Error alerts prevented invalid submissions

❗ Challenges & Solutions

Challenge	Solution
Poor quality scans	Used Sharp to normalize and grayscale before OCR
OCR misreads or missing fields	Regex + manual correction UI
Invalid form submissions	Handled using express-validator
Deployment folder not existing	Created folders dynamically

🏁 Conclusion

This project successfully automates personal loan form processing using OCR, Sharp, and MongoDB. It reduces manual effort, supports validation and corrections, and offers a scalable backend for data persistence.

🧠 Why This Approach Was Selected

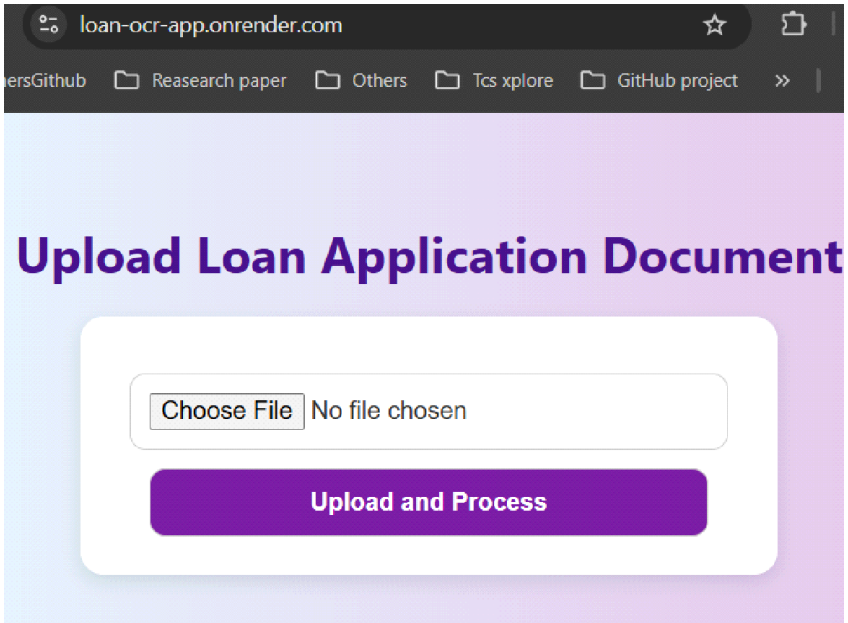
We chose this architecture and technology stack for its scalability, modularity, and open-source nature:

- Tesseract.js is a well-supported, browser-compatible OCR engine
- Sharp helps improve OCR quality by preprocessing poor scans
- express-validator enforces reliable data entry and avoids manual errors
- MongoDB Atlas is a secure cloud-native database ideal for storing application submissions

🧪 Testing & UI Results

Below are screenshots showing:

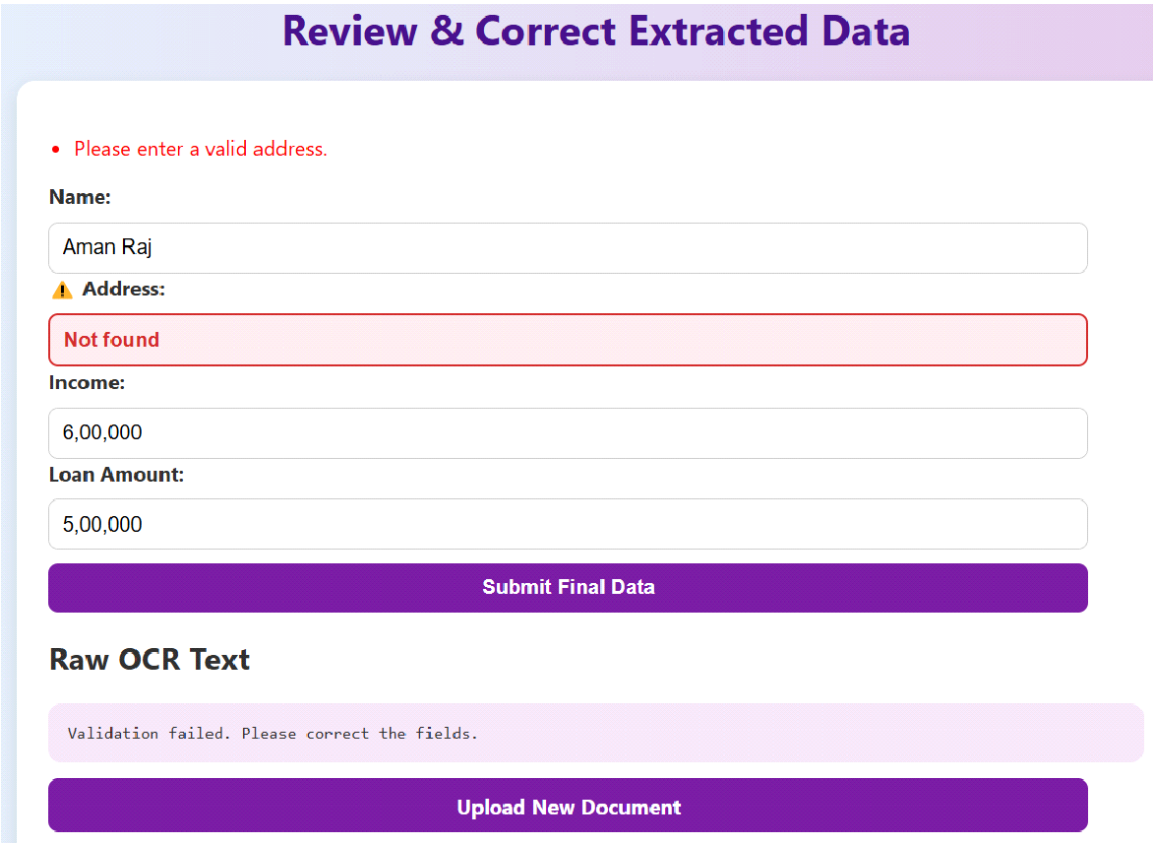
- OCR failure triggering manual correction and validation
- Successful field extraction and editable UI



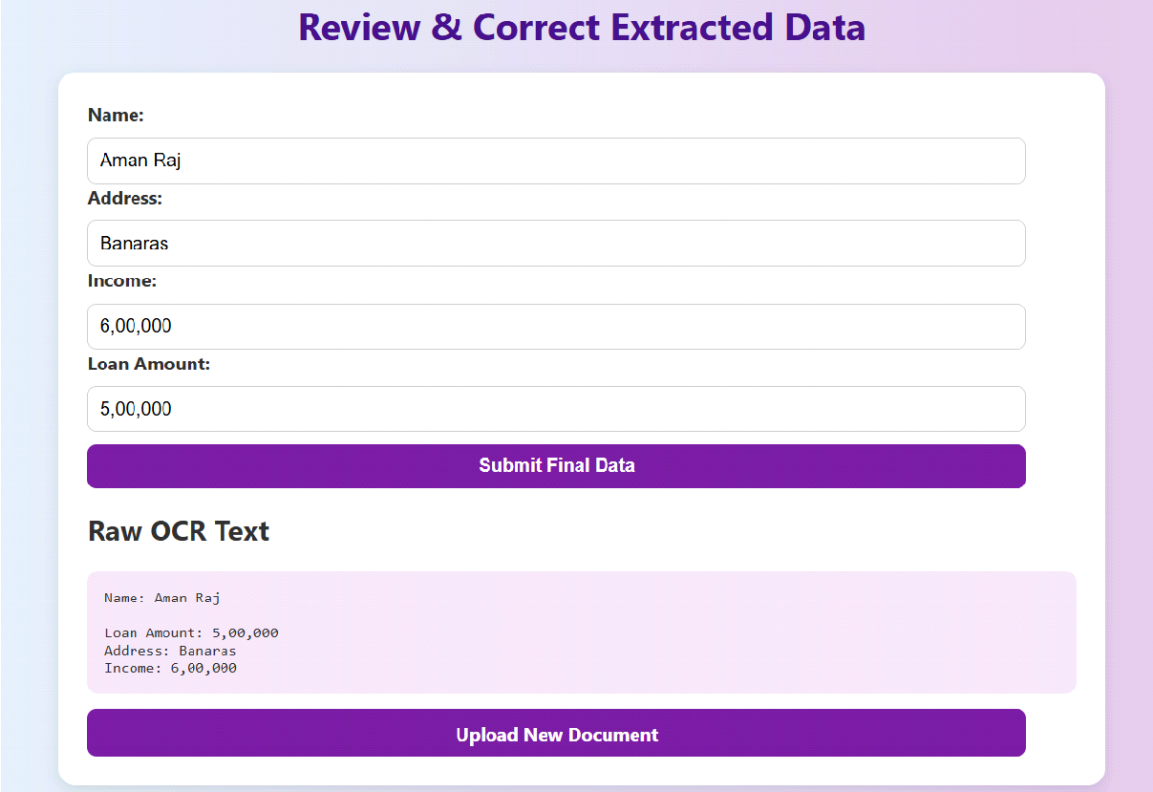
🔴 Example: Missing Address — user is alerted

```
Name: Aman Raj
Loan Amount: 5,00,000
Income: 6,00,000
```

✍️ Example: Editable Fields after OCR



✅ Final Corrected Data and Submission Screen



👨‍💻 Prepared By

Name: Aman Raj

CT/DT ID: CT20244436390

📧 Email: amanraj0149@gmail.com