Project Report

Submitted By

S. No.	Team Member	Roll No.
1	Aditya Kumar Yadav	24293916005
2	Pranjal Gupta A	24293916057
3	Gaurav Yadav	24293916123
4	Tejaswi Anand	24293916146
5	Bhakti	24293916046

(Semester III)

Under the Supervision of

Mr. Unmesh Shukla

(Supervisor)

Department of Computer Science and Engineering

and

Ms. Geetanjali Bhola

(Co-Supervisor)

Department of Computer Science and Engineering

Faculty of Technology

University of Delhi

NEW DELHI – 110007

(2025 – 2026)

Declaration

We, the undersigned, hereby declare that this project report titled "**Eligify - Smarter way to check, Track & Predict Exam Eligibility**" is our original work and has not been submitted as in part or full to any discipline of course. The work was carried out under the supervision of **Mr. Unmesh Shukla** and **Ms. Geetanjali Bhola** at Faculty of Technology, University of Delhi.

Team Member	Roll No.	Signature
Aditya Kumar Yadav	24293916005	
Pranjal Gupta A	24293916057	
Gaurav Yadav	24293916123	
Tejaswi Anand	24293916146	
Bhakti	24293916046	

Certificate

This is to certify that the project report entitled "**Eligify - Smarter way to check, Track & Predict Exam Eligibility**"

Submitted By

S. No.	Team Member	Roll No.
1	Aditya Kumar Yadav	24293916005
2	Pranjal Gupta A	24293916057
3	Gaurav Yadav	24293916123
4	Tejaswi Anand	24293916146
5	Bhakti	24293916046

in fulfilment of the requirements for Skill Enhancement Course (SEC) of Semester III of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out under our supervision. The work has been found satisfactory for submission.

Mr. Unmesh Shukla
(Supervisor)

Date:

Place:

Ms. Geetanjali Bhola
(Co-Supervisor)

Date:

Place:

Acknowledgement

We wish to express our sincere gratitude to our project supervisor, **Mr. Unmesh Shukla**, and our co-supervisor, **Ms. Geetanjali Bhola**, for their valuable guidance, continuous support, and constructive criticism throughout the duration of this project.

Finally, we would like to thank our friends and family for their unwavering encouragement.

Abstract

Eligify is a web-based examination eligibility assessment system designed to automate and simplify the process of determining whether a candidate is eligible for various competitive examinations in India.

The system integrates rule-based eligibility checking, OCR-powered document extraction, and academic verification to ensure accurate and real-time decision-making. Using Flask, OCR (Tesseract), SQLAlchemy, and Tailwind CSS, the system provides a secure, modular, and scalable platform for centralized eligibility checking.

The project demonstrates modern document processing techniques, strong security architecture, and a clean MVC-based backend and frontend structure.

Table of Contents

Keywords

Examination Eligibility, OCR (Optical Character Recognition), Rule Engine, Flask, Academic Verification, Application Tracking, Automation, Python.

List of Figures

List of Abbreviations & Symbols

1. **OCR:** Optical Character Recognition
2. **MVC:** Model View Controller
3. **UML:** Unified Modeling Language
4. **CSRF:** Cross-Site Request Forgery
5. **XSS:** Cross-Site Scripting
6. **API:** Application Programming Interface
7. **ORM:** Object Relational Mapping

1. Introduction

1.1 Background

Every year, thousands of students miss critical opportunities due to scattered and unorganized information about exam eligibility, deadlines, and counselling processes. This lack of a centralized, reliable platform results in stress, uninformed decisions, and diminished chances for academic growth. Eligify aims to address this challenge by creating a one-stop solution.

1.2 Problem Description

Currently, exam eligibility criteria are spread across many websites and official announcements, confusing students. Undergraduate aspirants especially face two major challenges:

- Difficulty in understanding and comparing eligibility criteria across different exams.
- Struggles with post-exam processes such as counselling and college/course prediction. The lack of a centralized, personalized platform wastes time and causes students to miss deadlines or make poor choices.

1.3 Objective

The primary objective is to build a technology platform that simplifies eligibility checks, notifications, and post-exam decision-making.

- **Check:** Build a rule-based engine for instant eligibility checks against exam rules (age, qualifications) with high accuracy.
- **Track:** Provide a personalized dashboard showing eligibility results, deadlines, and alerts.
- **Predict:** Offer guidance on college and course options.

1.4 Scope of the System

The platform is designed to be scalable, supporting both Indian and international examinations. It employs a hybrid architecture (Check, Track, Predict) to ensure a smooth student experience.

1.5 Proposed Solution Overview

Eligify acts as a central platform for exam eligibility and admissions globally. It saves students time searching for scattered information and provides accurate, real-time eligibility checks and application tracking.

1.6 Target Users & Stakeholders

- **Primary Users:** Students/Aspirants (Undergraduate and Competitive Exam candidates).
- **Stakeholders:** Educational Institutions, Examination Bodies.

1.7 Organization of the Report

The report is organized into sections covering Requirement Analysis, System Design, Implementation details, Testing strategies, Results, and Future Scope.

2. Requirement Analysis

2.1 Requirement Gathering Process

The analysis identified that existing systems (Manual Checking, PDF Readers, Government websites) are slow, error-prone, and lack centralization. This highlighted the need for automated eligibility systems and OCR-based document extraction.

2.2 Functional Requirements

- **Eligibility Checker:** Instant check of student profiles against exam rules.
- **Personalized Dashboard:** Display eligibility results, deadlines, and alerts.
- **Document Parsing:** Extract text from PDFs and Marksheets using OCR or Text Layer extraction.
- **Academic Verification:** Verify marks and credentials against rules.

2.3 Non-Functional Requirements

- **Accuracy:** 100% accuracy in eligibility rule evaluation.
- **Performance:** Text extraction < 200ms; OCR processing 3-6 seconds.
- **Security:** CSRF protection, XSS sanitization, and secure file handling.

2.4 User Stories / User Scenarios

- **Student A:** "As a student, I want to upload my marksheet so that the system can automatically tell me which exams I am eligible for."
- **Student B:** "As a student, I want a dashboard to track upcoming deadlines for exams I am eligible for."

2.5 Feasibility Study

The feasibility of ELIGIFY was validated by testing each core feature through small working prototypes. OCR extraction was first checked using real marksheets, confirming that Tesseract and preprocessing could reliably extract academic data. A basic rule engine was implemented to ensure exam eligibility conditions could be evaluated accurately and consistently. The verification logic was tested by comparing extracted values with user inputs, proving that tolerance-based matching worked correctly. Finally, combining OCR, verification, and rule evaluation on a local Flask server confirmed the project was technically doable within the given time.

3. System Design

3.1 System Architecture

ELIGIFY uses an MVC-based (Model-View-Controller) architecture on both frontend and backend:

- **Model:** Candidate, Exam, Academic Verification.
- **View:** Tailwind-based frontend screens.
- **Controller:** Flask API routes.

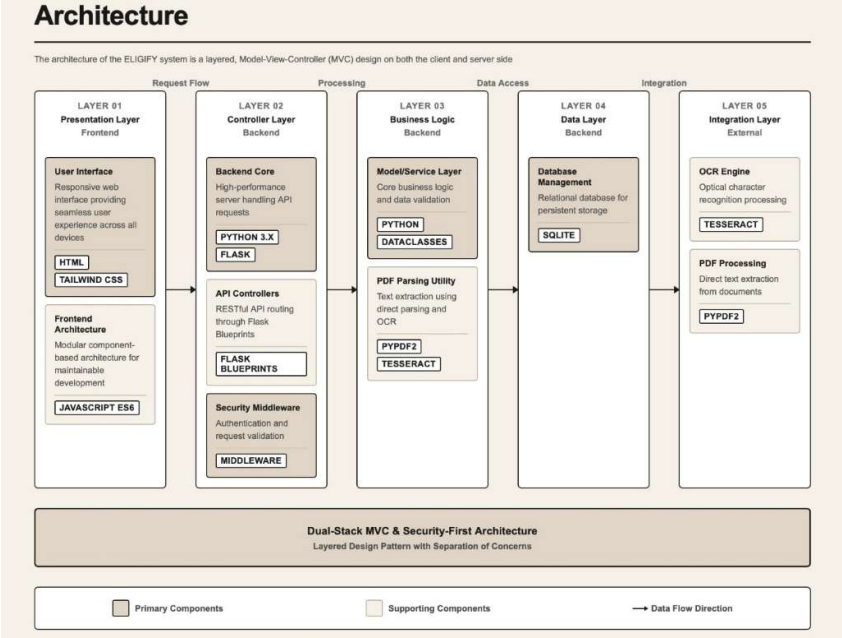

Fig 3.1: System Architecture

3.2 Data Flow

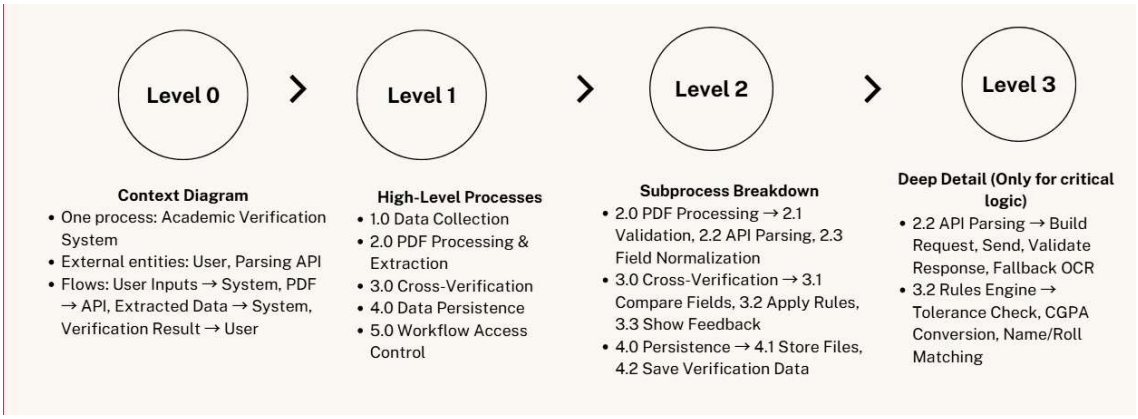

Fig 3.2: Data Flow Diagram

3.3 Database Schema

The system uses a normalized design with 9 core tables which supports Audit trails and JSON storage for flexible data-

1. User management (1 table)
    1. users — Google OAuth user information
2. Candidate profiles (1 table)
    1. candidate_profiles — Candidate academic and personal information
3. Document management (3 tables)
    1. document_uploads — Uploaded document audit trail
    2. parsed_documents — Extracted data from documents (JSON)
    3. academic_verifications — Verification results
4. Examinations (3 tables)
    1. exams — Examination metadata and eligibility criteria
    2. exam_subjects — Subjects required per exam
    3. exam_documents — Required documents per exam
5. Eligibility checking (1 table)
    1. eligibility_checks — Historical eligibility check records

Fig 3.3: Database ER Diagram
(Next Page)

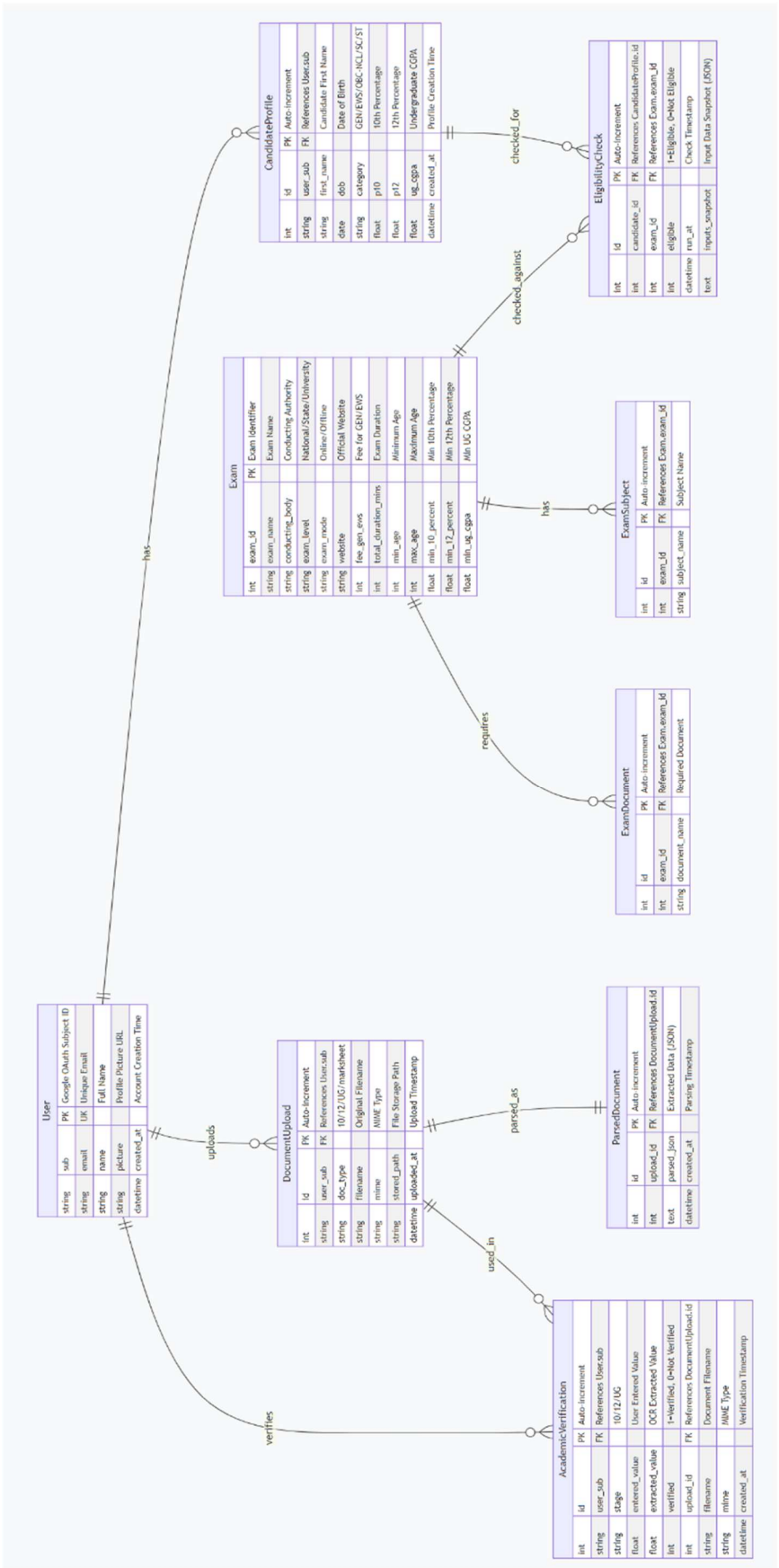

Fig 3.3: Database ER Diagram

4. Implementation

4.1 Technology Stack

- **Backend:** Python 3.8+, Flask 2.0+, SQLAlchemy ORM.
- **Frontend:** Vanilla JS (ES6 Modules), Tailwind CSS, HTML5.
- **Database:** SQLite (development), PostgreSQL (production).
- **OCR/Processing:** PyPDF2, pdf2image, OpenCV, Tesseract OCR 4.0+, Poppler utilities.

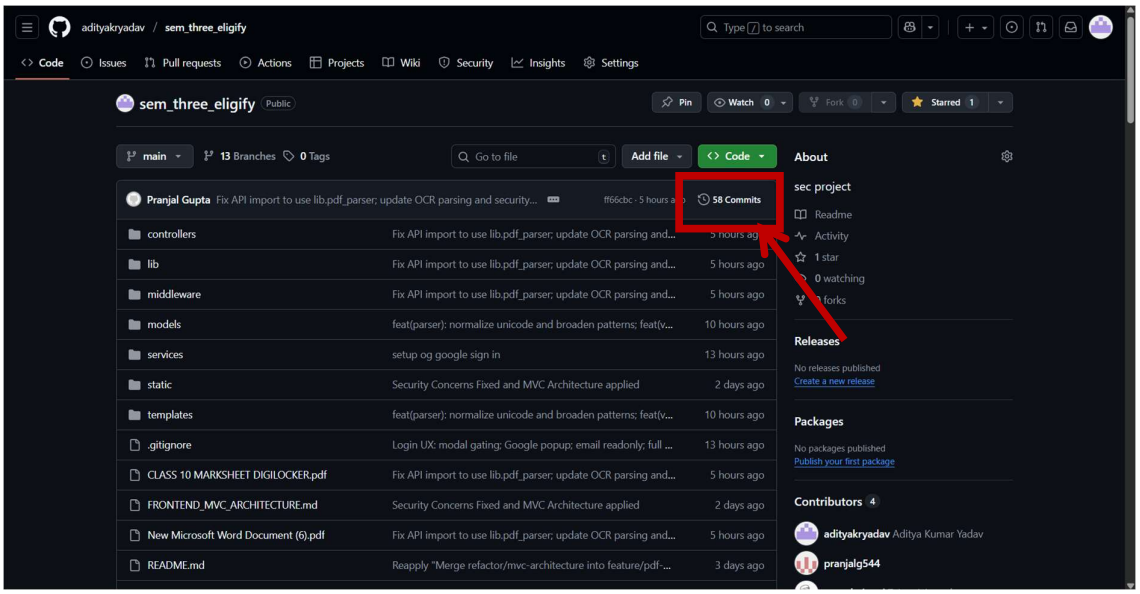

Fig 4.1 Github Repository

4.2 Backend Implementation

The backend is built on Flask. It handles API requests, file processing, and interaction with the database via SQLAlchemy. It implements a **Dual-mode extraction** system (Text Layer for fast extraction, OCR for scanned files) and Regex-based parsing.

4.3 Frontend Implementation

The frontend utilizes an MVC structure with modular services. It interacts with the backend via secure API calls and displays results using Tailwind CSS components.

4.4 API Endpoints

The backend exposes the following REST API endpoints to support PDF processing, academic verification, and eligibility evaluation:

- GET /api/exams – Fetch the complete list of supported examinations.
- POST /api/candidate-profile – Save or update the candidate's personal and academic profile.
- POST /api/parse-pdf – Extract raw text content from the uploaded PDF document.
- POST /api/parse-marksheet – Extract structured academic fields from the marksheet (total marks, obtained marks, identifiers).
- POST /api/verify-academic – Validate extracted academic data against the candidate's declared values.

- POST /api/check-eligibility – Evaluate eligibility using the rule engine based on verified academic information.
- GET /api/status – Check server health and service availability.

4.5 Security Mechanisms

- **Authentication:** Google OAuth 2.0.
- **Protection:** CSRF protection, XSS sanitization.
- **Rate Limiting:** Implemented to prevent abuse.
- **Input Validation:** Secure file-handling pipeline; Invalid PDFs are rejected.

4.6 Logging & Error Handling

The system includes audit trail support in the database schema to track verification actions and system errors.

4.7 Integration of Components

The Rule Engine integrates with the Document Verification Module and the Database ORM Models to provide a seamless "Check" experience.

5. Testing & Quality Assurance

5.1 Testing Strategy

The project employed a comprehensive testing strategy focusing on OCR accuracy, rule engine logic, and security vulnerabilities.

5.2 Test Case Design

Test cases covered:

- Standard digital PDFs.
- Standard marksheet formats.
- Scanned images (OCR stress test) - FAILED
- Non-standard marksheet formats -FAILED
- Boundary age and marks conditions.

5.3 Unit Testing

- **Eligibility Engine:** Achieved 100% accuracy in rule evaluation.
- **Document Parsing:** ~50% success rate.

5.4 Integration Testing

Verified the flow between PDF upload -> Text Extraction -> Rule Engine -> Database.

5.5 System Testing

Tested the entire pipeline with 50 marksheets (10th, 12th, UG).

5.6 User Acceptance Testing

Validated that the system accurately provides eligibility results and explanations to the end-user.

5.7 Test Results Summary

- **Eligibility Accuracy:** 100%.
- **Academic Verification:** 90% accuracy.
- **Security:** 0 XSS vulnerabilities; Invalid PDFs rejected.

6. Local Installation

6.1 Local Installation Environment & Tools

- **Languages:** Python 3.8+, JavaScript (ES6).
- **Frameworks:** Flask, Tailwind CSS.
- **Tools:** Tesseract OCR, Poppler.

6.2 Installation & Configuration Steps

```
# 1. Install Tesseract OCR
# Download from: https://github.com/UB-Mannheim/tesseract/wiki
# Install to: C:\Program Files\Tesseract-OCR\

# 2. Install Poppler
# Download from: https://github.com/oschwartz10612/poppler-windows/releases/
# Extract to: C:\Tools\poppler-24.08.0\

# 3. Set Environment Variables (PowerShell)
$env:TESSERACT_CMD = 'C:\Program Files\Tesseract-OCR\tesseract.exe'
$env:POPPLER_PATH = 'C:\Tools\poppler-24.08.0\Library\bin'

# 4. Setup Python Environment
cd eligify-3\eligify
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install -r requirements.txt

# 5. Create .env.local file
# Add: SECRET_KEY=your-secret-key-here
# Add: GOOGLE_CLIENT_ID=your-client-id (optional)

# 6. Run Application
python app.py

# 7. Open Browser
# Navigate to: http://localhost:3000
```

6.3 Hosting/Deployment Architecture

The system supports SQLite for local development and PostgreSQL for production deployment. It is designed to be scalable.

7. System Output & Demonstration

7.1 Screenshots of the Final System

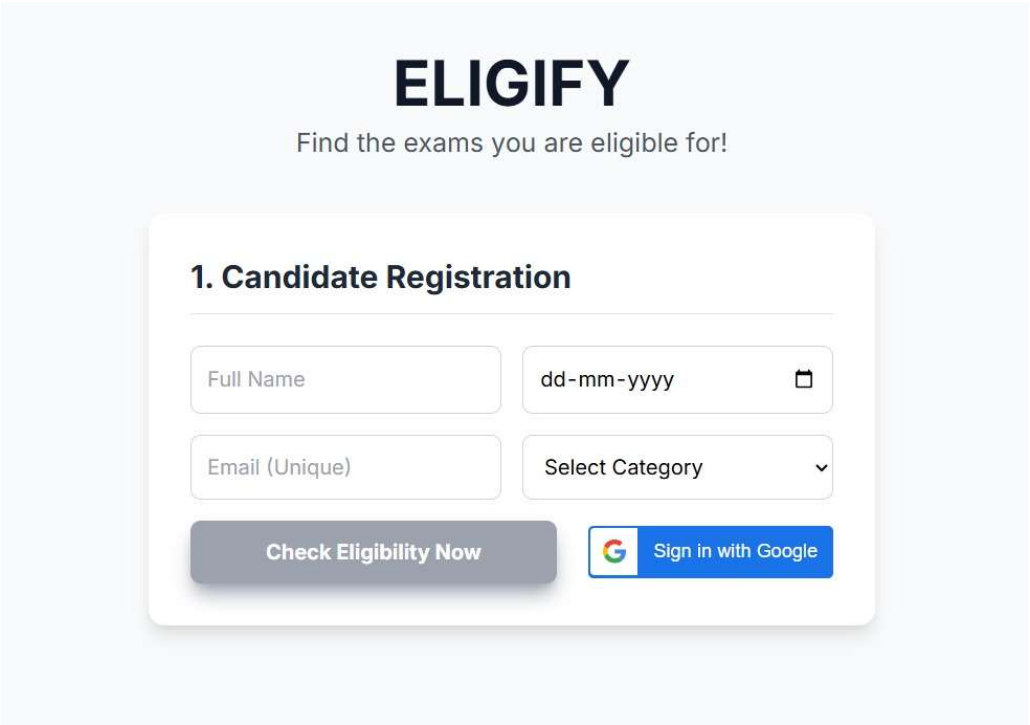

Fig 7.1 Webpage Dashboard

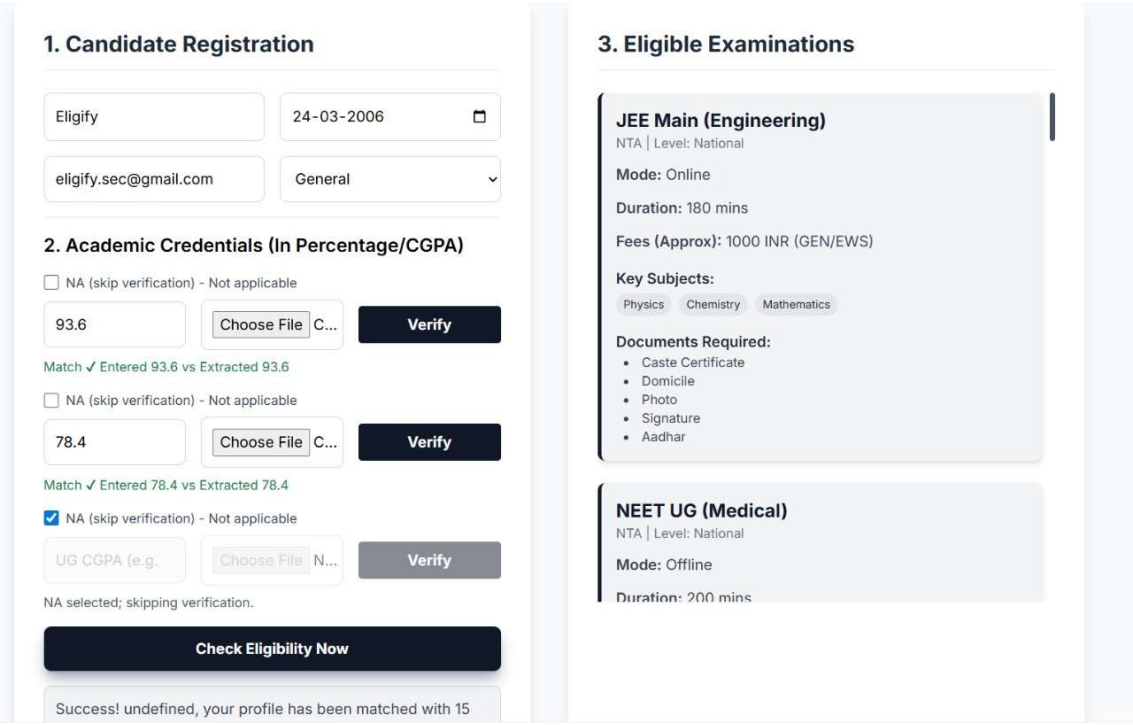

Fig 7.2 Checking Eligibility

7.2 Workflow Demonstration

- **Profile Creation:** Student enters basic details.
- **Upload:** Marksheet uploaded.
- **Processing:** System parses data (Example: Physics 85, Maths 90).
- **Result:** "Eligible for JEE Mains" or "Not Eligible (Age limit exceeded)".

8. Project Management

8.1 Team Roles & Responsibilities

- **Aditya Kumar Yadav:** Developer/Member- PDF Parsing & Data Extraction Feature
- **Pranjal Gupta A:** Developer/Member- Google OAuth Sign-in Feature
- **Gaurav Yadav:** Developer/Member- Data Insertion in Database
- **Tejaswi Anand:** Developer/Member- MVC Architecture Implementation
- **Bhakti:** Developer/Member- Database Design & Creation

8.2 Risk Assessment & Mitigation

- **Risk 1:** OCR failure on poor quality images.
    - *Mitigation:* Dual-mode extraction (Text layer + OCR fallback).
- **Risk 2:** Data privacy.
    - *Mitigation:* Secure file handling and sanitization middleware.

9. Conclusion & Future Enhancements

9.1 Summary of Implementation

Eligify successfully fills a major gap in the education system. It simplifies eligibility checks through a Rule-based engine, OCR-driven document analysis, and a secure web architecture. The system achieves high accuracy and strong validation, solving a real-world problem.

9.2 Achievements

- **Accuracy:** 100% eligibility accuracy and 100% document parsing success on selectable text Documents
- **Speed:** Reduced manual checking time significantly.
- **Architecture:** Built a secure, modular MVC system.

9.3 Limitations

- **Regex Extraction:** Fails on non-standard marksheets.
- **OCR:** Works well for digital PDFs but struggles with low-quality scanned ones.

9.4 Future Scope / Improvements

- **Phase I (Immediate):** Increasing the sample space for more precision of the project through rigorous testing.
- **Phase II (Personalization):** Personalized Attempt Counter (e.g., for UPSC CDS), Visual Eligibility Timeline, Proactive Exam Notifications, and Integrated Application Tracker.
- **Phase III:** Blockchain Verification for immutable academic records and credential portability.

References

1. Python Software Foundation, Python Language Reference, Python.org.
2. Pal, G., & Smitha, K. "Optical Character Recognition using Tesseract OCR Engine," International Journal of Computer Applications.
3. Google Developers, Google OAuth 2.0 Documentation, developers.google.com.
4. Flask Documentation, Flask Web Framework, https://flask.palletsprojects.com/.
5. SQLAlchemy Documentation, SQLAlchemy ORM, https://www.sqlalchemy.org/.
6. Smith, R. "An Overview of the Tesseract OCR Engine," Proceedings of ICDAR, 2007.