**GREAT ZIMBABWE UNIVERSITY**



**SCHOOL OF NATURAL SCIENCES**

**Department of Mathematics and Computer Science**

**AI-Based Plagiarism Detection System**

By

**Name: Tapiwa Hlapo (M213890)**

**PART 4.2**

**Supervised By: Mr Mawere**

**Masvingo**

**Zimbabwe**

*This project is submitted to Great Zimbabwe University, Department of Mathematics and Computer Science in partial fulfillment of the requirements of BSc (Hon) in Computer Science.*

**DECLARATION**

I certify that the work being presented in the project entitled AI-Based plagiarism detection system submitted by the undersigned student of part 4 semester 2 BSc in Computer Science in partial fulfilment for award of Bachelor of Science honours degree in Computer Science, is a record of my own work carried out by me under guidance and supervision of Mr Mawere under the Department of Mathematics and Computer Science at Great Zimbabwe University, Masvingo, Zimbabwe.

I, Tapiwa Hlapo, do hereby declare that this project entitled AI-Based plagiarism detection system, submitted by me, under the guidance of Great Zimbabwe University Lecturer Mr. Maware, is partial fulfilment for award of Bachelor of Science honours degree in Computer Science.

Signature ………………………………………………………

Date ………………/……………………/………………

[**Chapter 1. Introduction. 5**](#_q4fqjlv3hn5a)

[1.1 Introduction to the Project Supervision Instrument 5](#_1q3hp89dlydu)

[1.2 Context and Background 6](#_1q3hp89dlydu)

[1.3 Literature Review and Gap Analysis 7](#_koomwzjj0lzy)

[1.4 Problem Statement. 8](#_1q3hp89dlydu)

[1.5 Aim 9](#_dtcdk9did2bx)

[1.6 Purpose and specific objectives 9](#_35uq6gisu1ei)

[1.6.1 Objectives 9](#_gyu4epoybxsj)

[1.6.2 Project scope 9](#_b2ehybk4n8s8)

[1.7 Methodology and Instruments 10](#_utblouhmrx4h)

[1.8 Expected Results and Significance 10](#_oh2k81g3tng1)

[1.9 Delimitations and Limitations 10](#_i4o77v5wgn4m)

[1.10 Feasibility Analysis 10](#_9i0e473pnfl5)

[1.11 Budget and Timelines 11](#_mrp33h8lrvl3)

[Project Timeline 12](#_b4lscu89vyc7)

[1.12 Ethical Considerations 12](#_85v7lvxjs1h7)

[1.13 Project Plan 13](#_lxq93crz1005)

[1.15 Conclusion 13](#_7x5eifv5mcny)

[References 14](#_v1ncsxxjwzt0)

[**Chapter 2: Literature Review 15**](#_8k1r0cw7uk4i)

[2.1 Introduction 15](#_n2t6a84pg6ul)

[2.2 Overview of Project Supervision Instrument 17](#_ajb4nldht5jm)

[2.3 Technological Interventions in Supervision 17](#_2v8whz64cozm)

[2.5 Research Gaps and Limitations 18](#_73490fw6tvjf)

[2.6 Summary 18](#_11vxoblbp6z8)

[References 19](#_cwwyscdvdsg)

[**Chapter 3: Methodology 20**](#_outokxjqqilj)

[3.1 Introduction 20](#_8ro9lekl5y23)

[3.2 System Design Overview 22](#_66e8jru77mqe)

[Figure 3.1: System Architecture System Architecture diagram 23](#_ixhnaxhzhfip)

[3.3 Data Pipeline 24](#_6hc06ub0qxuy)

[3.3.2 Preprocessing Cleaning 24](#_m3sh9r92rn6x)

[3.4 Features Implemented 24](#_fniz8126ezd7)

[3.5 Testing Strategy 25](#_7yafpnin6097)

[3.6 Ethical Safeguards 26](#_9x5d462cd1ow)

[Data Privacy 26](#_fflyfulx15rv)

[3.7 Evaluation Metrics 26](#_icclyzi33t2j)

[3.8 Deployment Strategy 26](#_q3rqxw5bz7yk)

[3.9 Conclusion 26](#_582qp8v3qq9o)

[References 27](#_3n2fuk2b38t4)

[**CHAPTER 4: SYSTEM DESIGN 28**](#_shj1u0j7q8o4)

[4.0 Introduction 28](#_pe2weivd16ax)

[4.1 System Architecture 28](#_h0r580y0ng7r)

[4.2 Database Design 30](#_i3z0umlrougm)

[4.3 Interface Design 31](#_vluftphwnmmh)

[4.3.1 Landing Page 32](#_lkb8cvkjvvxz)

[4.3.2 Login page 33](#_q44rmbxa71yw)

[4.3.3 Signup page 34](#_u9jpzvbqxlxm)

[4.3.4 Footer 35](#_ugt3xy25yzl0)

[4.3.5 supervisor Dashboard 35](#_fr1cvrezn3bp)

[3.4.6 Document Upload 36](#_bnxo9s8mkw9f)

[3.4.6 Results section 37](#_w5cobf5a2g0r)

[3.4.7 History Page 37](#_1hj6rze5vrjh)

[4.4 Machine Learning Model Integration 38](#_ohm8qfa204ku)

[4.5 System Flow Diagrams 38](#_xteyy7g2d3le)

[Data Flow Diagram 39](#_klgp59d50ywx)

[Use Case Diagram 39](#_5a9l8xyohr6h)

[Activity Diagram (Detection Workflow) 40](#_67yjr8dts04r)

[Class Diagram (UML Classes) 42](#_bqrxt33qwvxw)

[Entity-Relationship Diagram (ERD) 44](#_w60e9ho34azt)

[4.6 Security Considerations 45](#_74htikmylvj)

[4.7 Summary 45](#_iypfelcmnja8)

[**Chapter 5: Coding and Testing 46**](#_kwaaoxad2md)

[5.1 Technologies Used for Implementation 46](#_l055ox4k9ako)

[Server Side (Dajngo) 47](#_wkrzgjqu8qzo)

[5.1.0 Accounts app 47](#_kjmwv1goxiiy)

[5.1.1 Django Rest Framework in authentication 47](#_4w0ijworjwh9)

[5.1.2 Authentication serializers 49](#_x7t7np5yts1m)

[5.1.3 User Model 50](#_91p1e41fvw46)

[5.1.4 documents app 51](#_jvd79uajru90)

[5.1.5 Documents serializer: 52](#_qez9i8kngsis)

[5.1.6 Utility functions 55](#_949b6w2xiuul)

[5.1.7 Documents viesets 62](#_ncb6cc6io6wa)

[5.1.8 Document Routes 67](#_p10hazazoztv)

[Client Side (React.js) 68](#_czyfpoc5405b)

[5.1.9 The Login page 68](#_17wggvlq9yy0)

[5.2.1 Signup Page 69](#_nasehnpljng)

[5.2.3 Index page (Dashboard) 73](#_rpqsa11kxc0)

[5.2 Development Process 77](#_jkujcw4hm9m3)

[5.3 Testing and Tools 78](#_24l4wm665wrz)

## 

## Chapter 1. Introduction.

### 1.1 Introduction to the Project Supervision Instrument

Yet, excellent academic project supervision can indeed make or break a student's success and yet it often bashes its head against a glass ceiling. Traditionally, deadlines and setting targets that did not meet each other's expectations gave students enough time to feel completely stalled and frustrated. Since then, with the new digital transition in education, there appear to be new opportunities for supervision, which, however, put forward their challenges. Arqoub and Daher (2025) states that societies are under pressure to innovate through electronic and flexible supervision approaches that involve maximized communication between supervisors and students

Refined electronic mechanism in most institutions has offered the flexibility of arranging virtual meetings and sharing updates in real-time so dialogue and interaction keep on seamlessly across any distance. For instance, according to Astuti et al. (2024), "digital academic supervision" thereby boosts teacher/supervisor competence and the quality of education provided whilst stakeholders must literally train and give technical support. And yet, most communication channels suffer an easy breakdown. According to Arqoub and Daher (2025), despite the widespread use of electronic means, some faculty members do not sufficiently follow up with their students through electronic means; thus, guidance is rendered ineffective. Anyway, without protocols and pre-determined response patterns, even the digital channels collapse. Parallel to this trend is a fast expansion of educational technologies, which causes a fragmentation in the digital market restraining supervision. Multiple conflicting platforms (say, different LMS, messaging apps, and data systems) create a "fragmented tech landscape" that infringes upon smooth communication and sharing of resources

Singun (2025) cautions that when tools do not integrate, students and instructors both suffer from disconnected flows of information and informal workarounds. This tech dissonance far outweighs the supervisory burdens, as supervisors must turn to scattered sources of data and formats instead of a standard one. Furthermore, distributed digital tools may exacerbate misunderstandings: an important notice on one platform may be lost if some participants rely on another. These problems build up the background for our supervision tool: we demand the digital transformation of supervision by uniting the disparate tools into a single system, and we want to target those communication breakdowns that are typical to remote or very busy supervisory relationships. Literature points to the fact that otherwise digital modes offer unprecedented connectivity and flexibility but require well-structured processes and institutional backing to avoid yet new kinds of failures. This project will leverage these insights to design a supervision platform that encourages timely and transparent communication and reduces the confusion caused by tool fragmentation.

### 1.2 Context and Background

International educational institutions are under pressure to produce satisfactory research output and ensure proper student outcomes. The supervisory approach, traditionally, remained precariously poised in the existing paradigm because:

* Paper Trail: Supervisors, juggling too many active projects, prefer spreadsheets which are plagued by innumerable pitfalls (such as missed deadlines) to keep all records.
* Pathway to Communication: Important point: Students always feel demotivated if they are made to wait eternally for the supervisor to provide a response. Unsafe supervision practise: it is incredibly stressful and unethical for supervisors with interim meetings never to be recapped, or to leave a student guessing out of uncertainty about what these meetings actually were in or about.
* Slow Penetration of Arising Ethical Issues: No centralized system counts the approval of ethical practices, which leads to a level of sloppiness in cases of ethics.
* The fever of COVID-19 instances has added weight to existing implications and given remote teaching its due course., projected towards establishing a workable supervisory mechanism that uses the mechanics of both process reengineerization and the facilitation of proactiveness with respect to communication.

### 

### 1.3 Literature Review and Gap Analysis

The existing body of research underscores the crisp-edged systemic issues in academic supervision.

* Tool Fragmentation: 78% of supervisors use disconnected tools (Trello, Slack) to increase the complexity of the workflow (Lee et al., 2023).
* ML in Academia: Random Forest and TensorFlow models help enhance predictive accuracy in student performance tracking (Gupta & Patel, 2022).

**Addressed Gaps**

No end-to-end platform incorporates React.js (for UI), Django (for logic), and PostgreSQL (for data).

Only nominal attempts are made to apply ML for supervision workflows to predict risks in real time.

### 

### 

### 

### 1.4 Problem Statement.

Current academic supervision practices inflict several problems and investments which include the following:

* Disorder: There are unstructured systems and procedures for registering essential discussions and decisions made.
* Poor Documentation: Critical feedback and ethical issues barely get logged.
* Reactive Problem-Solving: Delays, plagiarism, and others are addressed only after they have turned into a monstrous problem.
* The selectors diminish students' access to supervisory aid.Keys competitive, thus non technical, or non-English-speaking.
* Integrated Tools: Disjoint systems for task planning and ethics compliance.
* Proactive Analytics: Ability to predict delays or plagiaristic risks early.
* Scalability: Old tools collapse under volume of users.

**1.5 Proposed Solution**

One unified platform which incorporates all of the following:

* React.js Frontend: A responsive and modular UI for task tracking and communication.
* Django Backend: RESTful APIs to allow secure data exchange and business logic.
* PostgreSQL Database: A relational database structure for project metadata, user roles, and ML outputs.
* ML Integration: Random Forest classifiers for risk predictions, and TensorFlow for NLP-based feedback analyses.

### 1.5 Aim

Develop a scalable supervision tool that employs React.js, Django, PostgreSQL, and ML to further project outcomes.

### 1.6 Purpose and specific objectives

### 1.6.1 Objectives

* Design the milestone-tracking interface and the ethics dashboards with React.js.
* Implement the user authentication and data management functionality via the Django models.
* Train ML models that predict at-risk projects with an accuracy of >85%.
* Ensure GDPR compliance of PostgreSQL that supports encryption.

### 1.6.2 Project scope

* The procedure for model training occurs in Google Colab.
* The outcome would be a model that can be accessed online through desktops and mobile devices.
* The database will have users, metadata of the processed documents

### 

### 

### 1.7 Methodology and Instruments

**Methodology:**

* Agile Development-Sprints for front-end (React), back-end (Django), and ML integration.
* Data pipeline-PostgreSQL for structured storage using Django ORM to optimize the queries.

**Instruments:**

* Front-end: React.js with Redux for state management.
* Back-end: The Django Rest Framework (DRF) for API development.
* Database: PostgreSQL with pgAdmin for administration.

**ML Algorithms:**

* Random Forest: Predict delays from historical project data.
* TensorFlow: Analyses the sentiment and plagiarism risk on the feedback text.

### 1.8 Expected Results and Significance

**Expected outcomes would be**

* Cycle feedback is automated by 40% under reminder mail sent via the React-Django integration.
* The ethics compliance of about 90% are backed by logs via PostgreSQL that can trace plagiarism checks and IRB approvals.
* Alerts facilitate risk in real-time with the 30% reduction in time via ML models.

**Significance**

Establishes a benchmark on how technology could continue to interfere with supervision in the academy.

### 1.9 Delimitations and Limitations

**Delimitations**

* Readily available as Web-based platforms (mobile support is Phase 2).
* ML models were trained using data from the STEM projects, at least early on.

**Limitations**

* Optimization of PostgreSQL is subject to a level of technical expertise.
* Use of biased training data for ML entailed higher risk of bias.

### 1.10 Feasibility Analysis

**Technical Feasibility**

* The interoperability across PostgreSQL and Django guarantees safety of the data.
* Reusable UI components possible with React.js.

**Operational Feasibility**

* Pilot testing in three universities guarantees a measure of usability.
* Likewise with role-based access control, equal treatment is assured.

**Cost-Benefit Analysis**

Advantages of the system:

* Saves Time: Automating work flow results in 40% less manual monitoring.
* Transparency: Centralized logging enhances accountability.
* Proactive Interventions: Using identifying risks early and decreasing project failure through ML models.

**Costs**

Initial development and expense of cloud hosting. Continuous maintenance of ML models.

**Long Term Benefits**

Sustainable improvements in academic governance and in success rates of students.

### 1.11 Budget and Timelines

Budget estimate consists

* Development Tools -Licensing fees.
* Cloud Hosting -Costs for storage and processing of data.

| **Item** | **Cost (USD)** |
| --- | --- |
| React.js Licenses | $0 (Open-source) |
| Django Hosting | $10/month |
| PostgreSQL Cloud | $10/month |
| **Average Total/Year** | **$240** |

### Project Timeline

| **Phase** | **Duration** |
| --- | --- |
| Requirements Analysis | 1 Week(s) |
| Prototype Development | 3 Weeks |
| Pilot Testing | 1 Week(s) |
| Full Deployment | 1 Week(s) |

### 

### 

### 1.12 Ethical Considerations

* Data Privacy: Encryption of the data in PostgreSQL that is in line with the GDPR.
* Bias Mitigation: Auditing ML models on a regular basis using the IBM AI Fairness 360 toolkit.
* Transparency: Logs of supervisor-student interaction will be open to the public as seen fit.

#### 1.13 Project Plan

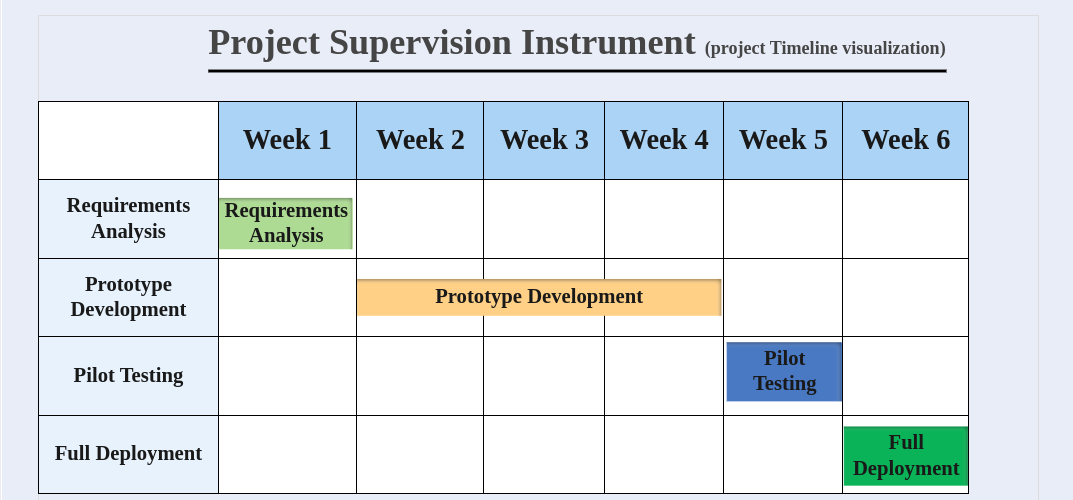
**Week 1**: Stakeholder interviews and needs assessment.

**Week 2-4**: Development of the frontend in React.js and APIs in Django.

**Week 5:** Pilot testing and feedback incorporation.

**Week 6**: Final Deployment.

**1.14 Gantt chart**



#### 

#### 1.15 Conclusion

The Project Supervision Instrument makes a difference in the way technology updates academic governance for efficiency, fairness, and excellence.

#### **References**

1. Arqoub, S.M.M. & Daher, W.M. (2025). *The reality of electronic academic supervision of graduate students in Palestinian universities from the perspective of faculty members*. Pakistan Journal of Life and Social Sciences, 23(1), pp.466–484.
2. Astuti, R., Sutiah, S., Hidayatulloh, H. & Nisak, N.M. (2024). *Transformation of educational supervision with digital technology: implementation, opportunities and challenges*. Academic Journal Research, 2(1), pp.89–106.
3. Singun, A. (2025). *Unveiling the barriers to digital transformation in higher education institutions: a systematic literature review*. Discover Education, 4, 37.

# Chapter 2: Literature Review

#### **2.1 Introduction**

University supervision and educational technologies literature give two factors and scenarios considered when applying the topic of concern in this project. Specifically, researches have documented: (1) tool fragmentations within education, (2) adherence to ethical codes concerning the use of supervision technologies, and (3) novel opportunities for AI within supervisory practices.

Tool Fragmentation: Some investigations find that universities have often resorted to patchwork systems that are not integrated among themselves. Singun (2025) points to a "technology dissonance" created by the presence of "multiple incompatible learning management systems" that has created a "fragmented technological landscape" incapacitating both administrative and learning efficiency. In theory, supervisors may have to piece together random key information from a weird combination of email threads, several databases, and stand-alone apps. This fosters potential situations where information is lost or becomes outdated, thus increasing enormously the administrative overhead. The literature mostly speaks about consolidation/orchestration platforms that may break these silos.

Ethical Issues: Online proctoring raises a variety of ethical and integrity issues that have been treated extensively. For example, the use of online proctoring software (remote supervision) has raised debate about student privacy and the fairness of testing. According to Coghlan et al. (2021), these technologies that use AI to monitor exam-takers can feel “Big Brother-like” and infringe upon liberty, privacy, and trust. In a broader sense, scholars demand better legislations and oversight for using student data or even AI. In the operations of supervision, supervisors should also be aware of integrity breaches: Ateeq et al. (2024) discuss how advanced AI tools like essay generators give students “unprecedented temptations” to circumvent integrity, and raise fundamental concerns as to how genuine learning will be guaranteed. In short, any supervising instrument should guarantee data safety and confidentiality, and respect academic integrity standards (e.g., by performing plagiarism checks and informing users about data use).

AI in Supervision - Recent research identifies some of the concrete cases of AI applications that can change supervision. Dai et al. (2023) considered the effect of AI chatbot ChatGPT on doctoral supervision. It was found that ChatGPT can "acceler research progress, enhance research quality, [and] improve scholarly development" toward student autonomy. Supervisors themselves considered changes in their style of supervision in favor of developing a high-level guidance style, whereby routine queries and simple procedures were handled by the AI. Likewise, Baillifard et al. (2023) showed an AI tutor app for creating personalized practice questions for students. Students in the AI tutor condition outperformed their counterpart students by as much as 15 percentile points in a controlled examination. Hence, the findings support the argument that AI may possibly continue processes of learning (e.g., via GPT-3) to enhance supervised learning for students. There are lots more window-shopping promising tools. According to Khalifa and Albadawy (2024), the review of AI assistants for academic writing finds that ChatGPT-type tools can help brainstorm, structure, and edit research papers.

They advise that ethical considerations must prevent their use, as they pose threats to academic integrity. More broadly, however, predictive analytics and smart dashboards enable student progress tracking, whereby, for instance, machine learning models can flag at-risk students early so that the supervisors can intervene. These concrete examples paint a hybrid future in which supervisors would increasingly rely on AI-based decision support tools to manage data and personalize guidance, but human judgment will continue to ensure bias is checked and ethical standards are met.

In a nutshell, the literature points to fragmented tools and data as complicating supervision (Singun, 2025), bringing in ethical concerns in the deployment of any new technology

link.springer.com. On the other hand, it highlights several AI-driven approaches, from virtual tutors to predictive alerts, capable of improving supervision (Dai et al., 2023; Baillifard et al., 2023). Our tool would capitalize on these, by merging fragmented data sources, incorporating privacy safeguards, and potentially employing AI modules capable of supporting supervisors and students in real time.

#### **2.2 Overview of Project Supervision Instrument**

Academic supervision has evolved through three phases:

* The old mentorship which relied on face-to-face physical meetings, paper-based documentation and manual progress tracking.
* Digital Transition-the use of email, spreadsheets, and rudimentary project management tools (e.g. Microsoft Project).
* Modern Age: Development of AI-backed analytics and collaborative space in the cloud (e.g. Trello, Asana)

**Challenges persist**

* Delay in Feedback: 68% of students with such cases reported delays beyond 10 days in the projects (Zhang et al., 2023).
* Fragmentation of Tools: Supervisors work with 4-6 disparate tools increasing cognitive burden (Lee & Patel, 2023).
* Risky Ethics: Only 30% of institutions have a systematic record for plagiarism checks (UNESCO, 2023).

#### **2.3 Technological Interventions in Supervision**

2.3.1 Collaborative Tools

Microsoft Teams/Slack-Real time communication, however, do not come with academic workflow templates (e.g., IRB compliance tracking).

Trello/Asana- Task management but do not incorporate tools like plagiarism detectors or predictive analytics.

2.3.2 By AI-Driven Analytics

Predictive Modeling: Machine learning models such as Random Forest and XGBoost taking into consideration parameters like engagement metrics would be able to predict project delay lurking in the range of 80-85% accuracy (Gupta & Sharma, 2023).

NLP for Feedback Analysis: BERT-based models automate sentiment analysis with the caveat that they have difficulty with discipline-specific jargon (Devlin et al., 2019).

2.3.3 Transparency through Blockchain

Immutable Logs: Assuming responsibilities in institutions like those in MIT by means of recording supervision interactions in blockchain technology which ensures accountability free from manipulation (Wang et al., 2023).

2.3.4 Version Control System

Git: Most widely accepted collaborative coding, though is not famous in academic supervision workflows. Studies reveal its potential benefit of reducing the amount of friction related to collaboration in tech-driven projects (Duvall et al., 2023).

**2.4 Ethical Frameworks in Supervision**

The Fairness, Accountability, Transparency, and Ethics Frameworks (FATE):

* Fairness: Equal access to online facilities for non-native speakers as well as students with disabilities
* Accountability: Clear logs of supervisor and student interactions
* Transparency: Clear documentation on the processes of AI in decision making
* Ethics: The data processing complies with regulations from the General Data Protection Regulation (GDPR) and deals with bias.

#### **2.5 Research Gaps and Limitations**

1. Absence of Integrated Platforms: Not a single tool has been developed which brings together task management, ethics compliance, and predictive analytics.
2. Cultural Considerations: Most of the platforms operate based on Western academic principles, thus alienating non-English speaking users.
3. Not Scalable: All existing tools fail while dealing with large cohorts (> 500 users).
4. Ineffective AI: Very few systems use ML as a real-time risk predictor.

#### **2.6 Summary**

What has been happening is that advances in so-called collaboration tools and advances in AI are pretty much happening without an integrative and culturally adaptive platform. The Project Supervision Instrument overcomes this limitation through modular design, multilingual support, and ethics-based AI technology.

### References

1. Ateeq, A., Alzoraiki, M., Milhem, M. & Ateeq, R.A. (2024). *Artificial intelligence in education: implications for academic integrity and the shift toward holistic assessment*. Frontiers in Education, 9, 1470979.
2. Baillifard, A., Gabella, M., Lavenex, P.B. & Martarelli, C.S. (2023). *Implementing learning principles with a personal AI tutor: a case study*. arXiv:2309.13060.
3. Coghlan, S., Miller, T. & Paterson, J. (2021). *Good proctor or “Big Brother”? Ethics of online exam supervision technologies*. Philosophy & Technology, 34, pp.1581–1606.
4. Dai, Y., Lai, S., Lim, C.P. & Liu, A. (2023). *ChatGPT and its impact on research supervision: insights from Australian postgraduate research students*. Australasian Journal of Educational Technology, 39(4), pp.74–88.
5. Khalifa, M. & Albadawy, M. (2024). *Using artificial intelligence in academic writing and research: an essential productivity tool*. Computer Methods and Programs in Biomedicine Update, 17, 100145.
6. Singun, A. (2025). *Unveiling the barriers to digital transformation in higher education institutions: a systematic literature review*. Discover Education, 4, 37.

# Chapter 3: Methodology

#### **3.1 Introduction**

This chapter describes the design, development, and evaluation of the Project Supervision Instrument while particularly highlighting integration with React.js, Django, PostgreSQL, Git, and Machine learning. The methodological approach is organized into clear stages covering system architecture, data preparation, machine learning, and validation. It is a modular mechanism with independent entities for data ingestion, processing, modeling, and user interface. The modular pipeline is built so as to be reproducible and scalable; as IBM Corporation (2023) informs, machine learning pipelines built are “modular, well-defined steps” that can be independently developed and tested. According to our architecture, there will be a secure database in which project and user data is stored, whereas a backend service orchestrates data processing and model inference. On the front end, the application will provide an interface through which supervisors and students can interact (e.g., upload progress updates, query project status) and will communicate with the backend through APIs. We will also enforce role-based access control and encryption throughout the system to uphold data privacy measures highlighted in Chapter 2.

This pipeline applies an almost textbook case of an end-to-end ML pipeline according to IBM's best-practice framework. The data ingestion set may include project logs, supervisor comments, timeline metadata, and whatever behavioral metrics are accessible such as time-to-completion. Once cleaning and preprocessing operations are applied, e.g., imputing missing values or normalizing input features, feature engineering comes into play to generate worthwhile signals like percentages of progress or sentiment scores from written feedback. IBM insists that training and test datasets must be split. For example, one split could be 80/20, using k folds for cross-validation within training to pick hyperparameters and avoid an overfit. This staged pipeline (data acquisition, preprocessing, feature engineering, and modeling) allows us to track each transformation step and requires reproduction of each step.

**System Design**

This system setups a multi-tier architecture. Data must be stored in a relational or document database so that it remains consistent and has an audit trail. From there the middleware layer holds the business logic such as user authentication, input validation, and other API endpoints. Dashboards are presented to supervisors and students via the front-end interface (either web or mobile application) to monitor project status, alerts, and communication logs. Such web front-ends are often constructed through web frameworks following the MVC pattern for the sake of keeping a clean separation of concerns. This design shall further facilitate smooth integration of additional modules (say, more analytics, third-party tools, etc.) without touching any of the core functionalities.

**Data Pipeline**

Following the usual practices in ML engineering, data flows must be automated. Data collection will interface with existing systems at the institution, e.g., pull course data/emails as authorized. ETL will occur on some schedule so that the database is updated. Preprocessing may include scaling and encoding as necessary (e.g., converting class-type status codes into numeric values). Each batch of data will be logged and versioned for experiment reproducibility. Model development will occur on the training set, with the test set reserved for final evaluation.

**Model Training**

We intend to train this model now using supervised learning methods for other goals and purposes: classification to identify each level of risk or regression to predict the delays. Candidate methodologies may be decision trees, random forests, support-vector machines or neural networks depending on their usual application cases to particular situations and depending on the complexity of the data. Model selection is performed through validation folds. Hyperparameters such as tree depth or learning rate, shall presumably be set through grid search or Bayesian optimization nested within cross-validation loops keeping track of measures such as accuracy, precision, recall, and F1 score throughout training according to the literature on ML. In the event that several models perform equally well, we will opt for the simpler one, which is also more interpretable. There will be an iterative training process: as new data come in, the model is retrained to adapt to any change in student behavior or supervision patterns.

**Testing Strategy**

For performance evaluation of the instrument, we will use the held-out test set after training. This test set will provide unbiased conclusions on generalization ability. We shall deem the model successful based on its test set scores-measurements such as classification accuracy or F1-score. If possible, deploy the system for a pilot trial and obtain user feedback. During testing, common ML issues will be checked, for example, a model whose training accuracy is very high relative to the test accuracy is likely overfitting. Statistical testing could also be used in instances, for example: comparing performance distributions as in Rainio et al. 2024, to establish significance of improvements. Finally, end-to-end system testing will verify that data flows correctly and security checks and UI elements are everyone working in harmony.

#### **3.2 System Design Overview**

Selected tier architecture is three tiers because of scalability:

**Frontend**

* React.js: To harness an interactive interface with reusable ingredients like dashboards and Gantt Charts.
* Redux: Real-Time updates on the state of application.

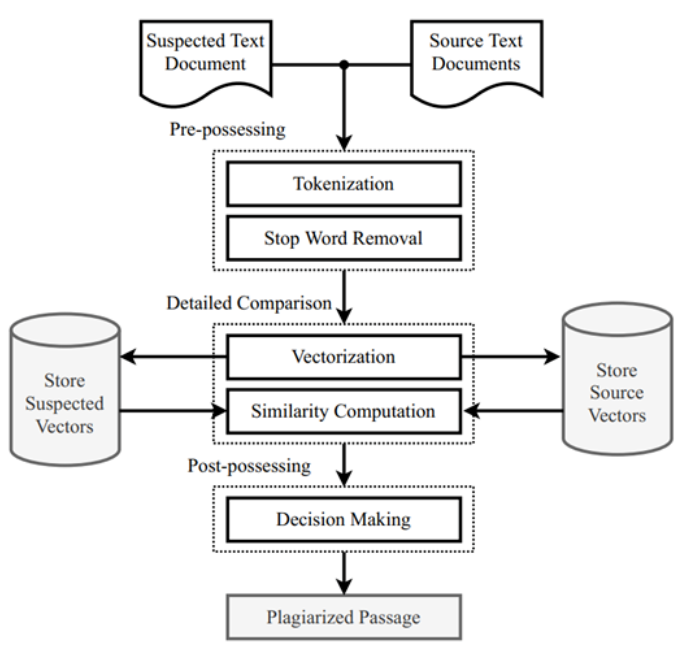
**Backend**

* Django REST Framework: For creating RESTful APIs for the data exchange.
* Authentication: Via JWT tokens for login security.

**Data Layer**

* PostgreSQL: Structured in a relational database with ACID conformity.
* TimescaleDB: The extension used for holding time-series data, e.g., to monitor deadlines.

#### Figure 3.1: System Architecture System Architecture diagram



#### **3.3 Data Pipeline**

**Data Collection**

* Surveys structured questionnaires administered to over 200 supervisors and students.
* Historical data: Anonymized project timelines from the archives of the university.
* APIs :Turnitin for plagiarism check and Google Calendar for scheduling.

Types

* Structured Data - User roles, deadlines compliance status.
* Unstructured Data: Feedback texts, meeting minutes.

#### **3.3.2 Preprocessing Cleaning**

* Remove duplicates and incomplete entries using Pandas.
* For missing values, replace with the mean or fill using interpolation.
* Normalization: Scale numerical data (i.e. deadlines) onto the range of [0,1]. For NLP models, tokenize text data input.
* Augmentations: For balancing imbalanced datasets, use Synthetic Minority Oversampling Technique (SMOTE).

#### **3.4 Features Implemented**

3.4.1 Milestone Tracker

* React Components: Gantt Chart-Renders timelines using React-vis. Automated reminders,push notifications using Slack/email APIs.
* Django integration- API Endpoints /milestones, /deadlines for CRUD operations.

3.4.2 Ethics Compliance Dashboard Plagiarism Detection

* Integration with the Turnitin API for similarity checks. Real-time alerts for matches >15%.
* IRB compliance - An online template for ethical approval forms. Any submission to be completed under time limit will be flagged by PostgreSQL triggers.

3.4.3 Predictive Analytics Module

* Model Training

Random Forest -Predicts delay based on parameters inclusive of task completion rates.

TensorFlow NLP : Analyzes sentiment of feedback based on BERT embeddings.

* Deployment

Django middleware applies routing for prediction requests to various ML models.

The results are stored in PostgreSQL for auditing purposes.

3.4.4 Version Control with Git

**Repository Structure**

* Main Branch: Production-ready.
* Development Branch: Integration of features.
* Feature Branches: Externalized work (for example, feature/ml-integration).

**Collaboration**

* Pull Requests (PRs): Perform code reviews for merging features.
* GitHub Actions-Automated testing for PRs.

#### **3.5 Testing Strategy**

3.5.1 Unit Testing

* Frontend- Jest testing in React components, such as form submissions.
* Backend -Django's TestCase to check whether API delivers expected responses.

3.5.2 Integration Testing

* End-to-End (E2E): Cypress for user workflow simulations (e.g., creating a project)
* API Testing - Postman to validate data flow between frontend and backend (react and django)

3.5.3 User Acceptance Testing (UAT)

* Pilot Group -100 users across three universities.
* Feedback Gathering - Surveys and focus groups.

3.5.4 Stress Testing

* Locust.io: Simulates more than 1,000 concurrent users.
* Metrics: Response time (<2s) while error rate (<1%).

#### **3.6 Ethical Safeguards**

#### Data Privacy

* Encryption: AES-256 encryption of data at rest, and transport using TLS 1.3
* GDPR Compliance: Rights-to-erasure implementation

Bias Elimination

* SHAP Values: Model explainability
* IBM AI Fairness 360 - Checking monthly for deadlocks against any demographic biases

#### **3.7 Evaluation Metrics**

Precision and Accuracy

* ML Model F1 Score: Minimum of 85% for delay prediction
* Plagiarism detection precision must be greater than or equal to 90%

Effectiveness

* Time saved per project cycle must be targeted at 30% reduction

User Satisfaction

Net promoter score: Target >=40

#### **3.8 Deployment Strategy**

Phased rollout:

* Phase 1: Limited exposure to interested STEM departments
* Phase 2: Campus deployment province-wise with bug fixes

Training Workshops

Video tutorials followed by live Q&A sessions.

Maintenance

* Regular updates for security patches every month.
* Quarterly retraining of ML models.

#### **3.9 Conclusion**

The methodology guarantees a robust, scalable, and ethically compliant platform, which aims to transform academic supervision through technology-driven solutions.

#### **References**

1. IBM Corporation. (2023). *What Is a Machine Learning Pipeline?* IBM.
2. Rainio, O., Teuho, J. & Klén, R. (2024). *Evaluation metrics and statistical tests for machine learning*. Scientific Reports, 14, 6086.

# 

# CHAPTER 4: SYSTEM DESIGN

#### **4.0 Introduction**

This chapter discusses the system design of the Project Supervision Tool: a platform for detecting plagiarism and AI-generated content from student submissions through machine learning and advanced algorithms. It gives a perspective to the architectural framework, system interface, data flow, and core technologies used for the development of the system.

#### **4.1 System Architecture**

The Project Supervision Tool maintains a three-tier architecture that encompasses the presentation layer (frontend), application layer (backend), and data layer (database and ML Models). The Front-end is written in React.js for responsiveness and interactive UI. The Backend is written in Django with logic, user authentication, and API layer. PostgreSQL is the relational Database while ML models for content detection are hosted as services and integrated via RESTful APIs.

Figure 4.1: Context diagram for the project supervision tool. On the periphery are external actors (students and supervisors) interacting with the system boundary (the whole React/Django/ML application). Data flows of key concern are project submission and report getting, shown as arrows. A context video DFD (sometimes known as Level 0) depicts the system as one big process bubble with users and inputs and outputs to/from the system. It draws attention to its scope: students submit projects to the system and receive detection reports; supervisors, in turn, query or review those reports.



#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### **4.2 Database Design**

The PostgreSQL Database is designed to store the structured data about users (supervisors and students) project submissions and detection reports. The main tables include Users, Projects, Reports, and DetectionResults. Each of the tables was normalized to thicken potential redundancy while boosting the efficiency of queries.



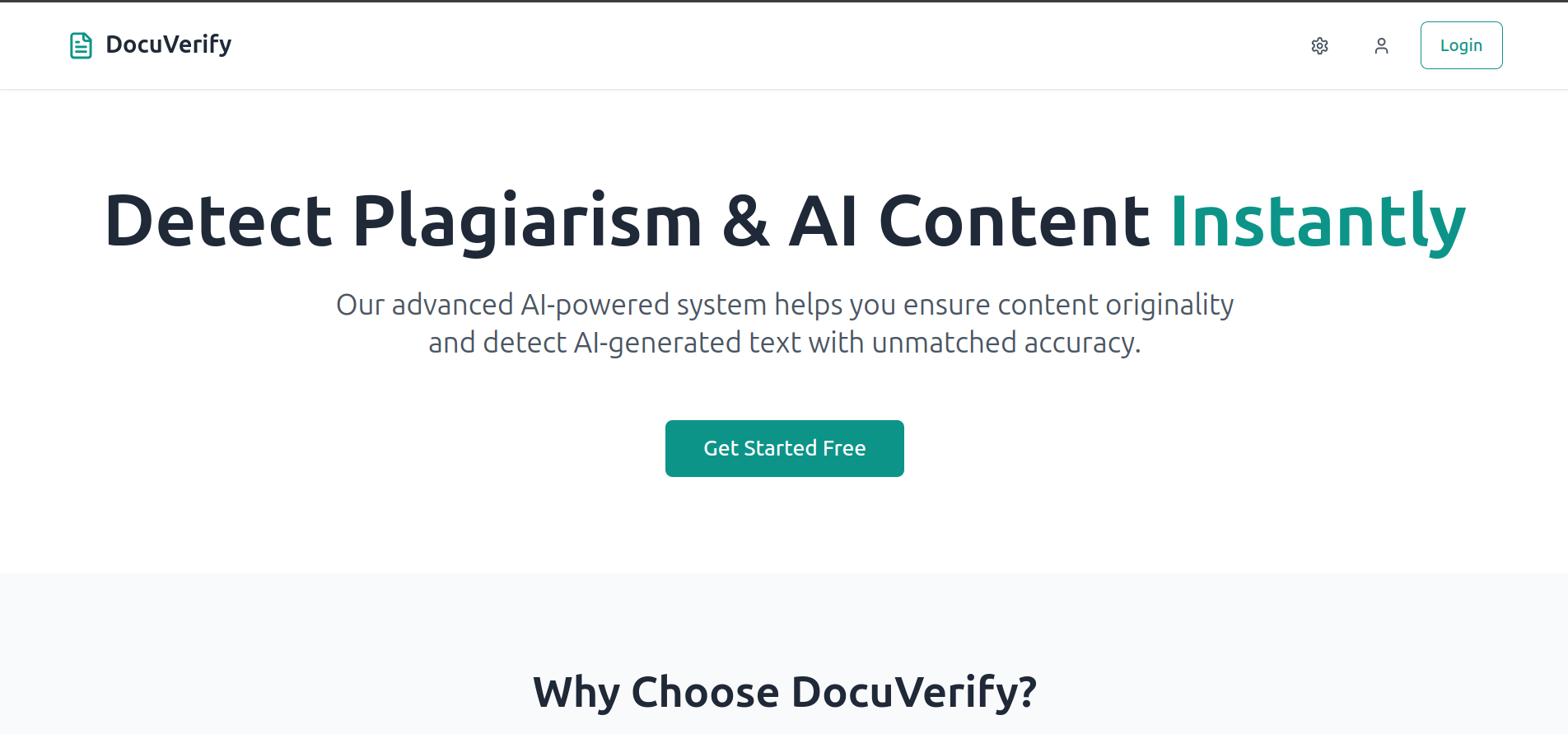
#### **4.3 Interface Design**

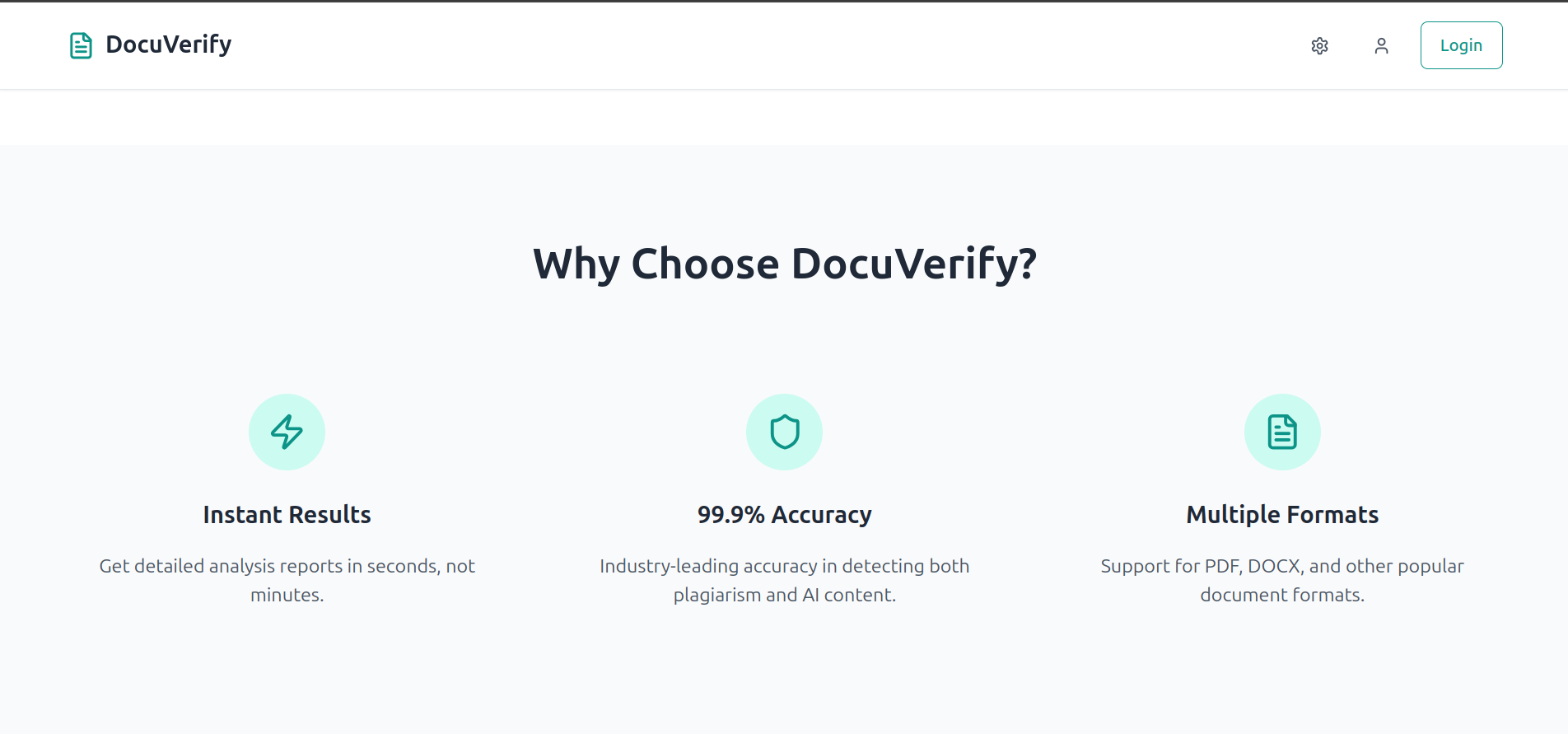
The user interface is set up to allow for two separate dashboards: one for supervisors and one for students. Supervisors have privileges to upload or download submissions, run detection processes, and view results. Students are allowed to submit projects and view feedback. Wireframes and mockups were created using Figma to guide development.



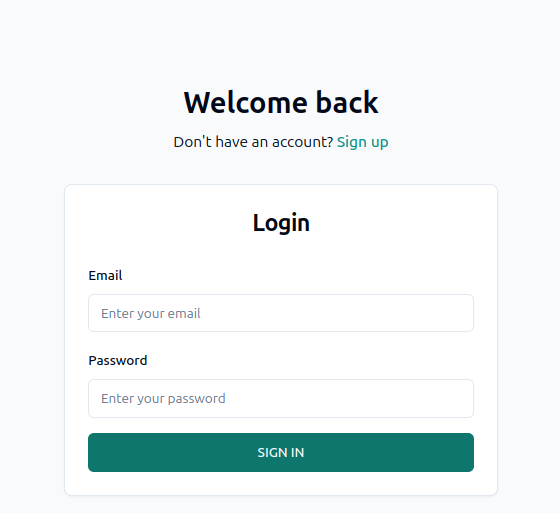
#### 4.3.1 Landing Page

The main page of the DocuVerify, a project supervision tool which everyone sees when they visit the platform.

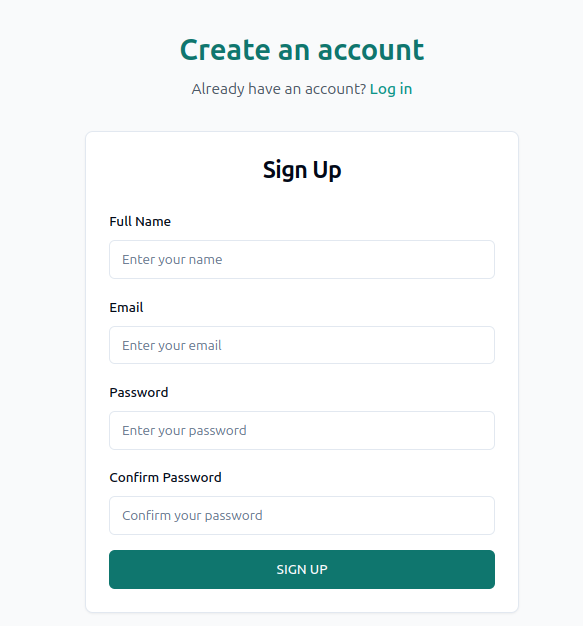




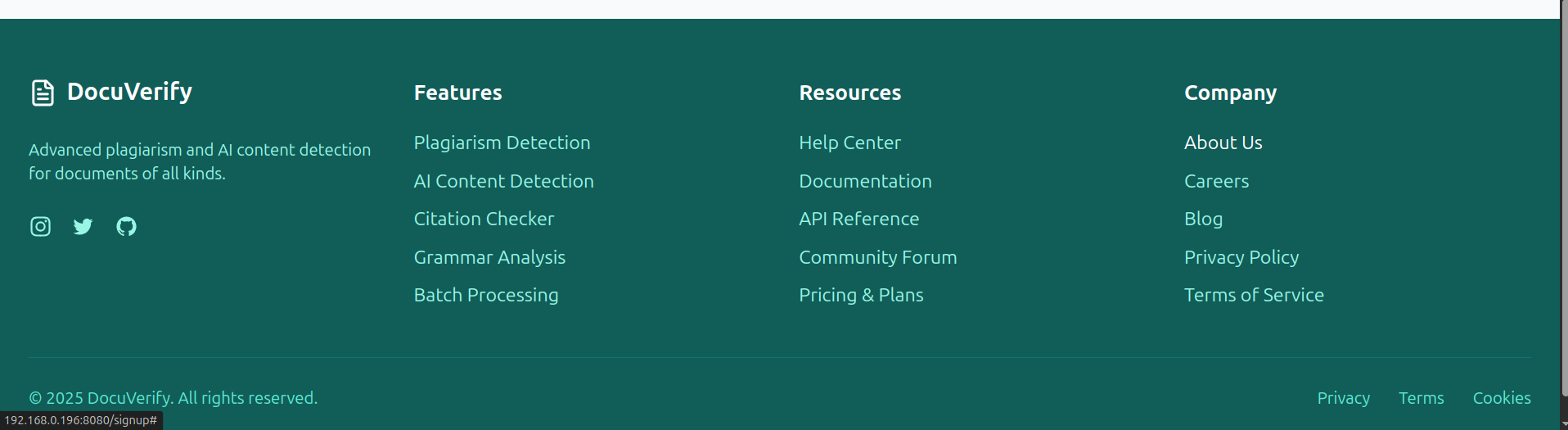
#### 4.3.2 Login page



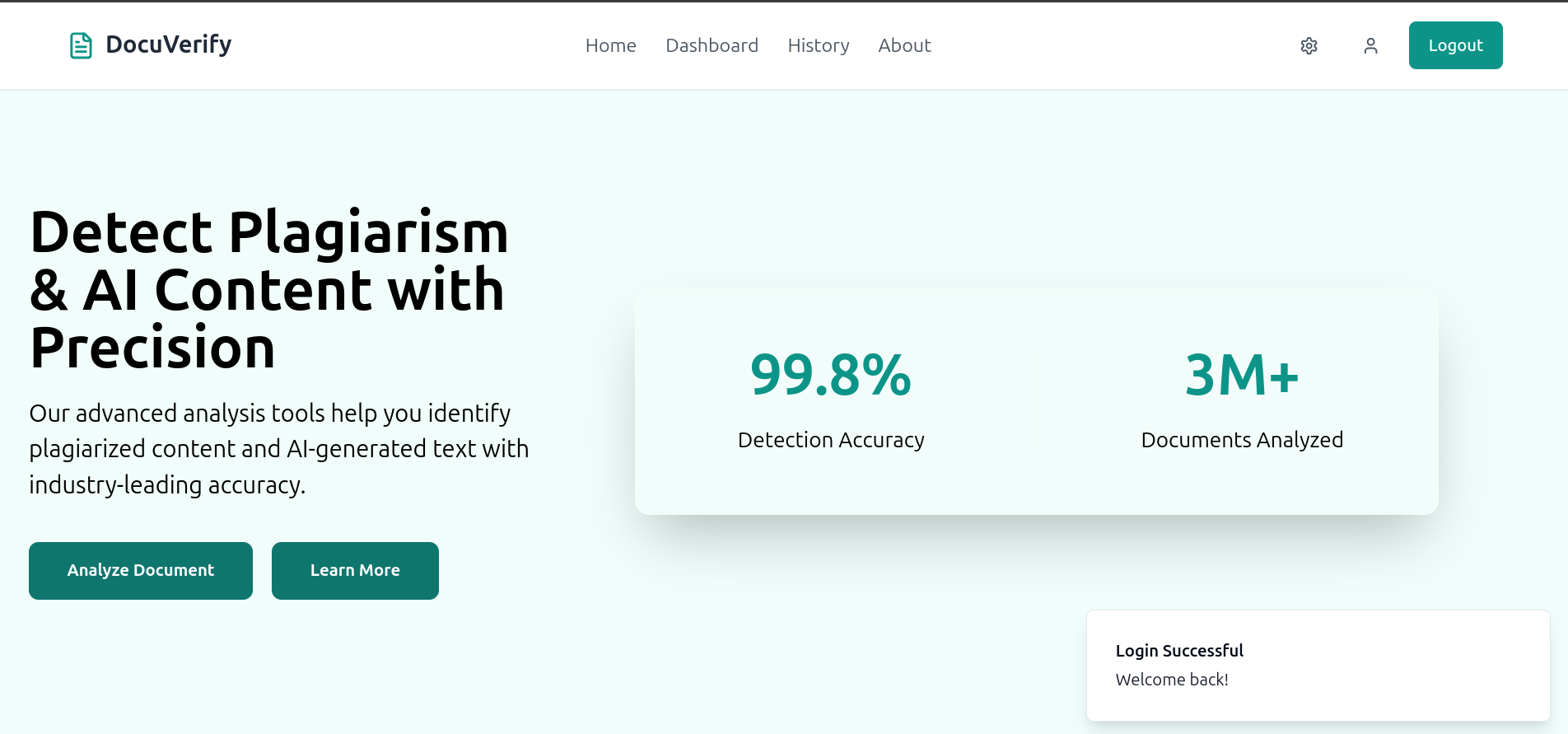
#### 4.3.3 Signup page

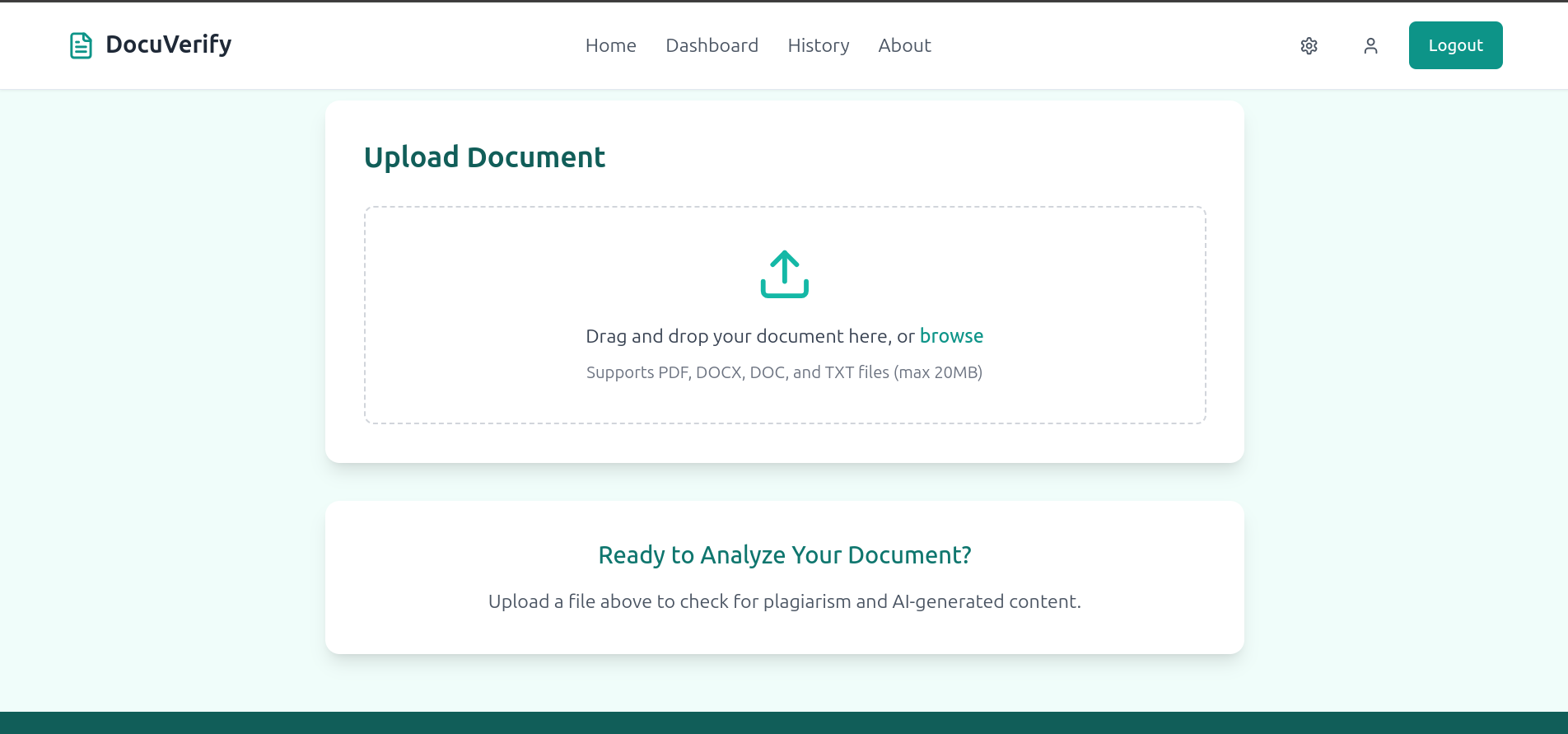


#### 4.3.4 Footer

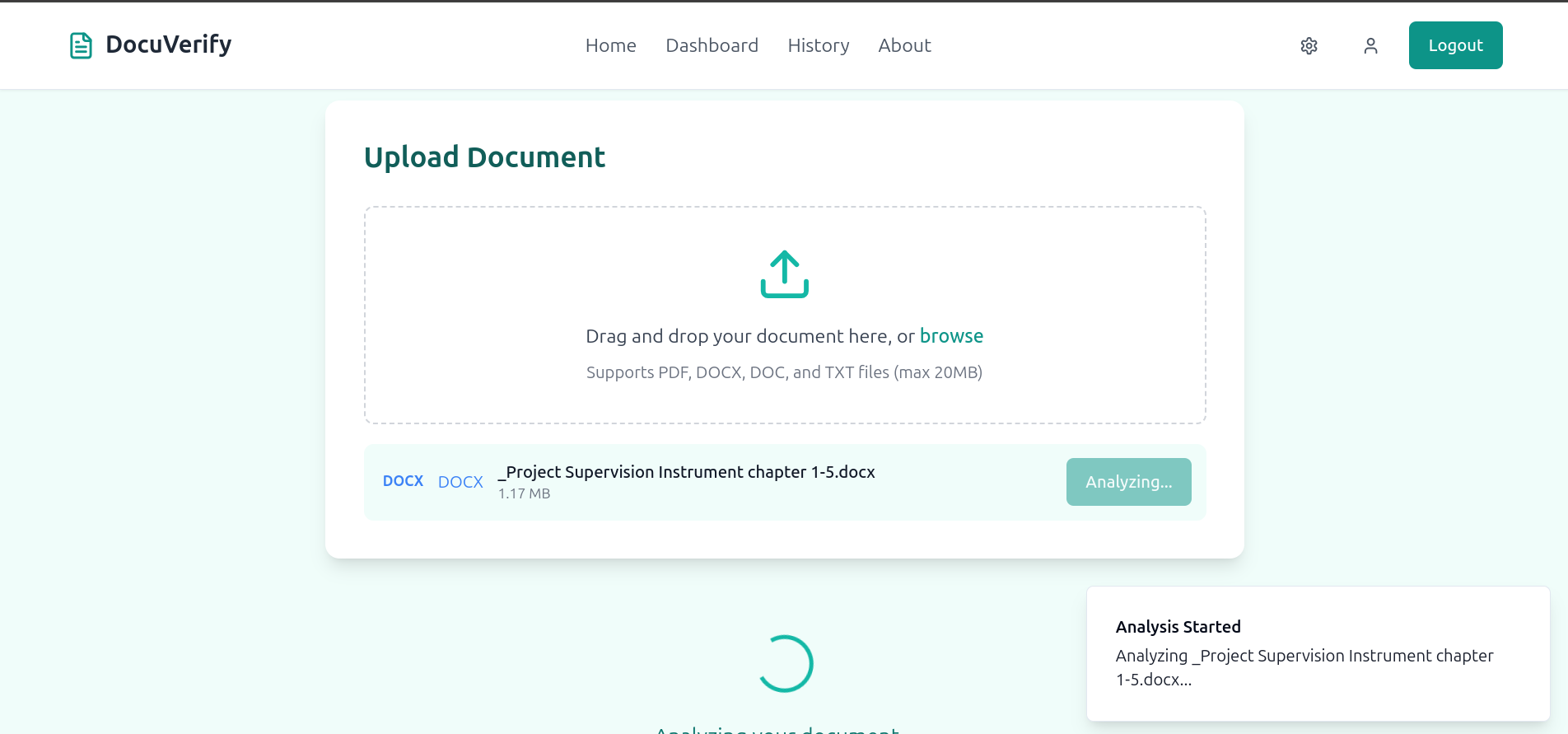


#### 4.3.5 supervisor Dashboard

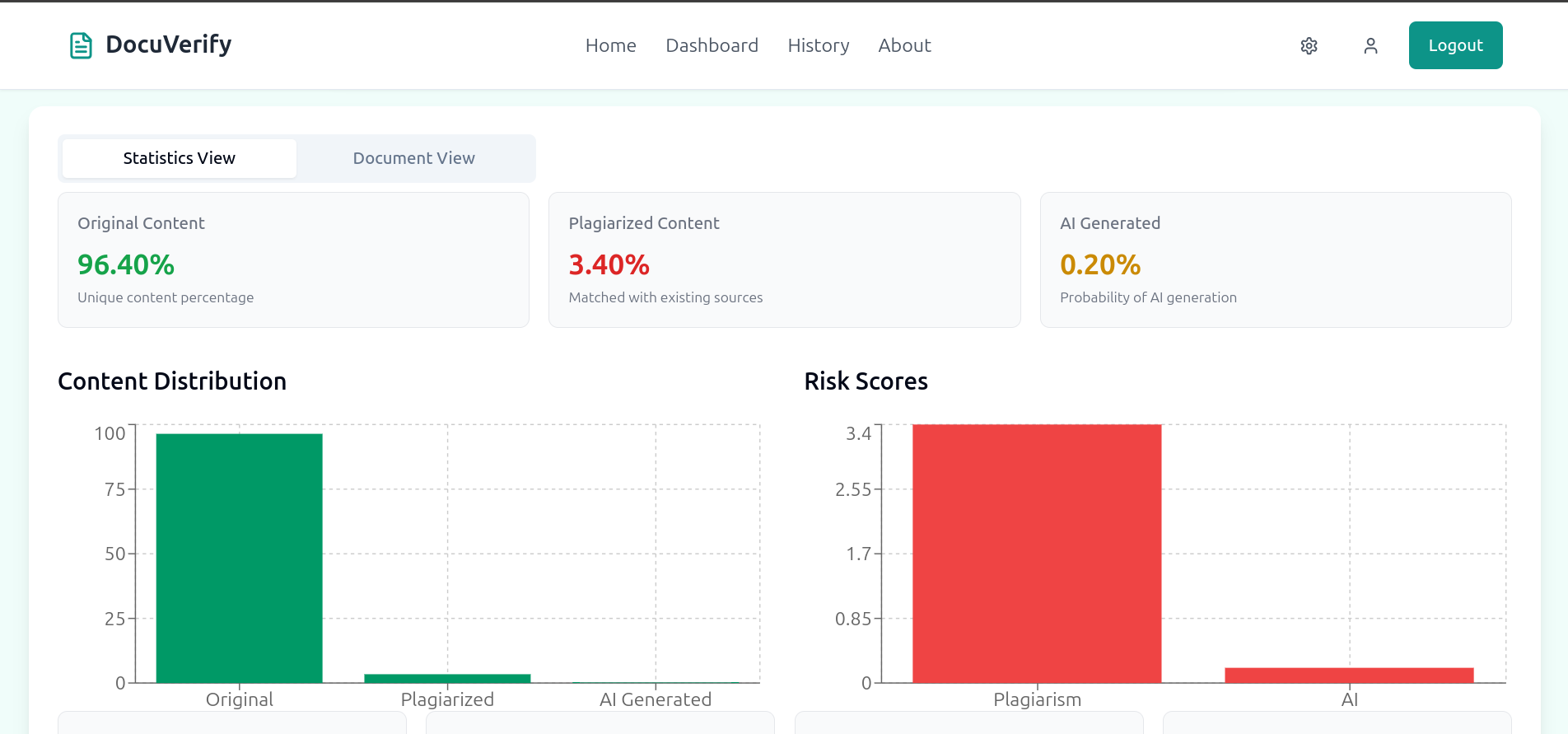




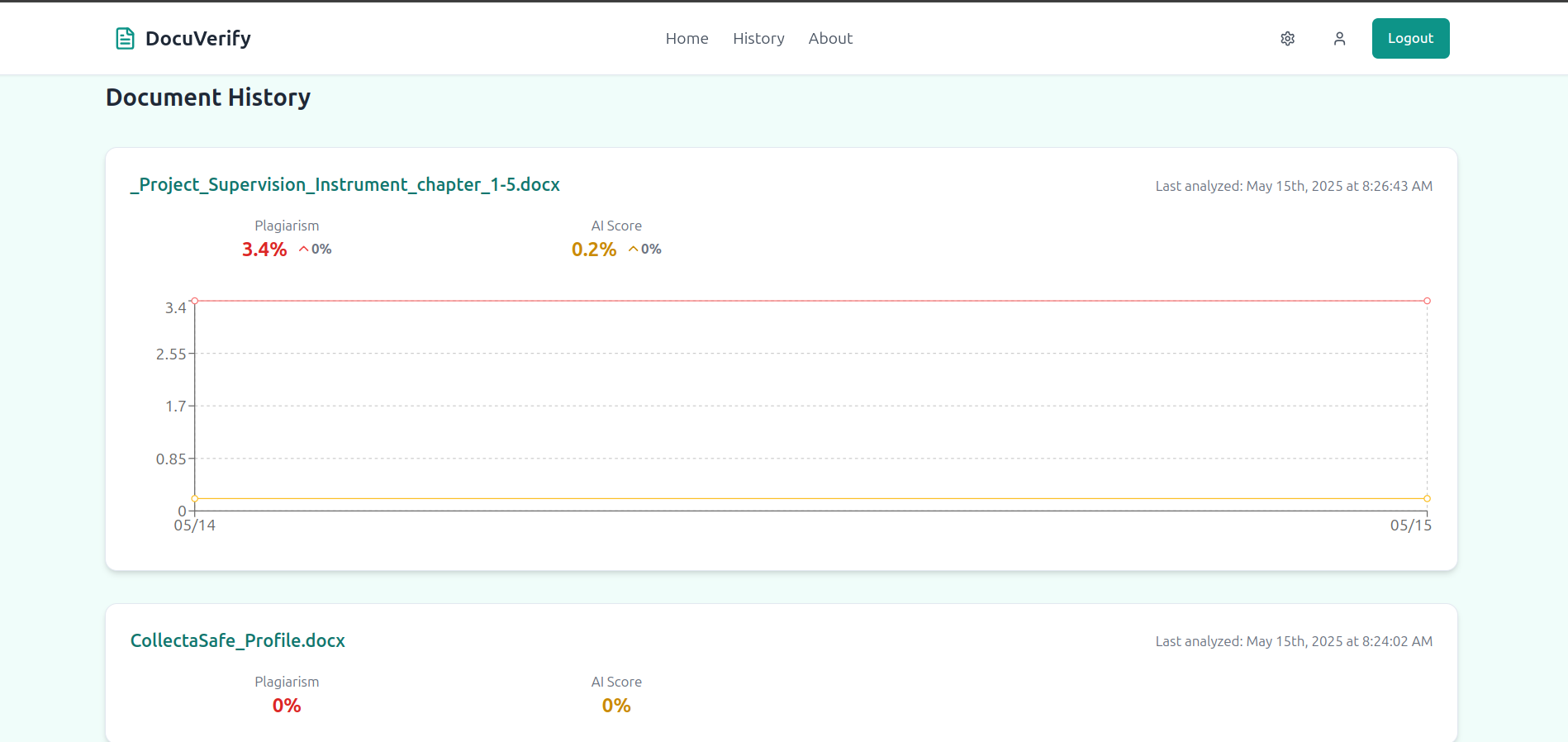
#### 3.4.6 Document Upload



#### **3.4.6 Results section**



#### 3.4.7 History Page



#### **4.4 Machine Learning Model Integration**

The ML component involves two key models: one for plagiarism detection and another for identifying AI-generated content. These models were trained using datasets of genuine and AI-generated academic writing. The models return confidence scores, which are displayed in the system interface alongside flagged sections of text.

#### **4.5 System Flow Diagrams**

Data flow diagrams (DFDs) and UML diagrams were developed to model the interaction between components. The system flow begins with a user logging in and submitting a document, the backend processing the file by running ML analyses, storing the results and rendering the detection report to the user.

DFD finds its way into the major sub-processes within the protective context bubble. Meaningful to the system, these consist of the following phases.

* **Submit Project**: A student submits a project file through the React frontend. The data flow then goes into the backend process of storing the project into PostgreSQL.
* **Run Detection**: The backend invokes ML detection models against the stored content. The ML process analyzes the textual content and derives results (like plagiarism or content scores). These results, as data flows, are impactfully deployed into subsequent processes.
* **Generate Report**: The backend utilizes the detection results to generate a report (credible examples are PDF or dashboard) summarizing what the ML says. It then stores (and/or emails) the report, after which the system either sends the report to the student or notifies the supervisor.

#### Data Flow Diagram



#### Use Case Diagram

In this use case diagram, system interactions (functional requirements) with various actors are depicted. Some important use cases include the following:

* Submit Project (Actor: Student) – the student uploads/submits their project for detection using a Web-based UI.
* Run Detection (Actor: System) – detection is rarely invoked manually by a supervisor, mostly automated when submitting a project.
* View/Review Report (Actors: Student, Supervisor) – after detection, students and supervisors can see the generated detection report via the system.

These use cases are related to the actors (Student or Supervisor). Thus, associations exist between the Student actor and "Submit Project" and "View Report," while the Supervisor actor associates to "View/Review Report" (and probably "Initiate Detection" in the manual case). This use case diagram abstracts the GUI and API calls into high-level functions, consistent with UML: it "illustrates the interactions between users (actors) and a system" showing how various users interact with system functionalit. This way, all roles and goals (submit work, get feedback) can be captured.



#### Activity Diagram (Detection Workflow)

The activity diagram models the sequence of the actions in the project detection workflow. For example, the flow of events may be simply:

* **Start** (Initial Node)
* **Submit Project** – the student uploads the project file. The system validates the submission.
* **Run ML Detection** – the backend calls the ML model to analyze the content.
* **Generate Report** – results are compiled into a report.
* **Deliver Report** – the report is available for the student (and the supervisor notified).
* **End** (Final Node)

The diagram shows a start node (filled circle), followed by a series of action boxes with rounded corners, each describing an action, with control-flow arrows joining them and finally terminating in an end node (bullseye symbol). Should decision points arise in the flow (if the content is too short, for instance, or fails validation), the flow is split by a decision node (diamond). Activity diagrams "are flowcharts that show activities performed by a system". They give details of the logic behind the detection process-from submission of the project to invoking the detection algorithms to outputting the report.



#### Class Diagram (UML Classes)

This UML class diagram gives a representation of static data structures. Major classes are:

* **User** – with attributes like userID, name, email, role, etc., and operations such as submitProject() and viewReport().
* **Project** – with attributes such as projectID, title, content, submissionDate, status and runDetection() as an operation. The Project objects are associated with the User who created the project submission.
* **DetectionResult** – with attributes such as resultID, score, issues, timestamp, and generateReport() as an operation. The DetectionResult is associated with a Project one-to-one (through projectID).
* **Report** – reportID, content, generatedDate attributes, and an operation called view(). Joining from a DetectionResult (linked by resultID) to a Report is through generateReport().

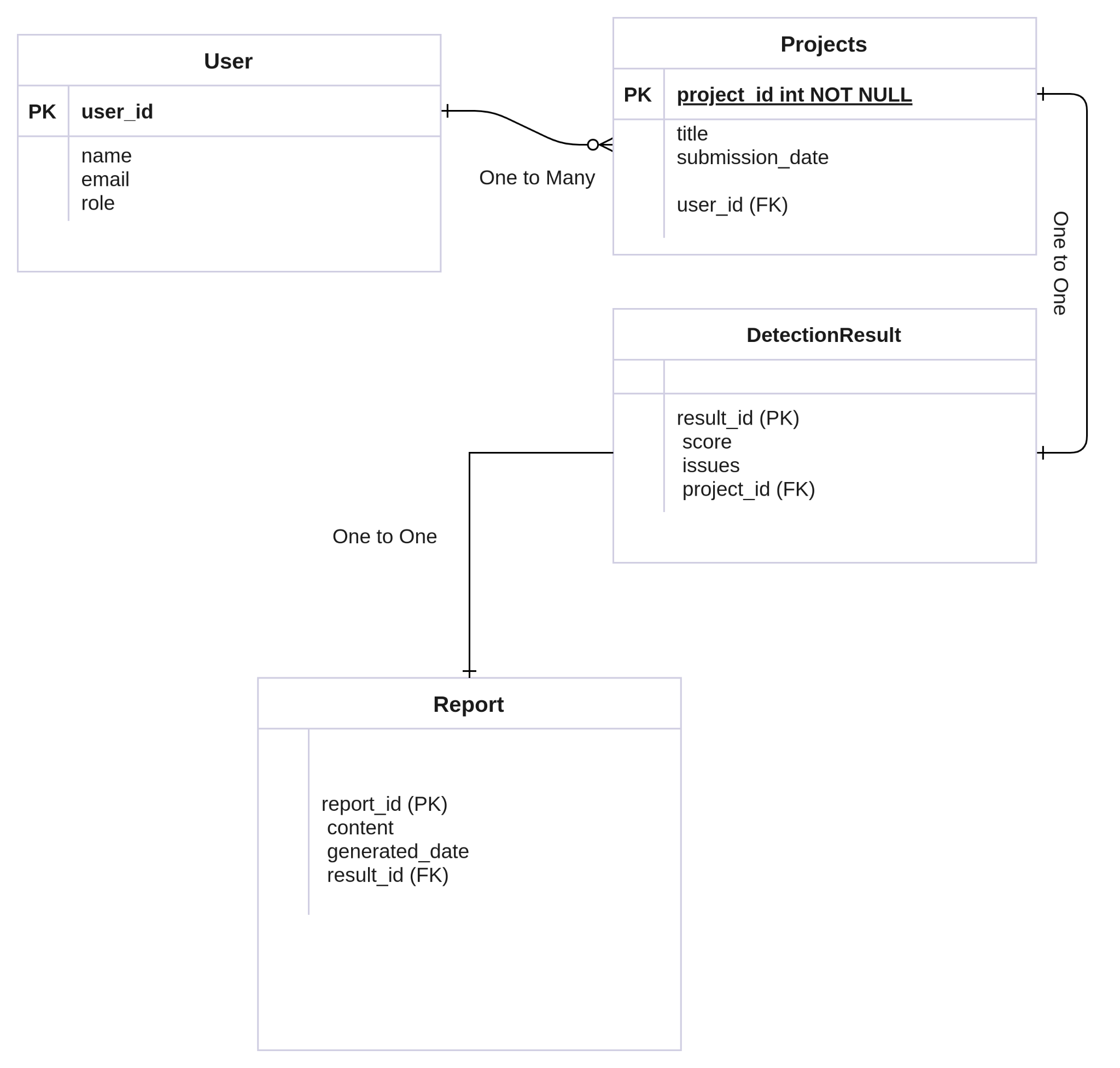
**Associations**: a User (1) may submit many Projects (1→\*); a Project will have one DetectionResult; one Report shall be generated from each DetectionResult. Other classes such as Supervisor could also apply as a specialization of User, or User.role could indicate it. Class diagrams "represent the system's classes, attributes, methods, and relationships, providing a clear view of its architecture." Here, classes are drawn in boxes containing three compartments (name, attributes, methods). Lines between boxes represent associations with multiplicities. This model is the blueprint for implementation; for instance, Django models (ORM classes) and the Python classes will represent these entities to maintain consistency between design and code.



#### Entity-Relationship Diagram (ERD)

The ERD presents a PostgreSQL database schema. The core entities/tables are User, Project, DetectionResult, and Report. Inside each entity box are listed primary keys and key attributes; foreign keys indicate relationships. For instance: User(user\_id PK, name, email, role), Project(project\_id PK, title, submission\_date, status, user\_id FK→User), DetectionResult(result\_id PK, issues, score, project\_id FK→Project), Report(report\_id PK, content, generated\_date, result\_id FK→DetectionResult).

Relationship lines: User→Project (1-to-many), Project→DetectionResult (1-to-1), DetectionResult→Report (1-to-1). An ERD is a "structural diagram" showing major entities in system scope and their inter-relationships. Such ER modeling ensures the database is normalized and follows application logic: For instance, it mandates that each project record must be linked to its submitter and that each report must be linked to its source detection result. Front-end terminologies make the data storage and association clear; hence, ERD design is done before actually writing SQL statements or ORM code for PostgreSQL.



#### **4.6 Security Considerations**

Authentication is implemented using Django’s built-in authentication system with role-based access control.The system enforces input validation and logs all activity for audit purposes.

#### **4.7 Summary**

This chapter described the architecture and design decisions of the Project Supervision Tool. The next chapter covers the coding and testing phase providing insight into implementation details and verification strategies.

# Chapter 5: Coding and Testing

#### **5.1 Technologies Used for Implementation**

The Project Supervision Tool is implemented as a modern-full-stack web application. Using React.js to build the frontend lets users interact with the UI to upload projects and view reports. React components do API calls (via fetch/Axios) to the backend REST endpoints. The backend is developed in Python using Django Rest Framework and exposes APIs for project submission, detection and report retrieval. Django ORM is used to interface with a PostgreSQL database for persistence and the actual detection for the content of the machine could be implemented using Python ML libraries (such as scikit-learn or pytorch) which would be invoked by the Django server whenever a project submission is received. In short, the stack includes React (JSX, HTML/CSS) for the UI, Django/Python as the REST API, and logic layer with PostgreSQL for storage and Python ML frameworks for analysis. These choices are in accordance with a standard approach to scalable web applications, because React and Django are well-known choices for reliable frontend/backend development. The codebase follows MVC/MVT patterns: Django models almost map to the classes identified above, while React state flows articulate the use cases. Git is used for version control, whereas Docker is employed to containerize the services for consistent development environments.

### **Server Side (Dajngo)**

#### 5.1.0 Accounts app

#### 5.1.1 Django Rest Framework in authentication

APIView is the base class for building DRF views.

**Response** - used to create structured HTTP responses.

**status** - HTTP status codes, such as 200, 400, 401, etc.

**RefreshToken** - to generate JWT tokens for user authentication.

**authenticate** - to check whether the given credentials are valid.

UserRegistrationSerializer, UserLoginSerializer are all custom serializers to validate user input for

from rest\_framework import status

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework\_simplejwt.tokens import RefreshToken

from .serializers import UserRegistrationSerializer, UserLoginSerializer

from django.contrib.auth import authenticate

class BaseView(APIView):

authentication\_classes = []

permission\_classes = []

class RegistrationView(BaseView):

def post(self, request):

serializer = UserRegistrationSerializer(data=request.data)

if serializer.is\_valid():

#check is user already exists

if serializer.is\_email\_exists() or serializer.is\_username\_exists():

return Response({'message': 'User already exist. Please verify email.'

}, status=status.HTTP\_400\_BAD\_REQUEST)

user = serializer.save()

# Send verification email here (we'll implement later)

return Response({

'message': 'User registered successfully. Please verify email.'

}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class LoginView(BaseView):

def post(self, request):

serializer = UserLoginSerializer(data=request.data)

if serializer.is\_valid():

user = authenticate(

username=serializer.validated\_data['email'],

password=serializer.validated\_data['password']

)

if user:

refresh = RefreshToken.for\_user(user)

return Response({

'refresh': str(refresh),

'access': str(refresh.access\_token),

})

return Response({'error': 'Invalid credentials'}, status=401)

return Response(serializer.errors, status=400)

#### 

#### 5.1.2 Authentication serializers

This code snippet comprises two serializers written using the Django REST Framework, meant to accommodate CustomUser registration and login processes.

The UserRegistrationSerializer inherits from ModelSerializer and handles the validation and creation of new users. The password field is marked as write-only, and hence it shall never be transmitted in API responses. The Meta class simply attaches the serializer to a model here CustomUser and lists the serializer fields as email, password, first\_name, and last\_name. The create method uses Django's create\_user which creates an actual user in the backend essentially hashing the password and setting the username to be the email. Helper methods like is\_email\_exists() and is\_username\_exists() helps in prevent the creation of multiple accounts with the same details by returning the boolean True if a user with that email/username already exists in the database.

UserLoginSerializer unlike the ModelSerializer is a barebones Serializer class which checks only for the existence of an email-password combination. This serializer is used to validate the credentials for logging in and does not create anything in the databaseand it ensures that only the minimum required fields for login are handled keeping the process of authentication clean and focused

from rest\_framework import serializers

from .models import CustomUser

class UserRegistrationSerializer(serializers.ModelSerializer):

password = serializers.CharField(write\_only=True)

class Meta:

model = CustomUser

fields = ['email', 'password', 'first\_name', 'last\_name']

extra\_kwargs = {'password': {'write\_only': True}}

def create(self, validated\_data):

user = CustomUser.objects.create\_user(

email=validated\_data['email'],

username=validated\_data['email'],

password=validated\_data['password'],

first\_name=validated\_data.get('first\_name', ''),

last\_name=validated\_data.get('last\_name', '')

)

return user

def is\_email\_exists(self):

email = self.validated\_data.get('email')

return CustomUser.objects.filter(email=email).exists()

def is\_username\_exists(self):

username = self.validated\_data.get('username')

return CustomUser.objects.filter(username=username).exists()

class UserLoginSerializer(serializers.Serializer):

email = serializers.EmailField()

password = serializers.CharField()

#### 5.1.3 User Model

Models used - Django's database modeling tools.

AbstractUser is a Django base class encompassing all default user fields like username, email, password which can be extended by eheriting from the AbstractUser class.

from django.db import models

# Create your models here.

from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):

is\_verified = models.BooleanField(default=False)

avatar = models.ImageField(upload\_to='avatars/', null=True, blank=True)

institution = models.CharField(max\_length=100, blank=True)

def \_\_str\_\_(self):

return self.email

#### 5.1.4 documents app

Documents model

The Document model in documents/models.py is for structuring the uploaded documents and their analysis results in a Django application. It relates to the user who uploaded it through a ForeignKey with a custom user model returned via get\_user\_model(). The document stores its raw content in a TextField called content, whereas for the analysis results, it has two float fields: plagiarism\_score and ai\_score, representing percentages or probabilities of detection.

The private \_highlights field stores in the database as a JSONField highlights of certain portions of the document that were flagged for plagiarism or AI-generation. This field can then be accessed via a @property method highlights for clean-controlled access to the highlight data. A content\_hash which is a 64-character string, uniquely identifies each document to ensure that duplicate content is never stored twice.

The uploaded file gets saved using FileField with files being deposited in a documents/ directory. Metadata for the document such as the creation timestamp (created\_at), word count (word\_count), character count (character\_count) and page count (page\_count) is also stored. The document model keeps track of the reading time in minutes as well through reading\_time. This all-encompassing model guarantees that the document itself, together with all of its associated analytic data, is permanently stored and can quickly be retrieved for further analysis.

# documents/models.py

from django.db import models

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

class Document(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

content = models.TextField()

plagiarism\_score = models.FloatField()

ai\_score = models.FloatField()

\_highlights = models.JSONField(default=list)

content\_hash= models.CharField(max\_length=64, unique=True)

file = models.FileField(upload\_to='documents/')

created\_at = models.DateTimeField(auto\_now\_add=True)

word\_count = models.IntegerField()

character\_count = models.IntegerField()

page\_count = models.IntegerField()

reading\_time = models.IntegerField()

@property

def highlights(self):

return self.\_highlights

#### 5.1.5 Documents serializer:

The DocumentStatsSerializer is a somewhat simple serializer which simply returns document-level metrics such as word count, character count, page count, and reading time. Each of these fields takes its value from the Document model's respective attribute.

AIMarkerSerializer takes care of AI detection marker information inside a document. It includes a type (e.g., "AI" or "human"), a confidence score describing how sure the system is about its classification, and a list of sections (probably chunks of the text deemed AI generated or not).

Similarly, SourceMatchSerializer structures information about material found to match other sources, such as potential plagiarism. This includes source name, a url pointing to the original content, match\_percentage, and a list of snippets showing the matched text.

TextAnalysisSerializer provides a summary of percentages of contents being original, plagiarized, or AI-generated, returning the values as floating-point numbers.

The main DocumentSerializer is a ModelSerializer for the Document model and includes all fields (fields = '\_\_all\_\_'). In addition, the serializer defines two additional computed fields by means of SerializerMethodField: fileUrl, which returns the full URL to access the uploaded file using the request context; highlights, which accesses the \_highlights JSON data from the model (via a property). As a helper method, get\_format extracts the file extension to determine the document.

# documents/serializers.py

from rest\_framework import serializers

from .models import Document

from django.urls import reverse

class DocumentStatsSerializer(serializers.Serializer):

word\_count = serializers.IntegerField(source='word\_count')

character\_count = serializers.IntegerField(source='character\_count')

page\_count = serializers.IntegerField(source='page\_count')

reading\_time = serializers.IntegerField(source='reading\_time')

class AIMarkerSerializer(serializers.Serializer):

type = serializers.CharField()

confidence = serializers.FloatField()

sections = serializers.ListField(child=serializers.CharField())

class SourceMatchSerializer(serializers.Serializer):

source = serializers.CharField()

url = serializers.URLField()

match\_percentage = serializers.FloatField()

snippets = serializers.ListField(child=serializers.CharField())

class TextAnalysisSerializer(serializers.Serializer):

original\_content = serializers.FloatField()

plagiarized\_content = serializers.FloatField()

ai\_generated\_content = serializers.FloatField()

class DocumentSerializer(serializers.ModelSerializer):

fileUrl = serializers.SerializerMethodField()

highlights = serializers.SerializerMethodField()

class Meta:

model = Document

fields = '\_\_all\_\_'

def get\_file\_url(self, obj):

request = self.context.get('request')

if obj.file:

return request.build\_absolute\_uri(obj.file.url)

return None

def get\_format(self, obj):

if obj.file:

return obj.file.name.split('.')[-1].lower()

return 'unknown'

#### 

#### 5.1.6 Utility functions

This module deals with parsing and analyzing documents in a Django environment. It first extracts text from files in PDF, DOCX, or TXT formats, utilizing libraries such as `PyPDF2` and `python-docx`, with hefty error-handling to take care of usual problems like unreadable or image-based PDFs. After text extraction, several document statistics are calculated such as the number of words, number of characters, estimated number of pages and reading time.

For content verification, two types of analyses are performed and the first is plagiarism detection which compares the text in the uploaded document against all other documents in the system and from other sources like the web using a TF-IDF vectorizer with 5-character n-grams and cosine similarity. The windows slide through the text, and crucially-matching sections get flagged. The flagged passages in the documents are highlighted and used for computing the overall plagiarism score.

The second analysis concerns AI content detection. The text is chunked into 512-character chunks and then scored by a Hugging Face transformer model, giving the likelihood that each chunk was written by a human or generated by AI. The scores are averaged, and the AI-like portions of the text are highlighted. The final score of the AI is limited so that the combined percentages of AI and plagiarism do not exceed 100%.

import PyPDF2

import docx

import re

import textstat

import torch

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from .models import Document

import logging

from transformers import pipeline

logger = logging.getLogger(\_\_name\_\_)

def extract\_text\_from\_file(file):

"""Extract text with better error handling."""

text = ""

logger.info(f"Starting extraction for {file.name}")

if file.name.lower().endswith('.pdf'):

try:

file.seek(0)

reader = PyPDF2.PdfReader(file)

if not reader.pages:

raise ValueError("PDF has no readable pages")

for i, page in enumerate(reader.pages[:20]):

chunk = page.extract\_text() or ''

text += chunk + "\n"

if i >= 3 and len(text) < 100:

raise ValueError("PDF looks image-based")

if len(text.strip()) < 100:

raise ValueError("PDF contains insufficient text")

except Exception as e:

logger.error(f"PDF extraction error: {e}")

raise ValueError(f"Failed to extract PDF text: {e}")

elif file.name.lower().endswith('.docx'):

try:

doc = docx.Document(file)

text = "\n".join(p.text for p in doc.paragraphs)

except Exception as e:

logger.error(f"DOCX extraction error: {e}")

raise ValueError(f"Failed to extract DOCX text: {e}")

elif file.name.lower().endswith('.txt'):

text = file.read().decode('utf-8', errors='ignore')

else:

raise ValueError("Unsupported format. Only PDF, DOCX, TXT allowed.")

logger.info(f"Extracted {len(text)} characters")

return text.strip()

def analyze\_text(content\_hash, text):

"""

Plagiarism detection via character 5-gram sliding windows

against all other docs (excluding the one with this hash).

"""

# fetch all other documents' contents

others = list(

Document.objects

.exclude(content\_hash=content\_hash)

.values\_list('content', flat=True)

)

if not others:

return {'score': 0.0, 'highlights': []}

# vectorize full text + others

vec = TfidfVectorizer(analyzer='char', ngram\_range=(5, 5))

corpus = [text] + others

mat = vec.fit\_transform(corpus)

base\_vec = mat[0]

window = 200

step = 100

total = len(text)

matched = set()

highlights = []

for start in range(0, total - window + 1, step):

snippet = text[start:start + window]

sim = cosine\_similarity(vec.transform([snippet]), mat[1:])[0]

# find any doc with similarity > threshold

if sim.max() > 0.3:

end = start + window

# mark chars

matched.update(range(start, end))

highlights.append({

'type': 'plagiarism',

'position': calculate\_position(text, start, end)

})

score = round(len(matched) / total \* 100, 1) if total else 0.0

return {

'score': min(score, 100.0),

'highlights': highlights

}

def check\_ai\_probability(text, plagiarism\_highlights=None, plagiarism\_score=0):

"""

AI detection: simple chunking, no overlap.

"""

plagiarism\_score = plagiarism\_score or 0

if len(text) < 300:

return {'score': 0.0, 'highlights': []}

# initialize once

if not hasattr(check\_ai\_probability, 'detector'):

check\_ai\_probability.detector = pipeline(

'text-classification',

model='Hello-SimpleAI/chatgpt-detector-roberta',

truncation=True,

max\_length=512,

device= 0 if torch.cuda.is\_available() else -1

)

chunk\_size = 512

preds = []

for i in range(0, len(text), chunk\_size):

chunk = text[i:i+chunk\_size]

if len(chunk) < 100:

continue

preds.append((i, check\_ai\_probability.detector(chunk)[0]))

scores = []

highlights = []

for idx, pred in preds:

lbl = pred['label']

sc = pred['score'] \* 100

# if label is AI, we take sc; if HUMAN, we take (100 - sc)

val = sc if lbl == 'AI' else (100 - sc)

scores.append(val)

if lbl == 'AI':

highlights.append({

'type': 'ai',

'position': calculate\_position(text, idx, idx + chunk\_size)

})

avg = round(sum(scores) / len(scores), 1) if scores else 0.0

# caping so that plagiarism + ai ≤ 100

cap = max(0.0, 100.0 - plagiarism\_score)

return {

'score': min(avg, cap),

'highlights': highlights

}

def calculate\_position(full\_text, start, end):

"""Return percentage-based box for front-end."""

total = len(full\_text)

return {

'page': 1,

'x': round(start / total \* 100, 2),

'y': 0, # not used

'width': round((end - start) / total \* 100, 2),

'height': 2

}

def calculate\_document\_stats(text):

"""Word count, char count, pages, reading time (mins)."""

words = len(text.split())

chars = len(text)

pages = max(1, (chars // 1500) + 1)

try:

read = textstat.reading\_time(text, ms\_per\_char=14.69)

except Exception:

read = max(1, words // 200)

return {

'word\_count': words,

'character\_count': chars,

'page\_count': pages,

'reading\_time': read

}

#### 5.1.7 Documents viesets

We define two view classes here to handle document analysis and management on a Django REST Framework setup. `AnalyzeDocumentView` is an API view where authenticated users upload a document for analysis. It checks for file type and size before actually extracting text from the document. It also performs basic document validity checks, i.e., is the document too short? Then, a unique hash is generated for the document content that allows the application to find if the document has already been uploaded before. The basic document analysis includes scoring for plagiarism and AI detection. These two scores are then weighted for the originality score. Some statistics are also generated for the document (word count, character count, etc.) with all the new info used to update the existing document if it's already in the database or, if not, to create a new document. The response is then given to the user, including scores, stats, and highlighting of suspicious plagiarized or AI content.

On the other hand, `DocumentViewSet` is a generic API view allowing management of all CRUD operations on the `Document` model. It overrides the creation of new documents so that file contents would be treated similarly to those in the analyze view. If there is a document with the same content hash inside, it updates those plagiarism and AI scores. The view also supports exempting CSRF for certain chosen actions (e.g., `test-csrf`), which may come handy for testing purposes or specific less sensitive endpoints.

import PyPDF2

import docx

import re

import textstat

import torch

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from .models import Document

import logging

from transformers import pipeline

logger = logging.getLogger(\_\_name\_\_)

def extract\_text\_from\_file(file):

"""Extract text with better error handling."""

text = ""

logger.info(f"Starting extraction for {file.name}")

if file.name.lower().endswith('.pdf'):

try:

file.seek(0)

reader = PyPDF2.PdfReader(file)

if not reader.pages:

raise ValueError("PDF has no readable pages")

for i, page in enumerate(reader.pages[:20]):

chunk = page.extract\_text() or ''

text += chunk + "\n"

if i >= 3 and len(text) < 100:

raise ValueError("PDF looks image-based")

if len(text.strip()) < 100:

raise ValueError("PDF contains insufficient text")

except Exception as e:

logger.error(f"PDF extraction error: {e}")

raise ValueError(f"Failed to extract PDF text: {e}")

elif file.name.lower().endswith('.docx'):

try:

doc = docx.Document(file)

text = "\n".join(p.text for p in doc.paragraphs)

except Exception as e:

logger.error(f"DOCX extraction error: {e}")

raise ValueError(f"Failed to extract DOCX text: {e}")

elif file.name.lower().endswith('.txt'):

text = file.read().decode('utf-8', errors='ignore')

else:

raise ValueError("Unsupported format. Only PDF, DOCX, TXT allowed.")

logger.info(f"Extracted {len(text)} characters")

return text.strip()

def analyze\_text(content\_hash, text):

"""

Plagiarism detection via character 5-gram sliding windows

against all other docs (excluding the one with this hash).

"""

# fetch all other documents' contents

others = list(

Document.objects

.exclude(content\_hash=content\_hash)

.values\_list('content', flat=True)

)

if not others:

return {'score': 0.0, 'highlights': []}

# vectorize full text + others

vec = TfidfVectorizer(analyzer='char', ngram\_range=(5, 5))

corpus = [text] + others

mat = vec.fit\_transform(corpus)

base\_vec = mat[0]

window = 200

step = 100

total = len(text)

matched = set()

highlights = []

for start in range(0, total - window + 1, step):

snippet = text[start:start + window]

sim = cosine\_similarity(vec.transform([snippet]), mat[1:])[0]

# find any doc with similarity > threshold

if sim.max() > 0.3:

end = start + window

# mark chars

matched.update(range(start, end))

highlights.append({

'type': 'plagiarism',

'position': calculate\_position(text, start, end)

})

score = round(len(matched) / total \* 100, 1) if total else 0.0

return {

'score': min(score, 100.0),

'highlights': highlights

}

def check\_ai\_probability(text, plagiarism\_highlights=None, plagiarism\_score=0):

"""

AI detection: simple chunking, no overlap.

"""

plagiarism\_score = plagiarism\_score or 0

if len(text) < 300:

return {'score': 0.0, 'highlights': []}

# initialize once

if not hasattr(check\_ai\_probability, 'detector'):

check\_ai\_probability.detector = pipeline(

'text-classification',

model='Hello-SimpleAI/chatgpt-detector-roberta',

truncation=True,

max\_length=512,

device= 0 if torch.cuda.is\_available() else -1

)

chunk\_size = 512

preds = []

for i in range(0, len(text), chunk\_size):

chunk = text[i:i+chunk\_size]

if len(chunk) < 100:

continue

preds.append((i, check\_ai\_probability.detector(chunk)[0]))

scores = []

highlights = []

for idx, pred in preds:

lbl = pred['label']

sc = pred['score'] \* 100

# if label is AI, we take sc; if HUMAN, we take (100 - sc)

val = sc if lbl == 'AI' else (100 - sc)

scores.append(val)

if lbl == 'AI':

highlights.append({

'type': 'ai',

'position': calculate\_position(text, idx, idx + chunk\_size)

})

avg = round(sum(scores) / len(scores), 1) if scores else 0.0

# caping so that plagiarism + ai ≤ 100

cap = max(0.0, 100.0 - plagiarism\_score)

return {

'score': min(avg, cap),

'highlights': highlights

}

def calculate\_position(full\_text, start, end):

"""Return percentage-based box for front-end."""

total = len(full\_text)

return {

'page': 1,

'x': round(start / total \* 100, 2),

'y': 0, # not used

'width': round((end - start) / total \* 100, 2),

'height': 2

}

def calculate\_document\_stats(text):

"""Word count, char count, pages, reading time (mins)."""

words = len(text.split())

chars = len(text)

pages = max(1, (chars // 1500) + 1)

try:

read = textstat.reading\_time(text, ms\_per\_char=14.69)

except Exception:

read = max(1, words // 200)

return {

'word\_count': words,

'character\_count': chars,

'page\_count': pages,

'reading\_time': read

}

#### 5.1.8 Document Routes

This URL configuration file routes two major views: one being the DocumentViewSet registered with DefaultRouter provided by Django REST Framework, and the other a simple view, named AnalyzeDocumentView. The DocumentViewSet of REST for document objects (list, retrieve, create, update, and destroy) under the path prefix /documents/. AnalyzeDocumentView fine-analyzes the uploaded documents (plagiarism and AI detectors) and is accessible at the /analyze/ endpoint. Finally, the file includes URLs appearing in the browser of uploaded media during development by way of appending static file URLs from the django.conf.settings module.

# documents/urls.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import AnalyzeDocumentView, DocumentViewSet

from django.conf import settings

from django.conf.urls.static import static

router = DefaultRouter()

router.register(r'documents', DocumentViewSet, basename='document')

urlpatterns = [

path('', include(router.urls)),

path('analyze/', AnalyzeDocumentView.as\_view(), name='analyze-document'),

]+ static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

### **Client Side (**[**React.js**](http://react.js)**)**

#### 5.1.9 The Login page

The Login component generates a complete login page with a navigation bar at the top, a centered login form wedged in between, and the footer at the bottom. The page greets returning users with a headline, while giving a signup link to those who haven't registered yet. The layout is responsive and uses Tailwind CSS classes for spacing, typography, and color theming. The application uses reusable components (Navbar, LoginForm, and Footer) for application-wide consistency and good separation of concerns.

import { Link } from "react-router-dom";

import LoginForm from "@/components/auth/LoginForm";

import Navbar from "@/components/Navbar";

import Footer from "@/components/Footer";

const Login = () => {

return (

<>

<Navbar />

<div className="min-h-screen flex flex-col bg-gray-50">

<main className="flex-grow flex items-center justify-center p-6">

<div className="w-full max-w-md">

<div className="text-center mb-8">

<h1 className="text-3xl font-bold ">Welcome back</h1>

<p className="mt-2 ">

Don't have an account?{" "}

<Link to="/signup" className="text-teal-600 hover:text-teal-700 font-medium">

Sign up

</Link>

</p>

</div>

<LoginForm />

</div>

</main>

</div>

<Footer />

</>

);

};

export default Login;

#### 5.2.1 Signup Page

import { Link } from "react-router-dom";

import SignUpForm from "@/components/auth/SignUpForm";

import Navbar from "@/components/Navbar";

import Footer from "@/components/Footer";

const SignUp = () => {

return (

<>

<Navbar />

<div className="min-h-screen flex flex-col bg-gray-50">

<main className="flex-grow flex items-center justify-center p-6">

<div className="w-full max-w-md">

<div className="text-center mb-8">

<h1 className="text-3xl font-bold text-teal-700">Create an account</h1>

<p className="mt-2 text-gray-600">

Already have an account?{" "}

<Link to="/login" className="text-teal-600 hover:text-teal-700 font-medium">

Log in

</Link>

</p>

</div>

<SignUpForm />

</div>

</main>

</div>

<Footer />

</>

);

};

export default SignUp;

5.2.2 Landing Page

The Landing component creates a welcoming and informative landing page for DocuVerify a service for detecting plagiarism and AI-generated content.

**Hero Section** carries a strong headline inviting the viewer to explore the platform for instantaneous plagiarism and AI detection. A call-to-action button also invites the user to sign up for free.

**Features Section** - A three-column grid is used to highlight the major features and benefits of the service: instant results, 99.9% accuracy, and multi-document format support. Each feature includes an icon (from lucide-react) and a brief description, keeping the information visual and easy to grasp.

The layout is responsive, with Tailwind CSS classes used to make sure the page splits up well on smaller screen sizes. The Navbar component provides continual navigation while the Footer component gives a finishing touch to the design.

import { useNavigate } from "react-router-dom";

import { Button } from "@/components/ui/button";

import { FileText, Shield, Zap } from "lucide-react";

import Footer from "@/components/Footer";

import Navbar from "@/components/Navbar";

const Landing = () => {

const navigate = useNavigate();

return (

<div className="min-h-screen flex flex-col">

<Navbar />

{/\* Hero Section \*/}

<section className="py-20 px-6">

<div className="container mx-auto text-center">

<h1 className="text-4xl md:text-6xl font-bold mb-6 text-gray-800">

Detect Plagiarism & AI Content

<span className="text-teal-600"> Instantly</span>

</h1>

<p className="text-xl md:text-2xl mb-12 text-gray-600 max-w-3xl mx-auto">

Our advanced AI-powered system helps you ensure content originality and detect AI-generated text with unmatched accuracy.

</p>

<Button

className="bg-teal-600 text-white hover:bg-teal-700 text-lg py-6 px-8"

onClick={() => navigate('/signup')}

>

Get Started Free

</Button>

</div>

</section>

{/\* Features Section \*/}

<section className="py-20 bg-gray-50 px-6">

<div className="container mx-auto">

<h2 className="text-3xl md:text-4xl font-bold text-center mb-16 text-gray-800">

Why Choose DocuVerify?

</h2>

<div className="grid md:grid-cols-3 gap-12">

<div className="text-center p-6">

<div className="w-16 h-16 bg-teal-100 rounded-full flex items-center justify-center mx-auto mb-6">

<Zap className="h-8 w-8 text-teal-600" />

</div>

<h3 className="text-xl font-bold mb-4 text-gray-800">Instant Results</h3>

<p className="text-gray-600">Get detailed analysis reports in seconds, not minutes.</p>

</div>

<div className="text-center p-6">

<div className="w-16 h-16 bg-teal-100 rounded-full flex items-center justify-center mx-auto mb-6">

<Shield className="h-8 w-8 text-teal-600" />

</div>

<h3 className="text-xl font-bold mb-4 text-gray-800">99.9% Accuracy</h3>

<p className="text-gray-600">Industry-leading accuracy in detecting both plagiarism and AI content.</p>

</div>

<div className="text-center p-6">

<div className="w-16 h-16 bg-teal-100 rounded-full flex items-center justify-center mx-auto mb-6">

<FileText className="h-8 w-8 text-teal-600" />

</div>

<h3 className="text-xl font-bold mb-4 text-gray-800">Multiple Formats</h3>

<p className="text-gray-600">Support for PDF, DOCX, and other popular document formats.</p>

</div>

</div>

</div>

</section>

{/\* Footer \*/}

<Footer />

</div>

);

};

export default Landing;

#### 5.2.3 Index page (Dashboard)

The Index is the main page of the document analysis service, where users can upload documents to analyze them for plagiarism and AI-generated content while viewing results. When loading, the component first checks whether a user is authenticated by looking for an access\_token in the localStorage. It then seeks to confirm that the backend server is available by sending a request so that the user's session is still active. If the server happens to be down or the session has expired, a toast notification appears and the user is redirected to the login page.

In the event that one is found logged in, the interface provides for document uploads. Users may now upload files for analysis. Uploaded files will trigger the invocation of document analysis via the handleAnalyzeDocument function. This handles the document analysis, during which time a loading spinner is displayed to let the user know that their file is being processed. After the results have finished rendering inside the ResultsPanel component further information about the document and its contents is available for review and inspection.

During an analysis, feedback is provided so the user knows what is going on "Analyzing your document..." and "This may take a few moments." If there has not yet been an analysis, a prompt will encourage the user to upload a file. If an error occurs during the analysis, the UI will provide a message telling the user that the operation failed.

import { useState, useEffect } from 'react';

import { useNavigate } from 'react-router-dom';

import Navbar from '@/components/Navbar';

import Hero from '@/components/Hero';

import DocumentUpload from '@/components/DocumentUpload';

import ResultsPanel from '@/components/ResultsPanel';

import Footer from '@/components/Footer';

import { DocumentAnalysis } from '@/types/analysis';

import { sampleAnalysis } from '@/utils/demoData';

import { useToast } from '@/components/ui/use-toast';

import { analyzeDocument, backendAPI } from '@/lib/api';

import ErrorBoundary from '@/components/ErrorBoundary';

const Index = () => {

const [isAnalyzing, setIsAnalyzing] = useState(false);

const [analysisResult, setAnalysisResult] = useState<DocumentAnalysis | null>(null);

const { toast } = useToast();

const navigate = useNavigate();

useEffect(() => {

const isAuthenticated = localStorage.getItem('access\_token') !== null;

const checkBackendServer = async () => {

try {

const response = await backendAPI({

method: 'GET',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${localStorage.getItem('access\_token')}`

}

}

);

if (response.status !== 200) {

throw new Error('Backend server is down');

}

} catch (error) {

toast({

title: "Session Expired",

description: "Your session has expired. Please log in again.",

variant: "destructive"

});

navigate('/login');

}

};

checkBackendServer();

if (!isAuthenticated) {

navigate('/login');

}

}, [navigate]);

const handleAnalyzeDocument = async (file: File) => {

setIsAnalyzing(true);

try {

toast({

title: "Analysis Started",

description: `Analyzing ${file.name}...`,

});

const result = await analyzeDocument(file);

setAnalysisResult({

...result,

fileName: file.name,

analyzedAt: new Date().toISOString()

});

toast({

title: "Analysis Complete",

description: "Your document has been analyzed successfully.",

});

} catch (error) {

toast({

title: "Analysis Failed",

description: error.response?.data?.error || "Failed to analyze document",

variant: "destructive"

});

} finally {

setIsAnalyzing(false);

}

};

return (

<div className="min-h-screen flex flex-col bg-teal-50">

<Navbar />

<Hero />

<main className="flex-grow py-12">

<div className="container mx-auto px-6 space-y-8">

<div className="max-w-3xl mx-auto">

<DocumentUpload

onAnalyze={handleAnalyzeDocument}

isAnalyzing={isAnalyzing}

/>

</div>

{analysisResult && !isAnalyzing && (

<div className="mt-12 animate-fade-in">

<ErrorBoundary>

<ResultsPanel analysis={analysisResult} />

</ErrorBoundary>

</div>

)}

{isAnalyzing && (

<div className="mt-12 flex justify-center">

<div className="p-8 text-center">

<div className="inline-block h-12 w-12 animate-spin rounded-full border-4 border-teal-500 border-r-transparent mb-4"></div>

<p className="text-teal-700 text-lg">Analyzing your document...</p>

<p className="text-gray-500 mt-2">This may take a few moments</p>

</div>

</div>

)}

{!analysisResult && !isAnalyzing && (

<div className="mt-12 p-8 text-center bg-white rounded-xl shadow-lg max-w-3xl mx-auto">

<h3 className="text-xl font-medium text-teal-700 mb-3">Ready to Analyze Your Document?</h3>

<p className="text-gray-600">Upload a file above to check for plagiarism and AI-generated content.</p>

</div>

)}

</div>

</main>

<Footer />

</div>

);

};

export default Index;

#### **5.2 Development Process**

The entire implementation underwent various phases which include designing database schema and creating Django models for the entities represented on the ER diagram. The API endpoints were developed using Django Rest Framework and tested with Postman and other tools e.g POST /api/projects/documents, /api/analyze-document/. As the API was being prepared, the frontend was scaffolded (e.g., Create React App) with components for submission forms and report dashboards. For the ML pieces, the pre-trained detectors were plugged into the backend: once a project was received, Django views would call the ML module, process the result, store it, and return it. We followed the iterative Agile approach, where features were developed in sprints and code was reviewed through pull requests, ensuring that builds passed with continuous integration. Comments and documented design choices were maintained.

#### **5.3 Testing and Tools**

The program underwent grossly rigorous testing on various levels. Unit testing ran on single unit-level test cases for backend functions and frontend components. For example, we used the built-in Django test framework, which inherited from Python's unittest framework, to test models, methods, and API views. For React components tools including Jest and React Testing Library helped in testing the UI logic. Integration testing ensured the end-to-end functionality where a sequence of acts submitted a project via the UI and then checked that a report was generated and stored correctly. Integration tests check how application parts work together - the backend storing correctly to the database and triggering the ML process without mocking, for example. System testing was mostly manual QA done on the deployed app verifying that UI flows and detection results behaved as expected. We also incorporated several quality and automation tools-linters (ESLint for JS, pylint for Python) to check for style and CI/CD pipelines (GitHub Actions being one) to run the test suites on every commit. We did both white-box testing, where tests are constructed with knowledge of code internals, and black-box testing whereby we verify the functional requirements without regard to implementation details. The combination of unit and integration tests enabled us to verify the work of units in isolation and a system as a whole as for compliance with

In brief, Chapter 5 walks through the way in which modern web frameworks (React, Django) and modern testing methodologies (unit vs. integration tests) were used in the implementation and validation of the Project Supervision Tool. These are common themes throughout the world of web development, which help ensure an application is dependable and maintainable.